Hierarchical Classifiers for Complex Spatio-temporal Concepts

Jan G. Bazan

Chair of Computer Science, University of Rzeszów
Rejtana 16A, 35-310 Rzeszów, Poland
bazan@univ.rzeszow.pl

Abstract. The aim of the paper is to present rough set methods of constructing hierarchical classifiers for approximation of complex concepts. Classifiers are constructed on the basis of experimental data sets and domain knowledge that are mainly represented by concept ontology. Information systems, decision tables and decision rules are basic tools for modeling and constructing such classifiers. The general methodology presented here is applied to approximate spatial complex concepts and spatio-temporal complex concepts defined for (un)structured complex objects, to identify the behavioral patterns of complex objects, and to the automated behavior planning for such objects when the states of objects are represented by spatio-temporal concepts requiring approximation. We describe the results of computer experiments performed on real-life data sets from a vehicular traffic simulator and on medical data concerning the infant respiratory failure.

Keywords: rough set, concept approximation, complex dynamical system, ontology of concepts, behavioral pattern identification, automated planning.

1 Introduction

Reasoning based on concepts constitutes one of the main elements of a thinking process because it is closely related to the skill of categorization and classification of objects. The term *concept* means *mental picture of a group of objects* (see [1]). While the term *conceptualize* is commonly understood to mean *form a concept or idea about something* (see [1]). In the context of this work, there is interest in classifying conceptualized sets of objects. Concepts themselves provide a means of *describing* (forming a mental picture of) sets of objects (for a similar understanding the term *concept*, see, *e.g.*, [2, 3, 4]).

Definability of concepts is a term well-known in classical logic (see, *e.g.*, [5]). Yet in numerous applications, the concepts of interest may only be defined approximately on the basis of available, incomplete information about them (represented, *e.g.*, by positive and negative examples) and selected primary concepts and methods for creating new concepts out of them. It brings about the necessity to work out approximate reasoning methods based on inductive reasoning (see, *e.g.*, [6, 7, 8, 9, 10, 11, 12, 13]).

J.F. Peters et al. (Eds.): Transactions on Rough Sets IX, LNCS 5390, pp. 474–750, 2008.

In machine learning, this issue is known under the term *learning concepts by examples* (see, *e.g.*, [10]). The main problem of learning concepts by examples is that the description of a concept under examination needs to be created on the basis of known examples of that concept. By creating a concept description we understand detection of such properties of exemplary objects belonging to this concept that enable further examination of examples in terms of their membership in the concept under examination. A natural way to solve the problem of learning concepts by examples is inductive reasoning which means that while obtaining further examples of objects belonging to the concept (the so-called positive examples) and examples of objects not belonging to the concept (the so-called negative examples), an attempt is made to find such a description that correctly matches all or almost all examples of the concept under examination. Moreover, instead of speaking of learning concepts by examples, one may consider a more general learning of the so-called classifications which are partitions of all examples into a family of concepts (called decision classes) creating a partition of the object universe. A description of such a classification makes it possible to recognize the decision that should be made about examples unknown so far; that is, it gives us the answer as to what decision should be made that also includes examples not occurring in the process of classification learning.

Classifiers also known in literature as *decision algorithms*, *classifying algorithms* or *learning algorithms* may be treated as constructive, approximate descriptions of concepts (decision classes). These algorithms constitute the kernel of *decision systems* that are widely applied in solving many problems occurring in such domains as *pattern recognition*, *machine learning*, *expert systems*, *data mining* and *knowledge discovery* (see, *e.g.*, [6, 8, 9, 10, 11, 12, 13]).

In literature there can be found descriptions of numerous approaches to constructing classifiers, which are based on such paradigms of machine learning theory as *classical and modern statistical methods* (see, *e.g.*, [11, 13]), *neural networks* (see, *e.g.*, [11, 13]), *decision trees* (see, *e.g.*, [11]), *decision rules* (see, *e.g.*, [10, 11]), and *inductive logic programming* (see, *e.g.*, [11]). Many of the approaches mentioned above resulted in decision systems intended for computer support of decision making (see, *e.g.*, [11]). An example of such a system is RSES (*Rough Set Exploration System* [14, 15]) which has been developed for over ten years and utilizes rough set theory, originated by Professor Zdzisław Pawlak (see [16, 17, 18]), in combination with Boolean reasoning (see [19, 20, 21]).

With the development of modern civilization, not only the scale of the data gathered but also the complexity of concepts and phenomena which they concern are increasing rapidly. This crucial data change has brought new challenges to work out new data mining methods. Particularly, data more and more often concerns complex processes which do not give in to classical modeling methods. Of such a form may be medical and financial data, data coming from vehicles monitoring, or data about the users gathered on the Internet. Exploration methods of such data are in the center of attention in many powerful research centers in the world, and at the same time detection of models of complex processes and their properties (patterns) from data is becoming more and more attractive for applications

(see, *e.g.*, [22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]). Making a progress in this field is extremely crucial, among other things, for the development of intelligent systems which support decision making on the basis of results of analysis of the available data sets. Therefore, working out methods of detection of process models and their properties from data and proving their effectiveness in different applications are of particular importance for the further development of decision supporting systems in many domains such as medicine, finance, industry, transport, telecommunication, and others.

However, in the last few years essential limitations have been discovered concerning the existing data mining methods for very large data sets regarding complex concepts, phenomena, or processes (see, *e.g.*, [41, 42, 43, 44, 45, 46]). A crucial limitation of the existing methods is, among other things, the fact that they do not support an effective approximation of complex concepts, that is, concepts whose approximation requires discovery of extremely complex patterns. Intuitively, such concepts are too far in the semantical sense from the available concepts, *e.g.*, sensory ones. As a consequence, the size of spaces which should be searched in order to find patterns crucial for approximation are so large that an effective search of these spaces very often becomes unfeasible using the existing methods and technology. Thus, as it turned out, the ambition to approximate complex concepts with high quality from available concepts (most often defined by sensor data) in a fully automatic way, realized by the existing systems and by most systems under construction, is a serious obstacle since the classifiers obtained are often of unsatisfactory quality.

Recently, it has been noticed in the literature (see, *e.g.*, [42, 47, 48, 49, 50, 51, 52]) that one of the challenges for data mining is discovery of methods linking detection of patterns and concepts with domain knowledge. The latter term denotes knowledge about concepts occurring in a given domain and various relations among them. This knowledge greatly exceeds the knowledge gathered in data sets; it is often represented in a natural language and usually acquired during a dialogue with an expert in a given domain. One of the ways to represent domain knowledge is to record it in the form of the so-called *concept ontology* where ontology is usually understood as a finite hierarchy of concepts and relations among them, linking concepts from different levels (see, *e.g.*, [53, 54]).

In the paper, we discuss methods for approximation of complex concepts in real-life projects. The reported research is closely related to such areas as *machine learning* and *data mining* (*feature selection and extraction* [55, 56, 57], *classifier construction* [9, 10, 11, 12], *analytical learning* and *explanation based learning* [12, 58, 59, 60, 61]), *temporal and spatio-temporal reasoning* [62, 63, 64], *hierarchical learning and modeling* [42, 52, 65, 66, 67, 68], *adaptive control* [67, 69], *automated planning* (hierarchical planning, reconstruction of plans, adaptive learning plans) [70, 71, 72, 73, 74, 75, 76], *rough sets* and *fuzzy sets* (*approximation of complex vague concepts*) [77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87], *granular computing* (searching for compound patterns) [88, 89, 90, 91], *complex adaptive systems* [92, 93, 94, 95, 96, 97], *autonomous multiagent systems* [98, 99, 100, 101]), *swarm systems* [102, 103, 104], *ontologies development* [53, 54, 105, 106, 107].

It is also worthwhile mentioning that the reported research is also closely related to the domain of clinical decision-support for medical diagnosis and therapy (see, *e.g.*, [108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121]). Many reported results in this domain can be characterized as methods for solving specific problems such as *temporal abstraction problem* [117, 120, 121] or *medical planning problem* [108, 111, 112, 119]). Many methods and algorithms proposed in this paper can be also used for solving such problems.

The main aim of the paper is to present the developed methods for approximation of complex vague concepts involved in specification of real-life problems and approximate reasoning used in solving these problems. However, methods presented in the paper are assuming that additional domain knowledge in the form of the concept ontology is given. Concepts from ontology are often vague and expressed in natural language. Approximation of ontology is used to create hints in searching for approximation of complex concepts from sensory (low level) data.

The need of use of a domain knowledge expressed in the form of a concept ontology can be noticed in intensively developing domains connected with analysis and data processing as in the case of *reinforcement learning* (see, *e.g.*, [12, 122, 123, 124]). In the latter field, methods of learning new strategies with reinforcement take into account concept ontologies obtained from an expert, with the help of which it is possible to construct an approximation of a function estimating the quality of actions performed. Similarly, in a *Service Oriented Architecture* (SOA) [47, 49], the distribution of varied Web Services can be performed with the use of a domain knowledge, expressed using a concept ontology.

There also appeared propositions (see, *e.g.*, [42, 51, 52]) that use domain knowledge to search for the approximation of complex concepts in a hierarchical way which would lead to hierarchical classifiers able to approximate complex concepts with the high quality, *e.g.*, by analogy to biological systems [42]. This idea can be also related to learning of complex (*e.g.*, nonlinear) functions for fusion of information from different sources [125]. Therefore, currently, the problem of construction of such hierarchical classifiers is fundamental for complex concepts approximation and its solution will be crucial for construction of many methods of intelligent data analysis. These are, for example,

- methods of classification of objects into *complex spatial concepts* which are semantically distant from sensor data, *e.g.*, these are concepts as *safe vehicle driving on a highway*, *hazardous arrangement of two cooperating robots which puts them both at risk of being damaged*,
- methods of classification of object to *complex spatio-temporal concepts* semantically distant from sensor data which require observation of single objects or many objects over a certain period of time (*e.g.*, *acceleration of a vehicle on the road*, *gradual decrease of a patient's body temperature*, *robot's backward movement while turning right*),
- methods of *behavioral pattern* or *high risk pattern* identification where these types of patterns may be treated as complex concepts representing dynamic properties of objects; such concepts are expressed in a natural language on a

high level of abstraction and describing specific behaviors of a single object (or many complex objects) over a certain period of time (*e.g.*, *overtaking one vehicle by another*, *a traffic jam*, *chasing one vehicle after another*, *behavior of a patient under a high life threat*, *ineffective cooperation of a robot team*)
- methods of automatic learning of plans of complex object behavior, where a plan may be treated as a complex value of the decision which needs to be made for complex objects such as *vehicles*, *robots*, *groups of vehicles*, *teams of robots*, or *patients undergoing treatment*.

In the paper, we propose to link automatic methods of complex concept learning, and models of detection of processes and their properties with domain knowledge obtained in a dialogue with an expert. Interaction with a domain expert facilitates guiding the process of discovery of patterns and models of processes and makes the process computationally feasible. Thus presentation of new approximation methods of complex concepts based on experimental data and domain knowledge, represented using ontology concepts, is the main aim of this paper. In our opinion, the presented methods are useful for solving typical problems appearing when modeling complex dynamical systems.

1.1 Complex Dynamical Systems

When modeling complex real-world phenomena and processes mentioned above and solving problems under conditions that require an access to various distributed data and knowledge sources, the so-called *complex dynamical systems* (CDS) are often applied (see, *e.g.*, [92, 93, 94, 95, 96, 97]), or putting it in other way *autonomous multiagent systems* (see, *e.g.*, [98, 99, 100, 101]) or *swarm systems* (see, *e.g.*, [104]). These are collections of complex interacting objects characterized by constant change of parameters of their components over time, numerous relationships between the objects, the possibility of cooperation/competition among the objects and the ability of objects to perform more or less compound actions. Examples of such systems are *traffic*, *a patient observed during treatment*, *a team of robots during performing some task*, etc.

It is also worthwhile mentioning that the description of a CDS dynamics is often not possible with purely analytical methods as it includes many complex vague concepts (see, *e.g.*, [126, 127, 128]). Such concepts concern properties of chosen fragments of the CDS and may be treated as more or less complex objects occurring in the CDS. Hence, are needed appropriate methods of extracting such fragments that are sufficient to conclude about the global state of the CDS in the context of the analyzed types of changes and behaviors. In this approach, the CDS state is described by providing information about the membership of the complex objects isolated from the CDS in the complex concepts already established, describing properties of complex objects and relations among these objects. Apart from that, the description of the CDS dynamics requires following changes of the CDS state in time which leads to the so-called *trajectory* (history), that is, sequences of the CDS states observed over a certain period of time. Therefore, there are also needed methods for following changes of the selected

fragments of the CDS and changes of relations between the extracted fragments. In this paper, we use complex spatio-temporal concepts concerning properties, describing the dynamics of complex objects occurring in CDSs, to represent and monitor such changes. They are expressed in natural language on a much higher level of abstraction than so-called sensor data, so far mostly applied to the approximation of concepts. Examples of such concepts are *safe car driving*, *safe overtaking*, *patient's behavior when faced with a life threat*, *ineffective behavior of robot team*.

However, the identification of complex spatio-temporal concepts and using them to monitor a CDS requires approximation of these concepts. In this paper, we propose to approximate complex spatio-temporal concepts by hierarchical classifiers mentioned above and based on data sets and domain knowledge.

1.2 Problems in Modeling Complex Dynamical Systems

In modeling complex dynamical systems there appear many problems related to approximation of complex concepts used to describe the dynamics of the systems. One of these problems is obviously the problem of the gap between complex concepts and sensor data mentioned above. Apart from that, a series of other problems may be formulated whose solution is very important for complex concepts approximation and for complex dynamical systems monitoring. Below, we present a list of such problems including particularly those whose solution is the aim of this paper.

1. Problem of the gap between complex concepts and sensor data preventing an effective direct usage of sensor data to induce approximation of complex concepts by fully automatic methods.
2. Problem of complex concept stratification in classifier construction.
3. Problem of identification of behavioral patterns of complex objects in complex dynamical systems monitoring.
4. Problem of context of complex object parts while complex dynamical systems monitoring.
5. Problem of time speed-up in identification of behavioral patterns.
6. Problem of automated planning of complex object behavior when the object states are represented by complex concepts requiring approximation.
7. Problem of solving conflicts between actions in automated planning of complex object behavior.
8. Problem of synchronization of plans constructed for parts of a structured complex object.
9. Problem of plan adaptation.
10. Problem of similarity relation approximation between complex objects, complex object states, and complex object behavioral plans using data sets and domain knowledge.

In further subsections, a brief overview of the problems mentioned above is presented.

Problem of the Gap between Complex Concepts and Sensor Data. As we mentioned before, in spatio-temporal complex concepts approximation using sensor data, there occur major difficulties resulting from the fact that between spatio-temporal complex concepts and sensor data, there exists a gap which prevents an effective direct usage of sensor data for approximation of complex concepts. Therefore, in the paper we propose to fill the gap using domain knowledge represented mainly by a concept ontology and data sets chosen appropriately for this ontology (see Section 1.3).

Problem of Complex Concept Stratification. When we create classifiers for concepts on the basis of uncertain and imprecise data and knowledge semantically distant from the concepts under approximation, it is frequently not possible to construct a classifier which decisively classifies objects, unknown during classifier learning, to the concept or its complement. There appears a need to construct such classifiers that, instead of stating clearly about the object under testing whether it belongs to the concept or not, allow us to obtain only a certain type of membership degree of the object under testing to the concept. In other words, we would like to determine, with regards to the object under testing, how certain the fact that this object belongs to the concept is. Let us notice that this type of mechanism stratifies concepts under approximation, that is, divides objects under testing into layers labeled with individual values of membership degree to the concept. Such a mechanism can be obtained using different kinds of probability distributions (see [6, 43]). However, in this paper, instead of learning of a probability distribution we learn layers of concepts relevant for construction of classifiers. We call such classifiers as stratifying classifiers and we present two methods of a stratifying classifier construction (see Section 1.3). Our approach is inspired by papers about linguistic variables written by Professor Lotfi Zadeh (see [129, 130, 131]).

Problem of Identifying Behavioral Patterns. The study of collective behavior in complex dynamical systems is now one of the more challenging research problems (see, *e.g.*, [93, 99, 100, 102, 104, 132, 133, 134]), especially if one considers the introduction of some form of learning by cooperating agents (see, *e.g.*, [103, 122, 123, 124, 135, 136, 137]). For example, an efficient complex dynamical systems monitoring very often requires the identification of the so-called *behavioral patterns* or a specific type of such patterns called *high-risk patterns* or *emergent patterns* (see, *e.g.*, [93, 99, 100, 132, 138, 139, 140, 141, 142, 143, 144]). They are complex concepts concerning dynamic properties of complex objects expressed in a natural language on a high level of abstraction and describing specific behaviors of these objects. Examples of behavioral patterns may be: *overtaking one vehicle by another vehicle*, *driving a group of vehicles in a traffic jam*, *behavior of a patient under a high life threat*, etc. These types of concepts are difficult to identify automatically because they require watching complex object behavior over longer period of time and this watching usually is based on the identification of a sequence of less complex spatio-temporal concepts. Moreover, a crucial role

for identification of a given behavioral pattern is played by the sequence of less complex concepts which identify it. For example, in order to identify the behavioral pattern of *overtaking one vehicle by another*, it should first be determined whether the overtaking vehicle approaches the overtaken vehicle; next, whether the overtaking vehicle changes lanes appropriately and overtakes the vehicle; and finally, to determine that the overtaking vehicle returns to the previous lane driving in front of the overtaken vehicle. The methodology of a dynamical system modeling proposed in the paper enables approximation of behavioral patterns on the basis of data sets and domain knowledge expressed using a concept ontology (see Section 1.3).

Problem of Context for Complex Object Parts. In this paper, any complex dynamical system (CDS) is represented using descriptions of its global states or trajectories (histories), that is, sequences of CDS states observed over a certain period of time (see, *e.g.*, [145, 146, 147, 148, 149, 150, 151, 152] and Section 1.1). Properties of such states or trajectories are often dependent on specific parts of these states or trajectories. This requires to consider the relevant structure of states or trajectories making it possible to extract parts and the relevant context of parts. Moreover, each structured object occurring in a complex dynamical system is understood as a set of parts extracted from states or trajectories of a given complex dynamical system. Such parts are often related by relations representing links or interactions between parts. That is why both learning of the behavioral patterns concerning structured objects and the identification of such patterns, in relation to specific structured objects, requires the isolation of structured objects as sets of potential parts of such objects, that is, object sets of lesser complexity. The elementary approach to isolate structured objects consisting in examination of all possible subsets (of an established size) of the set of potential parts of structured objects cannot be applied because of potentially high number of such subsets. For example, during an observation of a highway from a helicopter (see, *e.g.*, [89, 153]), in order to identify a group of vehicles which are involved in the maneuver of dangerous overtaking, it would be necessary to follow (in the real time) the behavior of all possible groups of vehicles of an established size (*e.g.*, six vehicles, see Appendix A) that may be involved in this maneuver, which already with a relatively small number of visible vehicles becomes computationally too difficult.

Another possibility is the application of methods which use the context in which the objects being parts of structured objects occur. This type of methods isolate structured objects not by a direct indication of the set of parts of the searched structured object but by establishing one part of the searched structured object and attaching to it other parts, being in the same context as the established part. Unfortunately, also here, the elementary approach to determination of the context of the part of the structured object, consisting in examination of all possible subsets (of an established size) of the set of potential structured objects to which the established part of the structured object belongs, cannot be applied because of a large number of such subsets. For example, in order to identify a group of vehicles which are involved in a dangerous maneuver

and to which the vehicle under observation belongs, it would be necessary to follow (in the real time) the behavior of the possible groups of vehicles of an established size (*e.g.*, six vehicles, see Appendix A) to which the vehicle considered belongs, which is, with a relatively small number of visible vehicles, still computationally too difficult. Therefore, there are needed special methods of determining the context of the established part of the structured object based on a domain knowledge which enable to limit the number of analyzed sets of parts of structured objects. In the paper, we propose the so-called *sweeping method* which enables fast determination of the context of the established object treated as one of the parts of the structured object (see Section 1.3).

Problem of Time Speed-Up in Identification of Behavioral Patterns. Identification of a behavioral pattern in relation to a specific complex object may be performed by observing the behavior of these objects over a certain period of time. Attempts to shorten this time are usually inadvisable, because they may cause false identification of behavioral pattern in relation to some complex objects. However, in many applications there exists a need for a fast decision making (often in the real time) about whether or not a given object matches the established behavioral pattern. It is extremely crucial in terms of computational complexity because it enables a rapid elimination of these complex objects which certainly do not match the pattern. Therefore, in the paper, there is presented a method of elimination of complex objects in identification of a behavioral pattern, which is based on *the rules of fast elimination of behavioral patterns* which are determined on the basis of data sets and domain knowledge (see Section 1.3).

Problem of Automated Planning. In monitoring the behavior of complex dynamical systems (*e.g.*, by means of behavioral patterns identification) there may appear a need to apply methods of automated planning of complex object behavior. For example, if during observation of a complex dynamical system, a behavioral pattern describing inconvenient or unsafe behavior of a complex object (i.e., a part of system state or trajectory) is identified, then the system control module may try, using appropriate actions, to change the behavior of this object in such a way as to lead the object out of the inconvenient or unsafe situation. However, this type of short-term interventions may not be sufficient to lead the object out of the undesired situation permanently. Therefore, a possibility of automated planning is often considered which means construction of sequences of actions alternately with states (of plans) to be performed by the complex object or on the complex object in order to bring it to a specific state. In literature, there may be found descriptions of many automated planning methods (see, *e.g.*, [70, 71, 72, 73, 74, 75, 76]). However, applying the latter approaches, it has to be assumed that the current complex object state is known which results from a simple analysis of current values of available parameters of this object. Meanwhile, in complex dynamical systems, a complex object state is often described in a natural language using vague spatio-temporal conditions whose satisfiability cannot be tested on the basis of a simple analysis of available information about the object. For example, when planning the treatment of an infant suffering from

the respiratory failure, the infant's condition may be described by the following condition:

– *Patient with RDS type IV, persistent PDA and sepsis with mild internal organs involvement* (see Appendix B for mor medical details).

Stating the fact that a given patient is in the above condition requires an analysis of examination results of this patient registered over a certain period of time with a large support of a domain knowledge provided by experts (medical doctors). This type of conditions may be represented using complex spatio-temporal concepts. Identification of these conditions requires, however, an approximation of the concepts representing them with the help of classifiers. Therefore, in the paper, we describe automated planning methods of behavior of complex objects whose states are described using complex concepts requiring approximation (see Section 1.3).

Problem of Solving Conflicts between Actions. In automated planning methods, during a plan construction there usually appears a problem of nondeterministic choice of one action possible to apply in a given state. Therefore, usually there may be many solutions to a given planning problem consisting in bringing a complex object from the initial state to the final one using different plans. Meanwhile, in practical applications there often appears a situation that the automatically generated plan must be compatible with the plan proposed by the expert (*e.g.*, the treatment plan should be compatible with the plan proposed by human experts from a medical clinic). Hence, we inevitably need tools which may be used during a plan generation to solve the conflicts appearing between actions which may be performed at a given planning state. It also concerns making the decision about what state results from the action performed. That is why, in the paper, we propose a method which indicates the action to be performed in a given state or shows the state which is the result of the indicated action. This method uses a special classifier constructed on the basis of data sets and domain knowledge (see Section 1.3).

Problem of Synchronizing Plans. In planning the behavior of structurally complex objects consisting of parts being objects of lesser complexity, it is often not possible to plan effectively the behavior of a whole such object. That is why, in such cases the behavior of all parts is usually planned separately. However, such an approach to behavior planning for a complex object requires plan synchronization constructed for individual parts in such a way as not to make these plans contradicting one to another but be complement in order to plan the best behavior for the whole complex object. For example, treatment of a certain illness A, which is the result of illnesses B and C requires such a treatment planning of illnesses B and C so as not to make their treatments contradictory, but to make them to support and to complement one another during treatment of illness A. In the paper, *a planning synchronization method* for parts of a complex object is presented. It uses two classifiers constructed on the basis of data sets and domain knowledge (see Section 1.3). If we treat plans constructed for parts

of a structured object as processes of some kind, then the method of synchronizing those plans is a method of synchronization of processes corresponding to the parts of a structured object. It should be emphasized, however, that the significant novelty of the method of synchronization of processes presented herein in relation to the ones known from literature (see, *e.g.*, [154, 155, 156, 157, 158, 159]) is the fact that the synchronization is carried out by using classifiers determined on the basis of data sets and domain knowledge.

Plan Adaptation Problem. After constructing a plan for a complex object, the execution of this plan may take place. However, the execution of the whole plan is not always possible in practice. It may happen that, during the plan execution such a state of complex object occurred that is not compatible with the state predicted by the plan. Then, the question arises whether the plan should still be executed or whether it should be reconstructed (updated).

If the current complex object state differs slightly from the state expected by the plan, then the execution of the current plan may perhaps be continued. If, however, the current state differs significantly from the state from the plan, then the current plan has to be reconstructed. It would seem that the easiest way to reconstruct the plan is construction of a new plan which commences at the current state of the complex object and ends at the final state of the old plan (a total reconstruction of the plan). However, in practical applications, a total reconstruction can be too costly in terms of computation or resources. Therefore, we need other methods which can effectively reconstruct the original plan in such a way as to realize it at least partially. Hence, in the paper, we propose a method of plan reconstruction called *a partial reconstruction*. It consists of constructing a short so-called *repair plan* which quickly brings the complex object to the so-called *return state* from the current plan. Next, on the basis of the repair plan, a reconstruction of the current plan is performed by replacing its fragment beginning with the current state and ending with the return state of the repair plan (see Section 1.3).

It is worth noticing that this issue is related to the domain of artificial intelligence called *the reasoning about changes* (see, *e.g.*, [160, 161]). Research works in this domain very often concern construction of a method of concluding about changes in satisfiability of concepts on a higher level of a certain concept hierarchy as a basis for discovery of plans aimed at restoration of the satisfiability of the desired concepts on a lower level of this hierarchy.

Problem of Similarity Relation Approximation. In building classifiers approximating complex spatio-temporal concepts, there may appear a need to estimate the similarity or the difference of two elements of a similar type such as complex objects, complex object states or plans generated for complex objects. This is an example of a classical case of *the problem of defining similarity relation* (or perhaps defining *dissimilarity relation* complementary to it) which is still one of the greatest challenges of data mining and knowledge discovery. The existing methods of defining similarity relations are based on building similarity functions on the basis of simple strategies of fusion of local similarities of compared elements. Optimization of the similarity formula established is performed

by tuning both parameters of local similarities and their linking parameters (see, *e.g.*, [162, 163, 164, 165, 166, 167, 168, 169, 170, 171]). Frequently, however, experts from a given domain are not able to provide such a formula that would not raise their doubts and they limit themselves to the presentation of a set of examples of similarity function values, that is, a set of pairs of the compared elements labeled with degrees representing similarity function value. In this case, defining the similarity function requires its approximation with the help of a classifier, and at the same time such properties of compared elements should be defined that enable to approximate the similarity function. The main difficulty of the similarity function approximation is an appropriate choice of these properties. Meanwhile, according to the domain knowledge there are usually many various aspects of similarity between compared elements. For example, when comparing medical plans constructed for treatment of infants with a respiratory failure (see Appendix B), similarity of antibiotic therapies, similarity of applied mechanical ventilation methods, similarity of PDA closing and others should be taken into account. Each of these aspects should be considered in a specific way and presentation of formulas describing them can be extremely difficult for an expert. Frequently, an expert may only give examples of pairs of comparable elements together with their similarity in each of these aspects. Moreover, a fusion of different similarity aspects into a global similarity should also be performed in a way resulting from the domain knowledge. This way may be expressed, for example, using a concept ontology. In the paper, we propose *a method of similarity relation approximation* based on the usage of data sets and domain knowledge expressed, among other things, on the basis of a concept ontology (see Section 1.3).

1.3 Overview of the Results Achieved

As we mentioned before, the aim of this paper is to present a set of approximation methods of complex spatio-temporal concepts and approximate reasoning concerning these concepts, assuming that the information about concepts is given mainly in the form of a concept ontology.

The results described in the paper may be divided into the following groups:

1. methods for construction of classifiers stratifying a given concept,
2. general methodology of concept approximation with the usage of data sets and domain knowledge represented mainly in the form of a concept ontology,
3. methods for approximation of spatial concepts from an ontology,
4. methods for approximation of spatio-temporal concepts from an ontology defined for unstructured objects,
5. methods for approximation spatio-temporal concepts from an ontology defined for structured objects,
6. methods for behavioral pattern identification of complex objects in states of complex dynamical systems,

7. methods for automated planning of behavior of complex objects when the object states are represented by vague complex concepts requiring approximation,
8. implementation of all more crucial methods described in the paper as the RSES system extension.

In further subsections we briefly characterize the above groups of results.

At this point we present the publications on which the main results of our research have been partially based. The initial version of method for approximation of spatial concepts from an ontology was described in [172]. Methods for approximation of spatio-temporal concepts and methods for behavioral pattern identification were presented in [88, 173, 174, 175, 176, 177, 178]. Papers [173, 176, 177, 178] concern behaviors related to recognition of vehicle behavioral patterns or a group of vehicles on the road. The traffic simulator used to generate data for the needs of computer experiments was described in [179]. The paper [174] concerns medical applications related to recognition of high death risk pattern of infants suffering from respiratory failure, whereas papers [88, 175] concern both applications which were mentioned above. Finally, methods for automated planning of behavior of complex objects were described in [88, 180, 181].

Methods for Construction of Classifiers Stratifying Concepts. In practice, construction of classifiers often takes place on the basis of data sets containing uncertain and imprecise information (knowledge). That is why it is not often possible to construct a classifier which decisively classifies objects to the concept or its complement. This phenomenon occurs particularly when there is a need to classify objects not occurring in a learning set of objects, that is, those which are not used to construct the classifier.

One possible approach is to search for classifiers approximating probability distribution (see, *e.g.*, [6, 43]). However, in application, one may often require a less exact method based on classifying objects to different linguistic layers of the concept. This idea is inspired by papers of Professor Lotfi Zadeh (see, *e.g.*, [129, 130, 131]). In our approach, the discovered concept layers are used as patterns in searching for approximation of a more compound concept. In the paper, we present methods for construction of classifiers which, instead of stating clearly whether a tested object belongs to the concept or not, enable to obtain some membership degree of the tested object to the concept. In the paper, we define the concept of *a stratifying classifier* as a classifying algorithm stratifying concepts, that is, classifying objects to different concept layers (see Section 3). We propose two approaches to construction of these classifiers. One of them is *the expert approach* which is based on the defining, by an expert, an additional attribute in data which describes membership of the object to individual concept layers. Next, a classifier differentiating layers as decision classes is constructed. The second approach called *the automated approach* is based on the designing algorithms being the classifier extensions which enable to classify objects to concept layers on the basis of certain premises and experimental observations. In the paper, a new method of this type is proposed which is based on shortening of decision rules relatively to various coefficients of consistency.

General Methodology of Concept Approximation from Ontology. One of the main results presented in this paper is a methodology of approximating concepts from ontology. Generally, in order to approximate concepts a classical in machine learning [10] method of concept approximation is applied on the basis of positive and negative examples. It is based on the construction of a data table for each concept, known in rough set theory as a decision table (a special information system with a distinguished attribute called *decision* [16]) with rows (called objects) corresponding to positive and negative examples of the concept approximated and columns describing properties (features, attributes) of examples expressed by formulas in a considered language. The last column, called the decision column, is treated as a description of membership of individual examples to the concept approximated. For a table constructed in such a way, classifiers approximating a concept are built.

In such an approach, the main problem is the choice of examples of a given concept and properties of these examples.

The specificity of methodology of concept approximation proposed here in comparison with other methods (see, *e.g.*, [11, 52, 182]) is the usage of a domain knowledge expressed in the form of a concept ontology together with the rough set methods.

For concepts from the lowest level of an ontology hierarchy (the sensor level), not depending on the remaining concepts, we assume that so-called sensor attributes are also available which on the basis of given positive and negative examples, enable approximating these concepts by using classical methods of classifier construction.

However, the concept approximation methods, applied on a higher level of ontology consist in approximation of concepts using concepts from the lower ontology level. In this way, there are created hierarchical classifiers which use domain knowledge recorded in the form of ontology levels. In other words, patterns discovered for approximation of concepts on a given hierarchy level are used in construction of more compound patterns relevant for approximation of concepts on the next hierarchy level.

To approximate concepts from the higher ontology level, sensor attributes cannot be applied directly because the "semantical distance" of the higher level concepts from sensor attributes is too long and they are defined on different abstraction levels, i.e., searching for relevant features to approximate such concepts directly from sensory features becomes unfeasible (see the first problem from Section 1.2). For example, it is hardly believable that given only sensor attributes describing simple parameters of driving a vehicle (*e.g.*, location, speed, acceleration), one can approximate such a complex concept as *safe driving a vehicle*. Therefore, we propose a method, by means of which concepts from the higher ontology level exclusively be approximated by concepts from one level below. The proposed approach to concept approximation of a higher level is based on the assumption that the concept from the higher ontology level is semantically not too far from concepts lying on the lower level in the ontology. "Not too far" means that it may be expected that it is possible to approximate a concept

from the higher ontology level with the help of lower ontology level concepts and patterns used for or derived from their construction, for which classifiers have already been built.

If we assume that approximation of concepts on the higher ontology level takes place using lower level concepts, then according to an established concept approximation methodology, positive and negative examples of the concept approximated are needed as well as their properties which serve the purpose of approximation. However, because of the semantical differences between concepts on different ontology levels, mentioned above, examples of lower ontology level concepts cannot be directly used to approximate a higher ontology level concept. For example, if the concept of a higher level concerns a group of vehicles (*e.g.*, *driving in a traffic jam*, *chase of one vehicle after another*, *overtaking*), whereas the lower level concepts concern single vehicles (*e.g.*, *accelerating*, *decelerating*, *changing lanes*), then the properties of a single vehicle (defined in order to approximate lower ontology level concepts) are usually insufficient to describe the properties of the whole group of vehicles. Difficulties with concept approximation on the higher ontology level using examples of the lower ontology level also appear when on the higher ontology level there are concepts concerning a time period different than that one related to the concepts on the lower ontology level. For example, a higher level concept may concern a time window, that is, a certain period of time (*e.g.*, *vehicle acceleration*, *vehicle deceleration*), whereas the lower level concepts may concern a certain instant, that is, a time point (*e.g.*, *a small vehicle speed*, *location of vehicle in the right lane*).

Hence, we present a method for construction of positive and negative examples of a concept of a higher ontology level consisting, in a general case, in arrangement (putting together) sets of examples of concepts of the lower ontology level. At the same time we define and represent such sets using patterns expressed in languages describing properties of examples of concepts of lower level in the ontology. These sets (represented by patterns) are arranged according to the so-called constraints resulting from the domain knowledge and determining which sets (patterns) may be arranged and which cannot be arranged for the construction of examples of higher level concepts. Thus, object structures on higher hierarchical levels come into being through linking (with the consideration of certain constraints) of objects from lower levels (and more precisely sets of these objects described by patterns). Such an approach enables a gradual modeling properties of more and more complex objects. Starting with elementary objects, objects being their sets or sequences of such objects, sets of sequences, etc. are gradually modeled. Different languages expressing properties of, *e.g.*, elementary objects, object sequences, or sets of sequences correspond to different model levels.

A crucial innovation feature of methods presented here is the fact that to define patterns describing examples of a lower ontology level, classifiers constructed for these concepts are used.

The example construction process for higher ontology level concepts on the basis of lower level concepts proceeds in the following way. Objects which are positive and negative examples of lower ontology level concepts are elements of a

certain relational structure domain. Relations occurring in such a structure express relations between these objects and may be used to extract sets of objects of the lower ontology level. Each extracted set of objects is also a domain of a certain relational structure, in which relations are defined using information from a lower level. The process of extraction of relational structures is performed in order to approximate a higher ontology level concept with the help of lower ontology level concepts. Hence, to extract relational structures we necessarily need the information about membership of lower level objects to the concepts from this level. Such information may be available for any tested object based on the application of previously created classifiers for the lower ontology level concepts. Let us note that classifiers stratifying concepts are of a special importance here. The language in which we define formulas (patterns) to extract new relational structures using relational structures and lower ontology level concepts, is called the *language for extracting relational structures* (ERS-language).

For relational structures extracted in such a way, properties (attributes) may be defined which lead to an information system whose objects are extracted relational structures and the attributes are the properties of these structures (RS-information system). Relational structure properties may be defined using patterns which are formulas in a language specially constructed for this purpose, i.e., in *a language for definnig features of relational structures* (FRS-language). For example, some of the languages used to define the properties of extracted relational structures, presented in this paper, use elements of temporal logics with linear time, *e.g.*, *Linear Temporal Logic* (see, *e.g.*, [183, 184, 185]).

Objects of RS-information system are often inappropriate to make their properties relevant for the approximation of the higher ontology level concepts. It is due to the fact that there are too many such objects and their descriptions are too detailed. Hence, when applied to the higher ontology level concept approximation, the extension of the created classifier would be too low, that is, the classifier would classify too small number of tested objects. Apart from that, the problem of computational complexity would appear which means that because of a large number of objects in such information systems, the number of objects in a linking table, constructed in order to approximate concepts determined in a set of objects of a complex structure, would be too large to construct a classifier effectively (see below).

That is why a grouping (clustering) of such objects is applied which leads to obtaining more general objects, i.e., clusters of relational structures. This grouping may take place using a language chosen by an expert and called the *language for extracting clusters of relational structures* ($ECRS$-language). Within this language, a family of patterns may be selected to extract relevant clusters of relational structures from the initial information system.

For the clusters of relational structures obtained, an information system may be constructed whose objects are clusters defined by patterns from this family, and the attributes are the properties of these clusters. The properties of these clusters may be defined by patterns which are formulas of a language specially constructed for this purpose, i.e., *a language for defining features of clusters*

of relational structures ($FCRS$-language). For example, some of the languages assigned to define the properties of relational structure clusters presented in this paper use elements of temporal logics with branching time, *e.g.*, *Branching Temporal Logic* (see, *e.g.*, [183, 184, 185]).

The information system with objects which are clusters of relational structures (CRS-information system) may already be used to approximate the concept of the higher ontology level. In order to do this, a new attribute is added to the system by the expert informs about membership of individual clusters to the concept approximated, and owing to that we obtain an approximation table of a higher ontology concept.

The method of construction of the approximation table of a higher ontology level concept may be generalized for concepts determined on a set of structured objects, that is, ones consisting of a set of parts (*e.g.*, *a group of vehicles on the road*, *a group of interacting illnesses*, *a robot team performing a task together*). This generalization means that CRS-information systems constructed for individual parts may be linked in order to obtain an approximation table of a higher ontology level concept determined for structured objects. Objects of this table are obtained through an arrangement (linking) of all possible objects of linked information systems. From the mathematical point of view this assumption is a Cartesian product of sets of objects of linked information systems. However, in terms of domain knowledge not all object links belonging to such a Cartesian product are possible (see [78, 84, 186, 187]). For example, if we approximate the concept of *safe overtaking*, it makes sense to arrange objects concerning only such vehicle pairs which are in the process of the overtaking maneuver.

For the reason mentioned above, that is, elimination of unrealistic complexes of objects, the so-called constraints are defined that are formulas built on the basis of arranged object features. The constraints determine which objects may be arranged in order to obtain an example of an object from a higher level and which may not. Additionally, we assume that to each arrangement allowed by the constraints, the expert adds a decision value informing whether a given arrangement belongs ore does not belong to the approximated concept of a higher level.

The table constructed in such a way serves the purpose of the approximation of a concept describing structured objects. However, in order to approximate a concept concerning structured objects, it is often necessary to construct not only all parts of the structured object but also features describing relations between parts. For example, *driving one vehicle after another*, apart from features describing the behavior of those two vehicles separately, features describing the location of these vehicles in relation to one another as well ought to be constructed. That is why in construction of a table of concept approximation for structured objects, there is constructed an additional CRS-information system whose attributes entirely describe the whole structured object in terms of relations between the parts of this object. In approximation of the object concerning structured objects, this system is arranged together with other CRS-information systems constructed for individual parts of the structured objects.

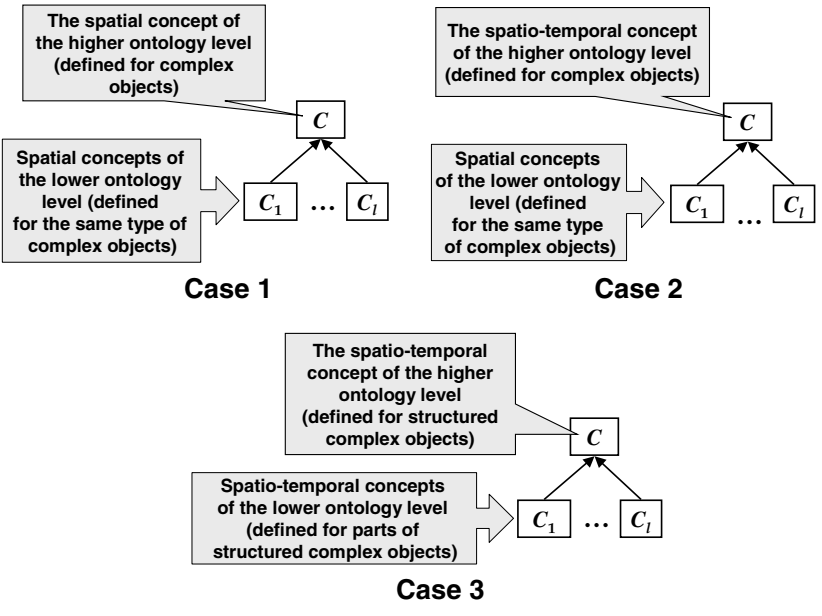


Fig. 1. Three cases of complex concepts approximation in ontology

A fundamental problem in construction of an approximation table of a higher ontology level concept is, therefore, the choice of four appropriate languages used during its construction. The first language serves the purpose of defining patterns in a set of examples of a concept of lower ontology level which enable the relational structure extraction. The second one enables to define the properties of these structures. The third one makes possible to define relational structure clusters and, finally, the fourth one, the properties of these clusters. All these languages must be defined in such a way as to make the properties of the created relational structure clusters useful on a higher ontology level for approximation of the concept occurring there. Moreover, when the approximated concept concerns structured objects, each of the parts of this type of objects may require another four the languages similar to those already mentioned above.

Definitions of the above four languages depends on the semantical difference between concepts from both ontology levels. In the paper, the above methodology is applied in the three following cases in which the above four languages are defined in a completely different way:

1. The concept of the higher ontology level is a spatial concept (it does not require observing changes of objects over time) and it is defined on the set of the same objects (examples) as concepts of the lower ontology level, and at the same time the lower ontology level concepts are also spatial concepts (see Case 1 from Fig. 1).

2. The concept of the higher ontology level is a spatio-temporal concept (it requires observing object changes over time) and it is defined on a set of the same objects (examples) as the lower ontology level concepts. Moreover, the lower ontology level concepts are spatial concepts exclusively (see Case 2 from Fig. 1).
3. The concept of the higher ontology level is a spatio-temporal concept defined on a set of objects which are structured objects in relation to objects (examples) of the lower ontology level concepts, that is, the lower ontology level objects are parts of objects from the higher ontology level. Additionally, and at the same time the lower ontology level concepts are also spatio-temporal concepts (see Case 3 from Fig. 1).

Methods described in the next three subsections concern the above three cases. These methods also found application in construction of methods of behavioral pattern identification and in automated planning.

Methods of Approximation of Spatial Concepts. In the paper, the method of approximating concepts from ontology is proposed when a higher ontology level concept is a spatial concept (not requiring an observation of changes over time) and it is defined on a set of the same objects (examples) as the lower ontology level concepts; at the same time, the lower level concepts are also spatial concepts. An exemplary situation of this type is an approximation of the concept of *Safe overtaking* (concerning single vehicles on the road) using concepts such as *Safe distance from the opposite vehicle during overtaking*, *Possibility of going back to the right lane* and *Possibility of safe stopping before the crossroads*.

The concept approximation method described in this subsection is an example of the general methodology of approximating concepts from ontology described previously. That is why its specificity is the domain knowledge usage expressed in the form of a concept ontology and application of rough set methods, mainly in terms of application of classifier construction methods.

The basic terms used in the presented method is *pattern* and *production rule*. Patterns are descriptions of examples of concepts from an ontology and they are constructed by classifiers stratifying these concepts. A production rule is a decision rule which is constructed on two adjacent levels of ontology. In the *predecessor* of this rule there are patterns for the concepts from the lower level of the ontology whereas in the *successor*, there is a pattern for one concept from the higher level of the ontology (connected with concepts from the rule predecessor) where both patterns from the predecessor and the successor of the rule are chosen from patterns constructed earlier for concepts from both adjacent levels of the ontology. A rule constructed in such a way may serve as a simple classifier or an argument "for"/"against" the given concept, enabling classification of objects which match the patterns from the rule predecessor with the pattern from the rule successor. In the paper, there is proposed an algorithmic method of induction of production rules, consisting in an appropriate search for data tables with attributes describing the membership of training objects to particular layers of concepts (see Section 5.4). These tables are constructed using the so-called constraints between concepts thanks to which the information put in the tables

only concerns those objects/examples which might be found there according to the production rule under construction.

Although a single production rule may be used as a classifier for the concept appearing in a rule successor, it is not a complete classifier yet, i.e., classifying all objects belonging to an approximated concept and not only those matching patterns of a rule predecessor. Therefore, in practice, production rules are grouped into the so-called *productions* (see Section 5.3), i.e., production rule collections, in a way that each production contains rules having patterns for the same concepts in a predecessor and the successor, but responding to their different layers. Such production is able to classify much more objects than a single production rule where these objects are classified into different layers of the concept occurring in a rule successor. Both productions and production rules themselves are only constructed for the two adjacent levels of ontology. Therefore, in order to use the whole ontology fully, there are constructed the so-called AR-schemes, i.e., *approximate reasoning schemes* (see, *e.g.*, [77, 89, 172, 188, 189, 190, 191, 192, 193, 194]) which are hierarchical compositions of production rules (see Section 5.7). The synthesis of an AR-scheme is carried out in a way that to a particular production from a lower hierarchical level of the AR-scheme under construction another production rule on a higher level may be attached, but only that one where one of the concepts for which the pattern occurring in the predecessor was constructed is the concept connected with the rule successor from the previous level. Additionally, it is required that the pattern occurring in a rule predecessor from the higher level is a subset of the pattern occurring in a rule successor from the lower level (in the sense of inclusion of object sets matching both patterns). To the two combined production rules other production rules can be attached (from above, from below or from the side) and in this way a multilevel structure is made which is a composition of many production rules. The AR-scheme constructed in such a way can be used as a hierarchical classifier whose entrance are predecessors of production rules from the lowest part of the AR-scheme hierarchy and the exit is the successor of a rule from the highest part of the AR-scheme hierarchy. That way, each AR-scheme is a classifier for a concept occurring in the rule successor from the highest part in the hierarchy of the scheme and, to be precise, for a concept for which a pattern occurring in the rule successor from the highest part in the hierarchy of the AR-scheme is determined.

However, similarly to the case of a single production rule, an AR-scheme is not a full classifier yet. That is why, in practice, for a particular concept there are many AR-schemes constructed which approximate different layers or concept regions.

In this paper, there are proposed two approaches for constructing AR-schemes (see Section 5.7). The first approach is based on memory with AR-schemes and consists in building many AR-schemes after determining production, which later on are stored and used for the classification of tested objects.

The second approach is based on a dynamic construction of AR-schemes. It is realized in a way that during classification of a given tested object, an

appropriate AR-schemes for classifying this particular object is built on the basis of a given collection of productions ("lazy" classification).

In order to test the quality and effectiveness of classifier construction methods based on AR-schemes, experiments on data generated from the traffic simulator were performed (see Section 5.8). The experiments showed that classification quality obtained through classifiers based on AR-schemes is higher than classification quality obtained through traditional classifiers based on decision rules. Apart from that, the time spent on classifier construction based on AR-schemes is shorter than when constructing classical rule classifiers, their structure is less complicated than that of classical rule classifiers (a considerably smaller average number of decision rules), and their performance is much more stable because of the differences in data in samples supplied for learning (*e.g.*, to change the simulation scenario).

Methods of Approximation of Spatio-temporal Concepts. We also propose a method of approximating concepts from ontology when a higher ontology level concept is a spatio-temporal concept (it requires observing changes of complex objects over time) defined on a set of the same objects as the lower ontology level concepts; at the same time, the lower ontology level concepts are spatial concepts only. This case concerns a situation when during an observation of a single object in order to capture its behavior described by a higher ontology level concept, we have to observe it longer than it requires to capture behaviors described by lower ontology level concepts. For example, lower ontology level concepts may concern simple vehicle behaviors such as *small increase in speed*, *small decrease in speed* or *small move towards the left lane*. However, the higher ontology level concept may be a more complex concept as, *e.g.*, *acceleration in the right lane*. Let us notice that determining whether a vehicle accelerates in the right lane requires its observation for some time called *a time window*. On the other hand, determining whether a vehicle speed increases in the right lane requires only a registration of the speed of a vehicle in two neighboring instants (time points) only. That is why spatio-temporal concepts are more difficult to approximate than spatial concepts whose approximation does not require observing changes of objects over time.

Similarly to spatial concept approximation (see above), the method of concept approximation described in this subsection is an example of the general methodology of approximating concepts from ontology described earlier. Its specificity is, therefore, the domain knowledge usage expressed in the form of a concept ontology and rough set method application, mainly in terms of application of classifier construction methods. However, in this case more complex ontologies are used, and they contain both spatial and spatio-temporal concepts.

The starting point for the method proposed is a remark that spatio-temporal concept identification requires an observation of a complex object over a longer period of time called *a time window* (see Section 6.4). To describe complex object changes in the time window, the so-called *temporal patterns* (see Section 6.6) are used, which are defined as functions determined on a given time window. These patterns, being in fact formulas from a certain language, also characterize

certain spatial properties of the complex object examined, observed in a given time window. They are constructed using lower ontology level concepts and that is why identification whether the object belongs to these patterns requires the application of classifiers constructed for concepts of the lower ontology level.

On a slightly higher abstraction level, the spatio-temporal concepts (also called *temporal concepts*) are directly used to describe complex object behaviors (see Section 6.5). Those concepts are defined by an expert in a natural language and they are usually formulated using questions about the current status of spatio-temporal objects, *e.g.*, *Does the vehicle examined accelerate in the right lane?*, *Does the vehicle maintain a constant speed during lane changing?* The method proposed here is based on approximating temporal concepts using temporal patterns with the help of classifiers. In order to do this a special decision table is constructed called *a temporal concept table* (see Section 6.9). The rows of this table represent the parameter vectors of lower level ontology concepts observed in a time window (and, more precisely, clusters of such parameter vectors). Columns of this table (apart from the last one) are determined using temporal patterns. However, the last column represents membership of an object, described by parameters (features, attributes) from a given row, to the approximated temporal concept.

Temporal concepts may be treated as nodes of a certain directed graph which is called a *behavioral graph*. Links (directed edges) in this graph are the temporal relations between temporal concepts meaning a temporal sequence of satisfying two temporal concepts one after another. These graphs are of a great significance in complex objects approximation for structured objects (see below).

Methods of Approximation of Spatio-temporal Concepts for Structured Objects. The method of spatio-temporal concept approximation presented in the previous subsection is extended to the case when higher ontology level concepts are defined on a set of objects which are structured objects in relation to objects (examples) of the lower ontology level concepts, that is, the lower ontology level objects are parts of objects from the higher ontology level. Moreover, lower ontology level concepts are also spatio-temporal concepts. This case concerns a situation when during a structured object observation, which serves the purpose of capturing its behavior described by a higher ontology level concept, we must observe this object longer than it is required to capture the behavior of a single part of the structured object described by lower ontology level concepts. For example, lower ontology level concepts may concern complex behaviors of a single vehicle such as *acceleration in the right lane*, *acceleration and changing lanes from right to left*, *decelerating in the left lane*. However, a higher ontology level concept may be an even more complex concept describing behavior of a structured object consisting of two vehicles (the overtaking and the overtaken one) over a certain period of time, for example, *the overtaking vehicle changes lanes from right to left, whereas the overtaken vehicle drives in the right lane*. Let us notice that the behavior described by this concept is a crucial fragment of the overtaking maneuver and determining whether the observed group of two vehicles behaved exactly that way, requires observing a sequence of

behaviors of vehicles taking part in this maneuver for a certain period of time. They may be: *acceleration in the right lane*, *acceleration and changing lanes from right to left*, *maintaining a stable speed in the right lane*.

Analogously to the case of spatial and spatio-temporal concept approximation for unstructured objects, the method of concept approximation described in this subsection is an example of the general methodology of approximating concepts from ontology described previously. Hence, its specificity is also the domain knowledge usage expressed in the form of a concept ontology and rough set methods. However, in this case, ontologies may be extremely complex, containing concepts concerning unstructured objects, concepts concerning structured objects as well as concepts concerning relations between parts of structured objects.

The starting point for the proposed method is the remark that spatio-temporal concept identification concerning structured objects requires observing changes of these objects over a longer period of time (the so-called longer time windows) than in the case of complex objects which are parts of structured objects. Moreover, spatio-temporal concept identification concerning structured objects requires not only an observation of changes of all constituent parts of a given structured object individually, but also an observation of relations between these constituent parts and changes concerning these relations. Therefore, in order to identify spatio-temporal concepts concerning structured objects in behavioral graphs, we may observe paths of their constituent objects corresponding to constituent part behaviors in a given period. Apart from that paths in behavioral graphs describing relation changes between parts of structured objects should be observed. The properties of these paths may be defined using functions which we call *temporal patterns for temporal paths* (see Section 6.17). These patterns, being in fact formulas from a certain language, characterize spatio-temporal properties of the examined structured object in terms of its parts and constraints between these parts. On a slightly higher abstraction level, to describe behaviors of structured objects, the so-called *temporal concepts for structured objects* (see Section 6.20) are used, which are defined by an expert in a natural language and formulated usually with the help of questions about the current status of structured objects, *e.g.*, *Does one of the two observed vehicles approach the other driving behind it in the right lane?*, *Does one of the two observed vehicles change lanes from the right to the left one driving behind the second vehicle?*

The method of temporal concept approximation concerning structured objects, proposed here, is based on approximation of temporal concepts using temporal patterns for paths in behavioral graphs of parts of structured objects with the usage of temporal patterns for paths in behavioral graphs reflecting relation changes between the constituent parts. In order to do this a special decision table is constructed called *a temporal concept table of structured objects* (see Section 6.20). The rows of this table are obtained by arranging feature (attribute) value vectors of paths from behavioral graphs corresponding to parts of the structured objects observed in the data set (and, more precisely, value vectors of cluster features of such paths) and value vectors of path features from the behavioral graph

reflecting relation changes between parts of the structured object (and, more precisely, value vectors of cluster features of such paths). From the mathematical point of view such an arrangement is a Cartesian product of linked feature vectors. However, in terms of domain knowledge not all links belonging to such a Cartesian product are possible and making sense (see [78, 84, 186, 187]).

According to the general methodology presented above, to eliminate such arrangements of feature vectors that are unreal or do not make sense, we define the so-called constraints which are formulas obtained on the basis of values occurring in the vectors arranged. The constraints determine which vectors may be arranged in order to obtain an example of a concept from a higher level and which may not. Additionally, we assume that to each feature vector arrangement, acceptable by constraints, the expert adds the decision value informing about the fact whether a given arrangement belongs to the approximated concept from the higher level.

Methods of Behavioral Pattern Identification. Similarly to the case of spatio-temporal concepts for unstructured complex objects, the spatio-temporal concepts defined for structured objects may also be treated as nodes of a certain directed graph which is called *a behavioral graph for a structured object* (see Section 6.22).

These graphs may be used to represent and identify the so-called behavioral patterns which are complex concepts concerning dynamic properties of complex structured objects expressed in a natural language depending on time and space. Examples of behavioral patterns may be: *overtaking on the road*, *driving in a traffic jam*, *behavior of a patient connected with a high life threat*. These types of concepts are much more difficult to approximate even than many temporal concepts.

In the paper, a new method of behavioral pattern identification is presented which is based on interpreting the behavioral graph of a structured object as a complex classifier enabling identification of a behavioral pattern described by this graph. This is possible based on the observation of the structured object behavior for a longer time and checking whether the behavior matches the chosen behavioral graph path. If this is so, then it is determined if the behavior matches the behavioral pattern represented by this graph, which enables a detection of specific behaviors of structured objects (see Section 6.23).

The effective application of the above behavioral pattern identification method encounters, however, two problems in practice. The first of them concerns extracting relevant context for the parts of structured objects (see the fourth problem from Section 1.2). To solve this problem *a sweeping method*, enabling a rapid structured object extraction, is proposed in this paper. This method works on the basis of simple heuristics called *sweeping algorithms around complex objects* which are constructed with the use of a domain knowledge supported by data sets (see Section 6.13).

The second problem appearing with behavioral pattern identification is the problem of fast elimination of such objects that certainly do not match a given behavioral pattern (see the fifth problem from Section 1.2). As one of the methods of solving this problem, we proposed the so-called method of fast

elimination of specific behavioral patterns in relation to the analyzed structured objects. This method is based on the so-called *rules of fast elimination of behavioral patterns* which are determined from the data and on the basis of a domain knowledge (see Section 6.24). It leads to a great acceleration of behavioral pattern identification because such structured objects, whose behavior certainly does not match a given behavioral pattern, may be very quickly eliminated. For these objects it is not necessary to apply the method based on behavioral graphs which greatly accelerates the global perception.

In order to test the quality and effectiveness of classifier construction methods based on behavioral patterns, there have been performed experiments on data generated from the road simulator and medical data connected to detection of higher-death risk in infants suffering from the respiratory failure (see Section 6.25 and Section 6.26). The experiments showed that the algorithmic methods presented in this paper provide very good results in detecting behavioral patterns and may be useful with complex dynamical systems monitoring.

Methods of Automated Planning. Automated planning methods for unstructured complex objects were also worked out. These methods work on the basis of data sets and a domain knowledge represented by a concept ontology. A crucial novelty in the method proposed here, in comparison with the already existing ones, is the fact that performing actions according to plan depends on satisfying complex vague spatio-temporal conditions expressed in a natural language, which leads to the necessity of approximation of these conditions as complex concepts. Moreover, these conditions describe complex concept changes which should be reflected in the concept ontology.

Behavior of unstructured complex objects is modeled using the so-called *planning rules* being formulas of the type: *the state before performing an action* $\rightarrow$ *action* $\rightarrow$ *state* 1 *after performing an action* | ... | *state k after performing an action*, which are defined on the basis of data sets and a domain knowledge (see Section 7.4). Each rule includes the description of the complex object state before applying the rule (that is, before performing an action), expressed in a language of features proposed by an expert, the name of the action (one of the actions specified by the expert which may be performed at a particular state), and the description of sequences of states which a complex object may turn into after applying the action mentioned above. It means that the application of such a rule gives indeterministic effects, i.e., after performing the same action the system may turn into different states. All planning rules may be represented in a form of the so-called *planning graphs* whose nodes are state descriptions (occurring in predecessors and successors of planning rules) and action names occurring in planning rules (see Section 7.4). In the graphical interpretation, solving the problem of automated planning is based on finding a path in the planning graph from the initial state to an expected final state. It is worth noticing that the conditions for performing an action (object states) are described by vague spatio-temporal complex concepts which are expressed in the natural language and require an approximation.

For specific applications connected with the situation when it is expected that the proposed plan of a complex object behavior is to be strictly compatible with

the determined experts' instructions (*e.g.*, the way of treatment in a specialist clinic is to be compatible with the treatment schemes used there), there has also been proposed an additional mechanism enabling to resolve the nondeterminism occurring in the application of planning rules. This mechanism is an additional classifier based on data sets and domain knowledge. Such classifiers suggest the action to be performed in a given state and show the state which is the result of the indicated action (see Section 7.7).

The automated planning method for unstructured objects has been generalized in the paper also in the case of planning of the behavior of structured objects (consisting of parts connected with one another by dependencies). The generalization is based on the fact that on the level of a structured object there is an additional planning graph defined where there are double-type nodes and directed edges between the nodes (see Section 7.11). The nodes of the first type describe vague features of states (meta-states) of the whole structured object, whereas the nodes of the second type concern complex actions (meta-actions) performed by the whole structured object (all its constituent parts) over a longer period of time (a time window). The edges between the nodes represent temporal dependencies between meta-states and meta-actions as well as meta-actions and meta-states. Similarly to the previous case of unstructured objects, planning of a structured object behavior is based on finding a path in a planning graph from the initial meta-state to the expected final meta-state; and, at the same time, each meta-action occurring in such a path must be planned separately on the level of each constituent part of the structured object. In other words, it should be planned what actions each part of a structured object must perform in order for the whole structured object to be able to perform the meta-action which has been planned. During the planning of a meta-action a synchronization mechanism (determining compatibility) of plans proposed for the part of a structured object is used, which works on the basis of a family of classifiers determined on the basis of data sets with a great support of domain knowledge. Apart from that, an additional classifier is applied (also based on a data set and the domain knowledge) which enables to determine whether the juxtaposition and execution of plans determined for the constituent parts, in fact, lead to the execution of the meta-action planned on the level of the whole structured object (see Section 7.13).

During the attempt to execute the plan constructed there often appears a need to reconstruct the plan which means that during the plan execution there may appear such a state of a complex object that is not compatible with the state suggested by the plan. A *total reconstruction* of the plan (building the whole plan from the beginning) may computationally be too costly. Therefore, we propose another plan reconstruction method called *a partial reconstruction*. It is based on constructing a short so-called *repair plan*, which rapidly brings the complex object to the so-called *return state* which appears in the current plan. Next, on the basis of the repair plan, a current plan reconstruction is performed through replacing its fragment beginning with the current state and ending with the return plan with the repair plan (see Section 7.9 and Section 7.17).

In construction and application of classifiers approximating complex spatio-temporal concepts, there may appear a need to construct, with a great support of the domain knowledge, a similarity relation of two elements of similar type, such as complex objects, complex object states, or plans generated for complex objects. Hence, in this paper we propose a new method of similarity relation approximation based on the use of data sets and a domain knowledge expressed mainly in the form of a concept ontology. We apply this method, among other things, to verify automated planning methods, that is, to compare the plan generated automatically with the plan suggested by experts from a given domain (see Section 7.18, Section 7.19 and Section 7.20).

In order to check the effectiveness of the automated planning methods proposed here, there were performed experiments concerning planning of treatment of infants suffering from the respiratory failure (see Section 7.21). Experimental results showed that the proposed method gives good results, also in the opinion of medical experts (compatible enough with the plans suggested by the experts), and may be applied in medical practice as a supporting tool for planning of the treatment of infants suffering from the respiratory failure.

Implementation and Data Sets. The result of the works conducted is also a programming system supporting the approximation of spatio-temporal complex concepts in the given concept ontology in the dialogue with the user. The system also includes an implementation of the algorithmic methods presented in this paper and is available on the web side of RSES system (see [15]).

Sections 5, 6 and 7, apart from the method description, contain the results of computing experiments conducted on real-life data sets, supported by domain knowledge. It is worth mentioning that the requirements regarding data sets which can be used for computing experiments with modeling spatio-temporal phenomena are much greater than the requirements of the data which are used for testing process of classical classifiers. Not only have the data to be representative of the decision making problem under consideration but also they have to be related to the domain knowledge available (usually cooperation with experts in a particular domain is essential). It is important that such data should fully and appropriately reflect complex spatio-temporal phenomena connected to the environment of the data collected.

The author of the paper acquired such data sets from two sources. The first source of data is the traffic simulator made by the author (see Appendix A). The simulator is a computing tool for generating data sets connected to the traffic on the street and at crossroads. During simulation each vehicle appearing on the simulation board behaves as an independently acting agent. On the basis of observation of the surroundings (other vehicles, its own location, weather conditions, etc.) this agent makes an independent decision what maneuvers it should make to achieve its aim which is to go safely across the simulation board and to leave the board using the outbound way given in advance. At any given moment of the simulation, all crucial vehicle parameters may be recorded, and thanks to this data sets for experiments can be obtained.

The second collection of data sets used in computer experiments was provided by Neonatal Intensive Care Unit, First Department of Pediatrics, Polish-American Institute of Pediatrics, Collegium Medicum, Jagiellonian University, Krakow, Poland. This data constitutes a detailed description of treatment of 300 infants, i.e., treatment results, diagnosis, operations, medication (see Section 6.26 and Appendix B).

1.4 Organization of the Paper

This paper is organized as follows. In Section 2 we briefly describe selected classical methods of classifier construction and concept approximation which are used in next subsections of the paper. These methods are based on rough set theory achievements and were described in the author's previous papers (see, *e.g.*, [14, 195, 196, 197, 198, 199, 200, 201, 202, 203]).

In Section 3 we describe methods of construction of a concept stratifying classifier.

The general methodology of approximating concepts with the use of data sets and a domain knowledge represented mainly in the form of a concept ontology is described in Section 4.

Methods of approximating spatial concepts from ontology are described in Section 5, whereas methods of approximating spatio-temporal concepts from ontology and methods of behavioral patterns identification are described in Section 6.

Methods of automated planning of complex object behavior when object states are represented with the help of complex objects requiring an approximation with the use of data sets and a domain knowledge are presented in Section 7.

Finally, in Section 8 we summarize the results and give directions for the future research.

The paper also contains two appendixes. The first appendix contains the description of the traffic simulator used to generate experimental data (see Appendix A). The second one describes medical issues connected with the infant respiratory failure (see Appendix B) concerning one of the data sets used for experiments.

2 Classical Classifiers

In general, the term *classify* means *arrange objects in a group or class based on shared characteristics* (see [1]). In this work, the term *classification* has a special meaning, *i.e.*, *classification* connotes any context in which some decision or forecast about object grouping is made on the basis of currently available knowledge or information (see, *e.g.*, [11, 204]).

A *classification algorithm* (*classifier*) is an algorithm which enables us to make a forecast repeatedly on the basis of accumulated knowledge in new situations (see, *e.g.*, [11]). Here we consider the classification provided by a classifying

algorithm which is applied to a number of cases to classify objects unseen previously. Each new object is assigned to a class belonging to a predefined set of classes on the basis of observed values of suitably chosen attributes (features).

Many approaches have been proposed to construct classification algorithms. Among them we would like to mention classical and modern statistical techniques (see, *e.g.*, [11, 13]), neural networks (see, *e.g.*, [11, 13, 205]), decision trees (see, *e.g.*, [11, 206, 207, 208, 209, 210, 211, 212]), decision rules (see, *e.g.*, [10, 11, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223]) and inductive logic programming (see, *e.g.*, [11, 224]).

In this section, we consider methods implemented in our system RSES (*Rough Set Exploration System*) (see [14, 225, 226, 227, 228, 229, 230, 231]). RSES is a computer software system developed for the purpose of data analysis (the data is assumed to be in the form of an information system or a decision table, see Section 2.1). In construction of classifiers, which is the main step in the process od data analysis with RSES, elements of rough set theory are used. In this paper, we call these algorithms the *standard RSES methods* of classifier construction.

The majority of the standard RSES methods of classifier construction have been applied in more advanced methods of classifier construction, which will be presented in Sections 3, 5, 6, and 7. Therefore, in this section we only give a brief overview of that methods of classifier construction. These methods are based on *rough set theory* (see [16, 17, 232]). In the Section 2.1 we start with introduction of basic rough set terminology and notation, necessary for the rest of this paper (see Section 2.1).

The analysis of data in the RSES system proceeds according to the scheme presented in Fig. 2. First, the data for analysis has to be loaded/imported into the system. Next, in order to have a better chance for constructing (learning) a proper classifier, it is frequently advisable to transform the initial data set. Such transformation, usually referred to as *preprocessing*, may consist of several steps. RSES supports preprocessing methods which make it possible to manage

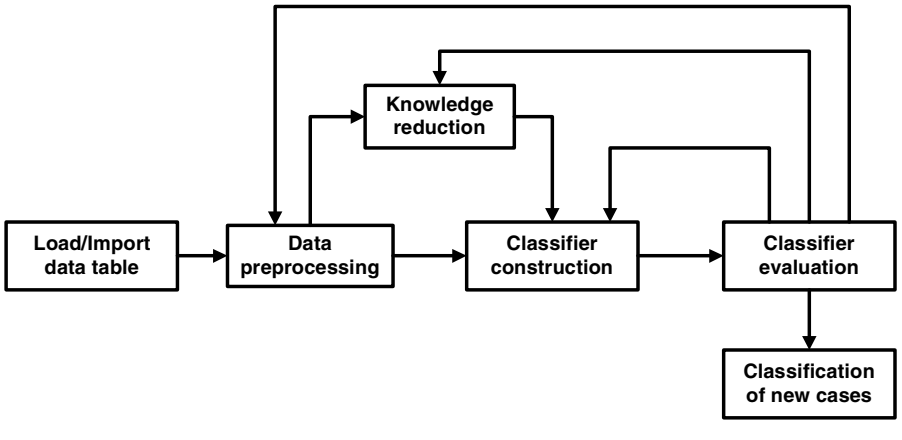


Fig. 2. The RSES data analysis process

missing parts in data, discretize numeric attributes, and create new attributes (see [14] and Section 2.2 for more details).

When the data is preprocessed, we can be interested in learning about its internal structure. By using classical rough set concepts such as reducts (see Section 2.1), dynamic reducts (see [14, 195, 196, 198, 201, 202, 203]), and positive region (see Section 2.1) one can discover dependencies that occur in our data set. Knowledge of reducts can lead to reduction of data by removing some of the redundant attributes.

Next, the classifier construction may be started. In the RSES system, these classifiers may be constructed using various methods (see [14] and sections 2.3, 2.4, 2.5, 2.6, 2.7 for more details).

A classifier is constructed on the basis of a training set consisting of labeled examples (objects with decisions). Such a classifier may further be used for evaluation on a test set or applied to new, unseen and unlabeled cases in order to determine the value of decision (classification) for them (see Section 2.9).

If the quality of the constructed classifier is insufficient, one may return to data preprocessing and/or knowledge reduction; another method of classifier construction may be applied as well.

2.1 Rough Set Basic Notions

In order to provide a clear description further in the paper and to avoid any misunderstandings, we bring here some essential definitions from rough set theory. We will frequently refer to the notions introduced in this section. Quite a comprehensive description of notions and concepts related to the classical rough set theory may be found in [189].

An *information system* (see [16, 17]) is a pair $\mathbf{A} = (U, A)$ where U is a non-empty, finite set called the *universe* of $\mathbf{A}$ and A is a non-empty, finite set of *attributes*, i.e., mappings $a : U \to V_a$, where V_a is called the *value set* of $a \in A$.

Elements of U are called *objects* and interpreted as, *e.g.*, cases, states, processes, patients, observations. Attributes are interpreted as features, variables, characteristic conditions.

We also consider a special case of information systems called decision tables. A *decision table* is an information system of the form $\mathbf{A} = (U, A, d)$ where $d \notin A$ is a distinguished attribute called the *decision*. The elements of A are called *condition attributes* or *conditions*.

One can interpret the decision attribute as a kind of partition of the universe of objects given by an expert, a decision-maker, an operator, a physician, etc. In machine learning decision tables are called training sets of examples (see [10]).

The cardinality of the image $d(U) = \{k : d(s) = k \text{ for some } s \in U\}$ is called the *rank of* d and is denoted by $r(d)$.

We assume that the set V_d of values of the decision d is equal to $\{v_d^1, ..., v_d^{r(d)}\}$.

Let us observe that the decision d determines a partition $CLASS_{\mathbf{A}}(d) = \{X_{\mathbf{A}}^1, \ldots, X_{\mathbf{A}}^{r(d)}\}$ of the universe U where $X_{\mathbf{A}}^k = \{x \in U : d(x) = v_d^k\}$ for $1 \leq k \leq r(d)$. $CLASS_{\mathbf{A}}(d)$ is called the *classification of objects of* $\mathbf{A}$ *determined*

by the decision d. The set $X^i_{\mathbf{A}}$ is called the *i-th decision class of* $\mathbf{A}$. By $X_{\mathbf{A}}(u)$ we denote the decision class $\{x \in U : d(x) = d(u)\}$, for any $u \in U$.

Let $\mathbf{A} = (U, A)$ be an information system. For every set of attributes $B \subseteq A$, an equivalence relation, denoted by $IND_{\mathbf{A}}(B)$ and called the *B-indiscernibility relation*, is defined by

$$IND_{\mathbf{A}}(B) = \{(u, u') \in U \times U : \ \forall_{a \in B}\ a(u) = a(u')\}. \tag{1}$$

Objects u, u' being in the relation $IND_{\mathbf{A}}(B)$ are indiscernible by attributes from B.

By $[u]_{IND_{\mathbf{A}}(B)}$ we denote the equivalence class of the relation $IND_{\mathbf{A}}(B)$, such that u belongs to this class.

An attribute $a \in B \subseteq A$ is *dispensable* in B if $IND_{\mathbf{A}}(B) = IND_{\mathbf{A}}(B \setminus \{a\})$, otherwise a is *indispensable* in B. A set $B \subseteq A$ is *independent* in $\mathbf{A}$ if every attribute from B is indispensable in B, otherwise the set B is *dependent* in $\mathbf{A}$. A set $B \subseteq A$ is called *a reduct* in $\mathbf{A}$ if B is independent in $\mathbf{A}$ and $IND_{\mathbf{A}}(B) = IND_{\mathbf{A}}(A)$. The set of all reducts in $\mathbf{A}$ is denoted by $RED_{\mathbf{A}}(A)$. This is the classical notion of a reduct and it is sometimes referred to as *global reduct*.

Let $\mathbf{A} = (U, A)$ be an information system with n objects. By $M(\mathbf{A})$ (see [21]) we denote an $n \times n$ matrix (c_{ij}), called the *discernibility matrix* of $\mathbf{A}$, such that

$$c_{ij} = \{a \in A : a(x_i) \neq a(x_j)\} \quad \text{for} \quad i, j = 1, \ldots, n\,. \tag{2}$$

A *discernibility function* $f_{\mathbf{A}}$ for an information system $\mathbf{A}$ is a Boolean function of m Boolean variables $\overline{a}_1, \ldots, \overline{a}_m$ corresponding to the attributes $a_1, \ldots, a_m$, respectively, and defined by

$$f_{\mathbf{A}}(\overline{a}_1, \ldots, \overline{a}_m) = \bigwedge \{\bigvee \overline{c}_{ij} : 1 \leq j < i \leq n \ \wedge \ c_{ij} \neq \emptyset\}, \tag{3}$$

where $\overline{c}_{ij} = \{\overline{a} : a \in c_{ij}\}$.

It can be shown (see [21]) that the set of all *prime implicants* of $f_{\mathbf{A}}$ determines the set of all *reducts* of A.

We present an exemplary deterministic algorithms for computation of the whole reduct set $RED_{\mathbf{A}}(A)$ (see, *e.g.*, [199]). This algorithm computes the discernibility matrix of $\mathbf{A}$ (see Algorithm 2.1).

The time cost of the reduct set computation using the algorithm presented above can be too high in the case the decision table consists of too many objects, attributes, or different values of attributes. The reason is that, in general, the size of the reduct set can be exponential with respect to the size of the decision table and the problem of the minimal reduct computation is NP-hard (see [21]). Therefore, we are often forced to apply approximation algorithms to obtain some knowledge about the reduct set. One way is to use approximation algorithms that need not give optimal solutions but require a short computing time. Among these algorithms are the following ones: Johnson's algorithm, covering algorithms, algorithms based on simulated annealing and Boltzmann machines, algorithms using neural networks and algorithms based on genetic algorithms (see, *e.g.*, [196, 198, 199] for more details).

Algorithm 2.1. Reduct set computation

Input: Information system $\mathbf{A} = (U, A)$
Output: Set $RED_{\mathbf{A}}(A)$ of all reducts of $\mathbf{A}$

```
1  begin
2      Compute indiscernibility matrix M(A)
3      Reduce M(A) using absorbtion laws
       // Let C_1, ..., C_d are non-empty fields of reduced M(A)
4      Build a familie of sets R_0, R_1, ..., R_d in the following way:
5      begin
6          R_0 = ∅
7          for i = 1 to d do
8              R_i = S_i ∪ T_i where S_i = {R ∈ R_{i-1} : R ∩ C_i ≠ ∅}
9                and T_i = (R ∪ {a})_{a∈C_i, R∈R_{i-1}: R∩C_i=∅}
10         end
11     end
12     Remove dispensable attributes from each element of family R_d
13     Remove redundant elements from R_d
14     RED_A(A) = R_d
15 end
```

If $\mathbf{A} = (U, A)$ is an information system, $B \subseteq A$ is a set of attributes and $X \subseteq U$ is a set of objects (usually called a *concept*), then the sets $\{u \in U : [u]_{IND_{\mathbf{A}}(B)} \subseteq X\}$ and $\{u \in U : [u]_{IND_{\mathbf{A}}(B)} \cap X \neq \emptyset\}$ are called the *B-lower* and the *B-upper approximations* of X in $\mathbf{A}$, and they are denoted by $\underline{B}X$ and $\overline{B}X$, respectively.

The set $BN_B(X) = \overline{B}X - \underline{B}X$ is called the *B-boundary* of X (*boundary region*, for short). When $B = A$, we also write $BN_{\mathbf{A}}(X)$ instead of $BN_A(X)$.

Sets which are unions of some classes of the indiscernibility relation $IND_{\mathbf{A}}(B)$ are called *definable* by B (or *B-definable* in short). A set X is, thus, B-definable iff $\overline{B}X = \underline{B}X$. Some subsets (categories) of objects in an information system cannot be exactly expressed in terms of the available attributes but they can be defined roughly.

The set $\underline{B}X$ is the set of all elements of U which can be classified with certainty as elements of X, given a knowledge about these elements in the form of values of attributes from B; the set $BN_B(X)$ is the set of elements of U which one can classify neither to X nor to $-X$ having a knowledge about objects represented by B.

If the boundary region of $X \subseteq U$ is the empty set, i.e., $BN_B(X) = \emptyset$, then the set X is called *crisp* (*exact*) with respect to B; in the opposite case, i.e., if $BN_B(X) \neq \emptyset$, the set X is referred to as *rough* (*inexact*) with respect to B (see, *e.g.*, [17]).

If $X_1, \ldots, X_{r(d)}$ are decision classes of $\mathbf{A}$, then the set $\underline{B}X_1 \cup \cdots \cup \underline{B}X_{r(d)}$ is called *the B-positive region of* $\mathbf{A}$ and denoted by $POS_B(d)$.

If $\mathbf{A} = (U, A, d)$ is a decision table and $B \subseteq A$, then we define a function $\partial_B : U \to \mathbf{P}(V_d)$, called *the B-generalized decision of* $\mathbf{A}$, by

$$\partial_B(x) = \{v \in V_d : \exists x' \in U\,(x' IND_{\mathbf{A}}(B) x \text{ and } d(x) = v)\}\,. \tag{4}$$

The A-generalized decision ∂_A of $\mathbf{A}$ is called the *generalized decision* of $\mathbf{A}$.

A decision table $\mathbf{A}$ is called *consistent (deterministic)* if $card(\partial_A(x)) = 1$ for any $x \in U$, otherwise $\mathbf{A}$ is *inconsistent (non-deterministic)*. Non-deterministic information systems were introduced by Witold Lipski (see [233]), while deterministic information systems independently by Zdzisław Pawlak [234] (see, also, [235, 236]). It is easy to see that a decision table $\mathbf{A}$ is consistent iff $POS_A(d) = U$. Moreover, if $\partial_B = \partial_{B'}$, then $POS_B(d) = POS_{B'}(d)$ for any pair of non-empty sets $B, B' \subseteq A$.

A subset B of the set A of attributes of a decision table $\mathbf{A} = (U, A, d)$ is *a relative reduct of* $\mathbf{A}$ iff B is a minimal set with respect to the following property: $\partial_B = \partial_A$. The set of all relative reducts of $\mathbf{A}$ is denoted by $RED(\mathbf{A}, d)$.

Let $\mathbf{A} = (U, A, d)$ be a consistent decision table and let $M(\mathbf{A}) = (c_{ij})$ be its discernibility matrix. We construct a new matrix $M'(\mathbf{A}) = (c'_{ij})$ assuming $c'_{ij} = \emptyset$ if $d(x_i) = d(x_j)$, and $c'_{ij} = c_{ij} - \{d\}$ otherwise. The matrix $M'(\mathbf{A})$ is called *the relative discernibility matrix of* $\mathbf{A}$. Now, one can construct the *relative discernibility function* $f_{M'(A)}$ of $M'(\mathbf{A})$ in the same way as the discernibility function.

It can be shown (see [21]) that the set of all *prime implicants* of $f_{M'(A)}$ determines the set of all *relative reducts* of $\mathbf{A}$.

Another important type of reducts are local reducts. *A local reduct* $r(x_i) \subseteq A$ (or a *reduct relative to decision and object* $x_i \in U$ where x_i is called a *base object*) is a subset of A such that:

1. $\forall_{x_j \in U}\ d(x_i) \neq d(x_j) \Longrightarrow \exists_{a_k \in r(x_i)}\ a_k(x_i) \neq a_k(x_j)$,
2. $r(x_i)$ is minimal with respect to inclusion.

If $\mathbf{A} = (U, A, d)$ is a decision table, then any system $\mathbf{B} = (U', A, d)$ such that $U' \subseteq U$ is called a *subtable* of $\mathbf{A}$.

A template of $\mathbf{A}$ is a formula $\bigwedge(a_i = v_i)$ where $a_i \in A$ and $v_i \in V_{a_i}$. A generalized template is a formula of the form $\bigwedge(a_i \in T_i)$ where $T_i \subset V_{a_i}$. An object *satisfies* (matches) a template if for every attribute a_i occurring in the template, the value of this attribute at a considered object is equal to v_i (belongs to T_i in the case of the generalized template). The template splits the original information system in the two distinct subtables containing objects that satisfy and do not satisfy the template, respectively.

It is worth mentioning that the notion of a template can be treated as a particular case of a more general notion, viz., that of *a pattern* (see Section 4.9).

2.2 Discretization

Suppose we have a decision table $\mathbf{A} = (U, A, d)$ where $card(V_a)$ is high for some $a \in A$. Then, there is a very low chance that a new object is recognized by rules

generated directly from this table because the attribute value vector of a new object will not match any of these rules. Therefore, for decision tables with real (numerical) value attributes, some discretization strategies are built in order to obtain a higher quality of classification. This problem was intensively studied (see, *e.g.*, [199, 237, 238] for more details).

The process of discretization is usually realized in two following steps (see, *e.g.*, [14, 199, 237, 238]). First, the algorithm generates a set of cuts. By a *cut* for an attribute $a_i \in A$ such that V_{a_i} is an ordered set we denote a value $c \in V_{a_i}$. The cuts can be then used to transform the decision table. As a result we obtain a decision table with the same set of attributes but the attributes have different values. Instead of $a(x) = v$ for an attribute $a \in A$ and an object $x \in U$, we rather get $a(x) \in [c_1, c_2]$ where c_1 and c_2 are cuts generated for attribute a by a discretization algorithm. The cuts are generated in a way that the resulting intervals contain possibly most uniform sets of objects w.r.t decision.

The discretization method available in RSES has two versions (see, *e.g.*, [14, 199, 238]) that are usually called *global* and *local*. Both methods belong to a bottom-up approaches which add cuts for a given attribute one-by-one in subsequent iterations of algorithm. The difference between these two methods lies in the way in which the candidate for a new cut is evaluated. In the global method, we evaluate all objects in the data table at every step. In the local method, we only consider a part of objects that are related to the candidate cut, i.e., which have the value of the attribute considered currently in the same range as the cut candidate. Naturally, the second (local) method is faster as less objects have to be examined at every step. In general, the local method produces more cuts. The local method is also capable of dealing with nominal (symbolic) attributes. Grouping (quantization) of a nominal attribute domain with use of the local method always results in two subsets of attribute values (see, *e.g.*, [14, 199, 238] for more details).

2.3 Decision Rules

Let $\mathbf{A} = (U, A, d)$ be a decision table and let $V = \bigcup\{V_a : a \in A\} \cup V_d$. Atomic formulas over $B \subseteq A \cup \{d\}$ and V are expressions of the form $a = v$, called *descriptors* over B and V, where $a \in B$ and $v \in V_a$. The set $\mathbf{F}(B, V)$ of formulas over B and V is the least set containing all atomic formulas over B, V and closed with respect to the classical propositional connectives $\vee$ (disjunction), $\wedge$ (conjunction), and $\neg$ (negation).

Let $\varphi \in \mathbf{F}(B, V)$. Then, by $|\varphi|_{\mathbf{A}}$ we denote the meaning of φ in the decision table $\mathbf{A}$, i.e., the set of all objects of U with the property φ, defined inductively by

1. if φ is of the form $a = v$, then $|\varphi|_{\mathbf{A}} = \{x \in U : a(x) = v\}$,
2. $|\varphi \wedge \varphi'|_{\mathbf{A}} = |\varphi|_{\mathbf{A}} \cap |\varphi'|_{\mathbf{A}}$,
3. $|\varphi \vee \varphi'|_{\mathbf{A}} = |\varphi|_{\mathbf{A}} \cup |\varphi'|_{\mathbf{A}}$,
4. $|\neg\varphi|_{\mathbf{A}} = U - |\varphi\}_{\mathbf{A}}$.

The set $\mathbf{F}(A, V)$ is called the set of *conditional formulas of* $\mathbf{A}$ and is denoted by $\mathbf{C}(A, V)$.

Any formula of the form $(a_1 = v_1) \wedge ... \wedge (a_l = v_l)$ where $v_i \in V_{a_i}$ (for $i = 1, ..., l$) and $P = \{a_1, ..., a_l\} \subseteq A$ is called a *P-basic* formula of $\mathbf{A}$.

If φ is a *P-basic* formula of $\mathbf{A}$ and $Q \subseteq P$, then by φ/Q we mean the Q-basic formula obtained from the formula φ by removing from φ all its elementary subformulas $(a = v_a)$ such that $a \in P \setminus Q$.

A *decision rule* for $\mathbf{A}$ is any expression of the form $\varphi \Rightarrow d = v$ where $\varphi \in \mathbf{C}(A, V)$, $v \in V_d$, and $|\varphi|_{\mathbf{A}} \neq \emptyset$. Formulas φ and $d = v$ are referred to as the *predecessor* (*premise* of the rule) and the *successor* of the decision rule $\varphi \Rightarrow d = v$ respectively.

If r is a decision rule in $\mathbf{A}$, then by $Pred(r)$ we denote the predecessor of r and by $Succ(r)$ we denote the successor of r .

An object $u \in U$ is *matched* by a decision rule $\varphi \Rightarrow d = v_d^k$ (where $1 \leq k \leq r(d)$) iff $u \in |\varphi|_{\mathbf{A}}$. If u is matched by $\varphi \Rightarrow d = v_d^k$, then we say that the rule is classifying u to the decision class X_k.

The number of objects matched by a decision rule $\varphi \Rightarrow d = v$, denoted by $Match_{\mathbf{A}}(\varphi \Rightarrow d = v)$, is equal to $card(|\varphi|_{\mathbf{A}})$.

The number $Supp_{\mathbf{A}}(\varphi \Rightarrow d = v) = card(|\varphi|_{\mathbf{A}} \cap |d = v|_{\mathbf{A}})$ is called *the number of objects supporting the decision rule* $\varphi \Rightarrow d = v$.

A decision rule $\varphi \Rightarrow d = v$ for $\mathbf{A}$ is *true* in $\mathbf{A}$, symbolically $\varphi \Rightarrow_{\mathbf{A}} d = v$, iff $|\varphi|_{\mathbf{A}} \subseteq |d = v|_{\mathbf{A}}$. If the decision rule $\varphi \Rightarrow d = v$ is true in $\mathbf{A}$, we say that the decision rule is *consistent* in $\mathbf{A}$, otherwise $\varphi \Rightarrow d = v$ is *inconsistent* or *approximate* in $\mathbf{A}$.

If r is a decision rule in $\mathbf{A}$, then the number $\mu_{\mathbf{A}}(r) = \frac{Supp_{\mathbf{A}}(r)}{Match_{\mathbf{A}}(r)}$ is called *the coefficient of consistency* of the rule r. The coefficient $\mu_{\mathbf{A}}(r)$ may be understood as the degree of consistency of the decision rule r. It is easy to see that a decision rule r for $\mathbf{A}$ is consistent iff $\mu_{\mathbf{A}}(r) = 1$.

The coefficient of consistency of r can be also treated as the degree of inclusion of $|Pred(r)|_{\mathbf{A}}$ in $|Succ(r)|_{\mathbf{A}}$ (see, *e.g.*, [239]).

If $\varphi \Rightarrow d = v$ is a decision rule for $\mathbf{A}$ and φ is P-basic formula of $\mathbf{A}$ (where $P \subseteq A$), then the decision rule $\varphi \Rightarrow d = v$ is called a *P-basic decision rule for* $\mathbf{A}$, or a *basic decision rule* in short.

Let $\varphi \Rightarrow d = v$ be a P-basic decision rule of $\mathbf{A}$ (where $P \subseteq A$) and let $a \in P$. We will say that the attribute a is *dispensable* in the rule $\varphi \Rightarrow d = v$ iff $|\varphi \Rightarrow d = v|_{\mathbf{A}} = U$ implies $|\varphi/(P \setminus \{a\}) \Rightarrow d = v|_{\mathbf{A}} = U$, otherwise attribute a is *indispensable* in the rule $\varphi \Rightarrow d = v$. If all attributes $a \in P$ are indispensable in the rule $\varphi \Rightarrow d = v$, then $\varphi \Rightarrow d = v$ will be called *independent* in $\mathbf{A}$.

The subset of attributes $R \subseteq P$ will be called a *reduct* of P-basic decision rule $\varphi \Rightarrow d = v$, if $\varphi/R \Rightarrow d = v$ is independent in $\mathbf{A}$ and $|\varphi \Rightarrow d = v|_{\mathbf{A}} = U$ implies $|\varphi/R \Rightarrow d = v|_{\mathbf{A}} = U$. If R is a reduct of the P-basic decision rule $\varphi \Rightarrow d = v$, then $\varphi/R \Rightarrow d = v$ is said to be *reduced*. If R is a reduct of the A-basic decision rule $\varphi \Rightarrow d = v$, then $\varphi/R \Rightarrow d = v$ is said to be *an optimal basic decision rule of* $\mathbf{A}$ (*a basic decision rule with minimal number of descriptors*). The set of all optimal basic decision rules of $\mathbf{A}$ is denoted by $RUL(\mathbf{A})$.

2.4 Two Methods for Decision Rule Synthesis

Classifiers based on a set of decision rules are the most elaborated methods in RSES. Several methods for calculation of the decision rule sets are implemented. Also, various methods for transforming and utilizing rule sets are available. However, in our computer experiments we usually use two methods for decision rules synthesis. We would like to mention those methods here.

The first method returns all basic decision rules with minimal number of descriptors (see, *e.g.*, [196, 198, 199, 240]). Therefore, this method is often called *an exhaustive method*. From the practical point of view, the method consists in applying an algorithm computing all reducts (see Algorithm 2.1) for each object individually, which results in obtaining decision rules with a minimal number of descriptors in relation to individual objects (see, *e.g.*, [196, 198, 199]).

The second method for basic decision rule synthesis, is the covering algorithm called LEM2 (see, *e.g.*, [216, 222, 223]). In LEM2, a separate-and-conquer technique is paired with rough set notions such as upper and lower approximations. This method tends to produce less rules than algorithms based on the exhaustive local reduct calculation (as in the previous method) and seems to be faster. On the downside, the LEM2 method sometimes returns too few valuable and meaningful rules (see also Section 2.10).

2.5 Operations on Rule Sets

In general, the methods used by RSES to generate rules may produce quite a bunch of them. Naturally, some of the rules may be marginal, erroneous or redundant. In order to provide a better control over the rule-based classifiers some simple techniques for transforming rule sets should be used. The simplest way to alter a set of decision rules is by filtering them. It is possible to eliminate from the rule set these rules that have insufficient support on training sample, or those that point at a decision class other than the desired one. More advanced operations on rule sets are *shortening* and *generalization*.

Rule shortening is a method that attempts to eliminate descriptors from the premise of the rule. The resulting rule is shorter, more general (applicable to more training objects) but it may lose some of its precision. The shortened rule may be less precise, i.e., it may give wrong answers (decisions) for some of the matching training objects.

We present an exemplary method of approximate rules computation (see, *e.g.*, [196, 198, 199]) that we use in our experiments. We begin with an algorithm for synthesis of optimal decision rules from a given decision table (see Section 2.4). Next, we compute approximate rules from the optimal decision rules already calculated. Our method is based on the notion of consistency of a decision rule (see Section 2.1). The original optimal rule is reduced to an approximate rule with the coefficient of consistency exceeding a fixed threshold.

Let $\mathbf{A} = (U, A, d)$ be a decision table and $r_0 \in RUL(\mathbf{A})$. The approximate rule (based on rule r_0) is computed using the Algorithm 2.2.

Algorithm 2.2. Approximate rule synthesis (by descriptor dropping)

Input:
1. decision table $\mathbf{A} = (U, A, d)$
2. decision rule $r_0 \in RUL(\mathbf{A})$
3. threshold of consistency μ_0 (*e.g.*, $\mu_0 = 0.9$)

Output: the approximate rule r_{app} (based on rule r_0)

begin
  Calculate the coefficient of consistency $\mu_{\mathbf{A}}(r_0)$
  if $\mu_{\mathbf{A}}(r_0) < \mu_0$ **then**
    STOP `// In this case no approximate rule`
  end
  $\mu_{max} = \mu_{\mathbf{A}}(r_0)$ and $r_{app} = r_0$
  while $\mu_{max} > \mu_0$ **do**
    $\mu_{max} = 0$
    for $i = 1$ **to** *the number of descriptors from* $Pred(r_{app})$ **do**
      $r = r_{app}$
      Remove i-th descriptor from $Pred(r)$
      Calculate the coefficient of consistency $\mu_{\mathbf{A}}(r)$ *and* $\mu = \mu_{\mathbf{A}}(r)$
      if $\mu > \mu_{max}$ **then**
        $\mu_{max} = \mu$ *and* $i_{max} = i$
      end
    end
    if $\mu_{max} > \mu_0$ **then**
      Remove i_{max} *-th conditional descriptor from* r_{app}
    end
  end
  return r_{app}
end

It is easy to see that the time and space complexity of Algorithm 2.2 are of order $O(l^2 \cdot m \cdot n)$ and $O(C)$, respectively (where l is the number of conditional descriptors in the original optimal decision rule r_0 and C is a constant).

The approximate rules, generated by the above method, can help to extract interesting laws from the decision table. By applying approximate rules instead of optimal rules one can slightly decrease the quality of classification of objects from the training set but we expect, in return, to receive more general rules with a higher quality of classification of new objects (see [196]).

On the other hand, generalization of rules is a process which consists in replacement of the descriptors having a single attribute value in rule predecessors with more general descriptors. In the RSES system there is an algorithm available which instead of simple descriptors of type $a(x) = v$, where $a \in A$, $v \in V_a$ and $x \in U$ tries to use the so-called generalized descriptors of the form $a(x) \in V$ where $V \subset V_a$ (see, *e.g.*, [14]). In addition, such a replacement is performed

only when the coefficient of consistency of the new rule is not smaller than the established threshold. Let us notice that such an operation is crucial in terms of enlargement of the extension of decision rules for the generalized decision rules are able to classify a greater number of tested objects.

It is worth mentioning that the application of the method of generalizing rules described above only makes sense for tables with attributes having a small number values. Such attributes are usually attributes with symbolic values. On the other hand a usage of this method for tables with numerical attributes requires a previous discretization of values of these attributes.

2.6 Negotiations Among Rules

Suppose we have a set of decision rules. When we attempt to classify an object from test sample with use of a rule set generated, it may happen that various rules suggest different decision values. In such conflict situations, we need a strategy to resolve controversy and reach a final result (decision). This problem was intensively studied (see, *e.g.*, [198, 199]). In its current version, RSES provides a conflict resolution strategy based on voting among rules. In this method, each rule that matches the object under consideration casts a vote in favor of the decision value it points at. Votes are summed up and the decision is chosen that has got majority of votes. This simple method may be extended by assigning weights to rules. Each rule, then votes with its weight and the decision that has the highest total of weighted votes is the final one. In RSES, this method is known as a *standard voting* and is based on a *basic strength* (weight) of decision rules (see Section 2.8). Of course, there are many other methods that can be used to resolve conflicts between decision rules (see, *e.g.*, [196, 198, 199, 216, 217, 241]).

2.7 Decomposition Trees

In the case of the decision tables larger, the computation of decision rules for these tables can be extremely difficult or even impossible.

This problem arises from a relatively high computational complexity of rule computing algorithms. Unfortunately, it frequently concerns covering algorithms such as, *e.g.*, LEM2 as well (see Section 2.4). One of the solutions to this problem is the so-called *decomposition*. Decomposition consists in partitioning the entrance data table into parts (subtables) in such a way as to be able to calculate decision rules for these parts using standard methods. Naturally, a method is also necessary which would aggregate the obtained rule sets in order to build a general classifier.

In this paper, we present a decomposition method based on a decomposition tree (see [165, 226, 242]) which may be constructed according to Algorithm 2.3.

This algorithm creates the decomposition tree in steps where each step leads to construction of the next level of the tree. At a given step of the algorithm execution, a binary partition of the decision table takes place using the best template (see Section 2.1) found for the table being partitioned. In this way, with each tree node (leaf), there is connected a template partitioning the subtable in this node into objects matching and not matching the template. This

Algorithm 2.3. Decomposition tree synthesis

Input: decision table $\mathbf{A} = (U, A, d)$
Output: the decomposition tree for the decision table $\mathbf{A}$

```
1 begin
2 |  Find the best template T in A (see Section 2.1)
3 |  Divide A in two subtables: A_1 containing all objects satisfying T and
  |  A_2 = A − A_1
4 |  if obtained subtables are of acceptable size in the sense of rough set
  |  methods then
5 |  |  STOP // The decomposition is finished
6 |  end
7 |  repeat lines 2-7 for all "too large" subtables
8 end
```

template and its contradiction are transferred as templates describing subtables to the next step of decomposition. Decomposition finishes when the subtables obtained are so small that the decision rules can be calculated for them using standard methods. After determining the decomposition tree, decision rule sets are calculated for all the leaves of this tree and, more precisely, for the subtables occurring in single leaves.

The tree and the rules calculated for training sample can be used in classification of unseen cases. Suppose we have a binary decomposition tree. Let u be a new object, $\mathbf{A}(T)$ be a subtable containing all objects matching a template T, and $\mathbf{A}(\neg T)$ be a subtable containing all objects not matching a template T. We classify object u starting from the root of the tree using Algorithm 2.4.

This algorithm works in such a way that such a leaf of a decomposition tree is sought first that the tested object matches the template describing the objects of

Algorithm 2.4. Classification by decomposition tree

```
1  begin
2  |  if u matches template T found for A then
3  |  |  go to subtree related to A(T)
4  |  else
5  |  |  go to subtree related to A(¬T)
6  |  end
7  |  if u is at the leaf of the tree then
8  |  |  go to line 12
9  |  else
10 |  |  repeat lines 2-11 substituting A(T) (or A(¬T)) for A
11 |  end
12 |  Classify u using decision rules for subtable attached to the leaf
13 end
```

that leaf. Next, the object is classified with the help of decision rules calculated for the leaf that was found.

The type of the decomposition method depends on the method of determining the best template. For instance, if decomposition is needed only because it is impossible to compute rules for a given decision table, then the best template for this table is the template which divides a given table into two equal parts. If, however, we are concerned with the table partition that is most compatible with the partition introduced by decision classes, then the measure of the template quality may be, for example, the number of pairs of objects from different decision classes, differentiated with the help of the partition introduced by a given template. Surely, the best template in this case is a template with the largest number of differentiated pairs.

The patterns determined may have different forms (see, *e.g.*, [165] for more details). In the simplest case, for a symbolic attribute, the best template might be of the forms $a(x) = v$ or $a(x) \neq v$ where $a \in A$, $v \in V_a$, and $x \in U$, whereas for a numerical attribute, the templates might be $a(x) > v$, $a(x) < v$, $a(x) \leq v$, or $a(x) \geq v$ where $a \in A$, $v \in V_a$, and $x \in U$.

The classifier presented in this section uses a binary decision tree, however, it should not be mistaken for C4.5 or ID3 (see, *e.g.*, [210, 243]) because, as we said before, rough set methods have been used in leaves of the decomposition tree in construction of the classifying algorithm.

2.8 Concept Approximation and Classifiers

Definability of concepts is a term well-known in classical logic (see, *e.g.*, [5, 244, 245]). In this classical approach a definable concept (set) is a relation on the domain of a given structure whose elements are precisely those elements satisfying some formula in the structure. Semantics of such formula enables to determine precisely for a given element (object) whether it belongs to the concept or not. However, the issue of definability of concepts is somewhat complicated by the pervasive presence of vagueness and ambiguity in natural language (see [126, 127, 244]). Therefore, in numerous applications, the concepts of interest may only be defined approximately on the basis of available, incomplete, imprecise or noisy information about them, represented, *e.g.*, by positive and negative examples (see [6, 7, 8, 9, 10, 11, 12, 13]). Such concepts are often called *vague* (imprecise) concepts. We say that a concept is vague when there may be cases (elements, objects) in which there is no clear fact of the matter whether the concept applies or not. Hence, the classical approach to concept definability known from classical logic cannot be applied for vague concepts. At the same time an approximation of a vague concept consists in construction of an algorithm (called a classifier) for this concept, which may be treated as a constructive, approximate description of the concept. This description enables to classify testing objects, that is, to determine for a given object whether it belongs to the concept approximated or not to which degree.

There is a long debate in philosophy on vague concepts (see, *e.g.*, [126, 127, 128]) and recently computer scientists (see, *e.g.*, [79, 82, 83, 246, 247, 248, 249]) as well

as other researchers have become interested in vague concepts. Since the classical approach to concept definability known from classical logic cannot be applied for vague concepts new methods of definability have been proposed. Professor Lotfi Zadeh (see [250]) introduced a very successful approach to definability of vague concepts. In this approach, sets are defined by partial membership in contrast to crisp membership used in the classical definition of a set. Rough set theory proposed a method of concept definability by employing the lower and upper approximation, and the boundary region of this concept (see Section 2.1). If the boundary region of a set is empty it means that a particular set is crisp, otherwise the set is rough (inexact). The non-empty boundary region of the set means that our knowledge about the set is not sufficient to define the set precisely. Using the lower and upper approximation, and the boundary region of a given concept a classifier can be constructed. Assume there is given a decision table $\mathbf{A} = (U, A, d)$, whose binary decision attribute with values 1 and 0 partitions the set of objects in two disjoint ones: C and C'. The set C contains objects with the decision attribute value equal to 1, and the set C' contains objects with the decision attribute value equal to 0. The sets C and C' may also be interpreted in such a way that the set C is a certain concept to be approximated and the set C' is the complement of this concept ($C' = U \setminus C$). If we define for concept C and its complement C', their A-lower approximations $\underline{A}C$ and $\underline{A}C'$, the A-upper approximation $\overline{A}C$, and the A-boundary $BN_A(C)$ ($BN_A(C) = \overline{A}C \setminus \underline{A}C$), we obtain a simple classifier which operates in such a way that a given testing object u is classified to concept C if it belongs to the lower approximation $\underline{A}C$. Otherwise, if object u belongs to the lower approximation $\underline{A}C'$, it is classified to the complement of concept C. However, if the object belongs neither to $\underline{A}C$ nor $\underline{A}C'$, but it belongs to $BN_A(C)$, then the classifier cannot make an unambiguous decision about membership of the object, and it has to respond that the object under testing simultaneously belongs to the concept C and its complement C', which means it is a border object. In this case the membership degree of a tested object $u \in U$ to concept $C \subseteq U$ is expressed numerically with the help of a *rough membership function* (see, *e.g.*, [16, 17]). The *rough membership function* μ_C quantifies the degree of relative overlap between the concept C and the equivalence class to which u belongs. It is defined as follows:

$$\mu_C(u) : U \to [0,1] \text{ and } \mu_C(u) = \frac{card([u]_{IND_{\mathbf{A}}(A)} \cap C)}{card([u]_{IND_{\mathbf{A}}(A)})}.$$

As we can see, in order to work the classifier described above, it is necessary for the tested object to belong to one of the equivalence classes of relation $IND_{\mathbf{A}}(A)$. However, there is one more instance remaining when the tested object does not belong to any equivalence class of relation $IND_{\mathbf{A}}(A)$. In such case, the classifier under consideration cannot make any decision about membership of the tested object and has to say: *"I do not know"*.

Unfortunately, the case when the tested object does not belong to any equivalence class of relation $IND_{\mathbf{A}}(A)$ frequently occurs in practical applications. It is due to the fact that if the objects under testing do not belong to the decision

table that was known at the beginning, but to its extension, the chances are small that in a given decision table, there exists an object (called a training object) whose conditional attribute values are identical to those in the testing object. However, it follows from the definitions of the relation $IND_{\mathbf{A}}(A)$ that the testing object for which there is no training object cannot be classified by the classifier described above. In such a case, one can say that the *extension* of this classifier is very small. For the above reason, the classic approach to classifying objects in the rough set theory (described above) requires generalization.

It is worth noticing that in machine learning and pattern recognition (see, *e.g.*, [6, 8, 9, 10, 11, 12, 13]), this issue is known under the term *learning concepts by examples* (see, *e.g.*, [10]). The main problem of learning concepts by examples is that the description of a concept under examination needs to be created on the basis of known examples of that concept. By creating a concept description we understand detecting such properties of exemplary objects belonging to this concept that enable further examination of examples in terms of their membership in the concept under examination. A natural way to solve the problem of learning concepts by examples is *inductive reasoning* (see, *e.g.*, [251, 252]). In inductive reasoning we assume as true the sentence stating a general regularity, at the same time we do that on the basis of acknowledging sentences stating individual instances of this regularity (see, *e.g.*, [251, 252]). This is the reasoning according to which decisions in the real world are often made relying on incomplete or even flawed information. This takes place in the cases of answers to questions connected with forecasting, checking hypotheses or making decisions.

In the case of the problem of learning concepts by examples, the usage of inductive reasoning means that while obtaining further examples of objects belonging to the concept (the so-called positive examples) and examples of objects not belonging to the concept (the so-called negative examples), an attempt is made to find such description that correctly matches all or almost all examples of the concept under examination.

From the theoretical point of view, in the rough set theory the classic approach to concept approximation was generalized by Professor Skowron and Professor Stepaniuk (see [253]). This approach is consistent with the philosophical view (see, *e.g.*, [126, 127]) and the logical view (see, *e.g.*, [128]). The main element of this generalization is an *approximation space*. The approximation space (see, *e.g.*, [246, 253, 254, 255]) is a tuple $AS = (U, I, \nu)$, where

- U is a non-empty set of objects,
- $I : U \to P(U)$ is an *uncertainty function* and $P(U)$ denotes the powerset of U,
- $\nu : P(U) \times P(U) \to [0, 1]$ is a *rough inclusion function*.

The uncertainty function I defines for every object $u \in U$ a set of objects indistinguishable with u or similar to u. The set $I(u)$ is called the neighborhood of u. If U is a set of objects of a certain decision table $\mathbf{A} = (U, A, d)$, then in the simplest case the set $I(u)$ may be the equivalence class $[u]_{IND_{\mathbf{A}}(A)}$. However, in a general case the set $I(u)$ is usually defined with the help of a special language such as GDL or NL (see Section 4.7).

The rough inclusion function ν defines the degree of inclusion of X in Y, where $X, Y \subseteq U$. In the simplest case, rough inclusion can be defined by:

$$\nu(X,Y) = \begin{cases} \frac{card(X \cap Y)}{card(X)} & \text{if } X \neq \emptyset \\ 1 & \text{if } X = \emptyset. \end{cases}$$

This measure is widely used by the data mining and rough set communities (see, *e.g.*, [16, 17, 246, 253]). However, rough inclusion can have a much more general form than inclusion of sets to a degree (see [192, 247, 249]).

It is worth noticing that in literature (see, *e.g.*, [247]) a *parameterized approximation space* is considered instead of the approximation space. Any parameterized approximation space consists of a family of approximation spaces creating the search space for data models. Any approximation space in this family is distinguished by some parameters. Searching strategies for optimal (sub-optimal) parameters are basic rough set tools in searching for data models and knowledge. There are two main types of parameters. The first ones are used to define object sets (neighborhoods), the second are measuring the inclusion or closeness of neighborhoods.

For an approximation space $AS = (U, I, \nu)$ and any subset $X \subseteq U$ the lower and the upper approximations are defined by:

- $LOW\,(AS, X) = \{u \in U : \nu\,(I\,(u), X) = 1\}$,
- $UPP\,(AS, X) = \{u \in U : \nu\,(I\,(u), X) > 0\}$, respectively.

The lower approximation of a set X with respect to the approximation space AS is the set of all objects which can be classified with certainty as object of X with respect to AS. The upper approximation of a set X with respect to the approximation space AS is the set of all objects which can be possibly classified as objects of X with respect to AS.

Several known approaches to concept approximations can be covered using the approximation spaces discussed here, *e.g.*, the approach given in [16, 17], approximations based on the variable precision rough set model (see, *e.g.*, [256]) or tolerance (similarity) rough set approximations (see, *e.g.*, [253]).

Similarly to the classic approach, the lower and upper approximation in the approximation space AS for a given concept C may be used to classify objects to this concept. In order to do this one may examine the membership of the tested objects to $LOW\,(AS, C)$, $LOW\,(AS, C')$ and $UPP\,(AS, C) \setminus LOW\,(AS, C)$.

However, in machine learning and pattern recognition (see, *e.g.*, [6, 8, 9, 10, 11, 12, 13]), we often search for approximation of a concept $C \subseteq U^*$ in an approximation space $AS^* = (U^*, I^*, \nu^*)$ having only a partial information about AS^* and C, i.e., information restricted to a sample $U \subseteq U^*$. Let us denote the restriction of AS^* to U by $AS = (U, I, \nu)$, i.e., $I(x) = I^*(x) \cap U$, $\nu(X, Y) = \nu^*(X, Y)$ for $x \in U$, and $X, Y \subseteq U$ (see Fig. 3).

To decide if a given object $u \in U^*$ belongs to the lower approximation or to the upper approximation of $C \subseteq U^*$, it is necessary to know the value $\nu^*(I^*(u), C)$. However, in the case there is only partial information about the approximation space AS^* available, one must make an estimation of such a value $\nu^*(I^*(u), C)$

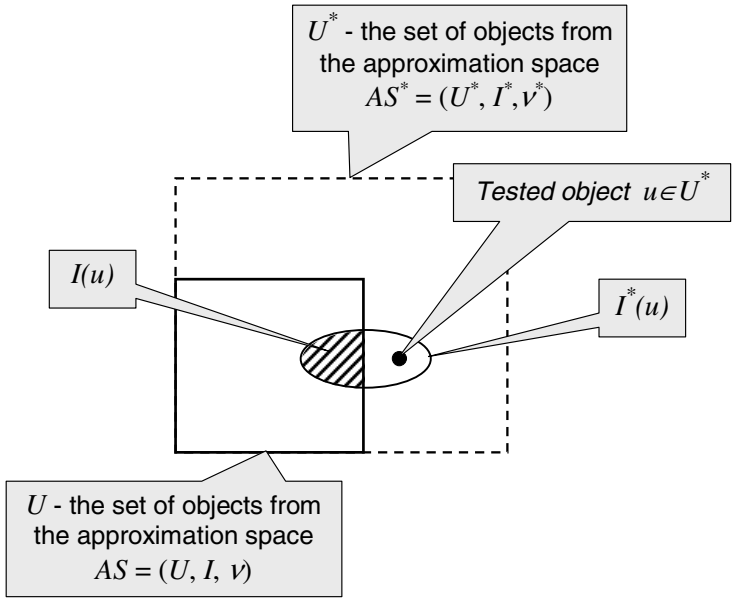


Fig. 3. An approximation space AS and its extension AS^*

rather than its exact value. In machine learning, pattern recognition or data mining, different heuristics are used for estimation of the values of ν^*. Using different heuristic strategies, values of another function ν' are computed and they are used for estimation of values of ν^*. Then, the function ν' is used for deciding if objects belong to C or not. Hence, we define an approximation of C in the approximation space $AS' = (U^*, I^*, \nu')$ rather than in $AS^* = (U^*, I^*, \nu^*)$. Usually, it is required that the approximations of $C \cap U$ in AS and AS' are close (or the same).

The approach presented above (see, *e.g.*, [83, 246, 248, 249]) became an inspiration for finding out of a number of methods which would enable to enlarge the extension of constructed classifiers, that is, to make the classifiers under construction to be able to classify any objects, and not only those belonging to a given decision table.

Some other issues concerning the rough set approach to vague concept approximation are discussed, e.g., in [83, 128, 248, 249]. Among these issues are the higher order vagueness (i.e., nondefinability of boundary regions), adaptive learning of concept approximation, concept drift, and sorites paradoxes.

One of the basic ways of increasing the extension of classifiers is to approximate the concepts not with the help of the equivalence class of relation IND (see above) but with the help of the patterns of the established language which different objects may match, both from the training table and its extension. A given object matches the pattern if it is compatible with the description of this pattern. Usually, the pattern is constructed in such a way that all or almost

all its matching objects belong to the concept under study (the decision class). Moreover, it is required that the objects from many equivalence classes of relation IND could match the patterns. Thus, the extension of classifiers based on patterns is dramatically greater than the extension of classifiers working on the basis of equivalence classes of relation IND. These types of patterns are often called *decision rules* (see Section 2.3). In literature one may encounter many methods of computing decision rules from data and methods enabling preprocessing the data in order to construct effective classifiers. Into this type of methods one may include, for example, discretization of attribute values (see Section 2.2), methods computing decision rules (see Section 2.3), shortening and generalization of decision rules (see Section 2.5).

The determined decision rules may be applied to classifiers construction. For instance, let us examine the situation, when a classifier is created on the basis of decision rules from the set $RUL(\mathbf{A})$ computed for a given decision table $\mathbf{A} = (U, A, d)$, and at the same time decision attribute d describes the membership to a certain concept C and its complement C'. [1]

The set of rules $RUL(\mathbf{A})$ is the sum of two subsets $RUL(\mathbf{A}, C)$ and $RUL(\mathbf{A}, C')$, where $RUL(\mathbf{A}, C)$ is the set of rules classifying objects to C and $RUL(\mathbf{A}, C')$ is a set of rules classifying objects to C'. For any tested object u, by $MRul(\mathbf{A}, C, u) \subseteq RUL(\mathbf{A}, C)$ and $MRul(\mathbf{A}, C', u) \subseteq RUL(\mathbf{A}, C')$ we denote sets of such rules whose predecessors match object u and classify objects to C and C', respectively.

Let $AS = (U, I, \nu)$ be an approximation space, where:

1. $\forall u \in U : I(u) = \bigcup_{r \in MRul(\mathbf{A}, C, u)} Supp_{\mathbf{A}}(r) \cup \bigcup_{r \in MRul(\mathbf{A}, C', u)} Supp_{\mathbf{A}}(r)$
2. $\forall X, Y \subseteq U : \nu(X, Y) = \begin{cases} \frac{card(X \cap Y)}{card(X)} & \text{if } X \neq \emptyset \\ 1 & \text{if } X = \emptyset. \end{cases}$

The above approximation space AS may be extended in a natural way to approximation space $AS' = (U^*, I^*, \nu')$, where:

1. $I^* : U^* \longrightarrow P(U^*)$ such that $\forall u \in U : I^*(u) = I(u)$,
2. $\forall X, Y \subseteq U^* : \nu'(X, Y) = \begin{cases} \frac{card(X \cap Y)}{card(X)} & \text{if } X \neq \emptyset \\ 1 & \text{if } X = \emptyset. \end{cases}$

Let us notice that such a simple generalization of functions I to I^* and ν to ν' is possible because function I may determine the neighborhood for a given object belonging to U^*. It results from the fact that decision rules from set $RUL(\mathbf{A})$ may recognize objects not only from set U but also from set $U^* \setminus U$. Approximation space AS' may now also be used to construct a classifier which classifies objects from set U^* to concept C or its complement C'. In creating such a classifier the key problem is to resolve the conflict between the rules

[1] For simplicity of reasoning we consider only binary classifiers, i.e. classifiers with two decision classes. One can easily extend the approach to the case of classifiers with more decision classes.

classifying the tested object to the concept or to its complement. Let us notice that this conflict occurs because in practice we do not know function ν^* but only its approximation ν'. That is why, there may exist such a tested object u_t that the values $\nu'(\{u_t\}, C)$ and $\nu'(\{u_t\}, C')$ are high (that is close to 1), while values $\nu^*(\{u_t\}, C)$ and $\nu^*(\{u_t\}, C')$ are very different (*e.g.*, $\nu^*(\{u_t\}, C)$ is close to 1 and $\nu^*(\{u_t\}, C')$ is close to 0).

Below, we present the definition of such a classifier in the form of a function that returns the value YES when the tested object belongs to C or the value NO when the tested object belongs to C':

$$\forall u \in U : Classifier(u) = \begin{cases} YES & \text{if } \nu'(\{u\}, C) > 0.5 \\ NO & \text{otherwise.} \end{cases} \tag{5}$$

Obviously, other rough inclusion functions may be defined (see, *e.g.*, [192, 247, 249]). Thus, we obtain different classifiers. Unfortunately, a classifier defined with the help of Equation (5) is impractical because the function ν' used in it does not introduce additional parameters which enable to recognize of objects to the concept and its complement whereas in practical applications in constructing classifiers based on decision rules, functions are applied which give the strength (weight) of the classification of a given tested object to concept C or its complement C' (see, *e.g.*, [196, 199, 216, 217, 241]). Below, we present a few instances of such weights (see [199]).

1. A *simple strength* of decision rule set is defined by

$$SimpleStrength(C, u_t) = \frac{card(MRul(\mathbf{A}, C, u_t))}{card(RUL(\mathbf{A}, C))}.$$

2. A *maximal strength* of decision rule set is defined by

$$MaximalStrength(C, u_t) = max_{r \in MRul(\mathbf{A}, C, u_t)} \left\{ \frac{Supp_{\mathbf{A}}(r)}{card(C)} \right\}.$$

3. A *basic strength* or *a standard strength* of decision rule set is defined by

$$BasicStrength(C, u_t) = \frac{\sum\limits_{r \in MRul(\mathbf{A}, C, u_t)} Supp_{\mathbf{A}}(r)}{\sum\limits_{r \in RUL(\mathbf{A}, C)} Supp_{\mathbf{A}}(r)}.$$

4. A *global strength* of decision rule set is defined by

$$GlobalStrength(C, u_t) = \frac{card\left(\bigcup\limits_{r \in MRul(\mathbf{A}, C, u_t)} Supp_{\mathbf{A}}(r) \right)}{card(C)}.$$

Using each of the above rules weight, a rough inclusion function corresponding to it may be defined. Let us mark any established weight of rule sets as S. For weight S we define an exemplary rough inclusion function ν_S in the following way:

$$\forall X,Y \subseteq U : \nu_S(X,Y) = \begin{cases} 0 & \text{if } Y = \emptyset \wedge X \neq \emptyset \\ 1 & \text{if } X = \emptyset \\ \frac{S(Y,u)}{S(Y,u)+S(U\setminus Y,u)} & \text{if } X = \{u\} \text{ and} \\ & \quad S(Y,u) + S(U \setminus Y,u) \neq 0 \\ \frac{1}{2} & \text{if } X = \{u\} \text{ and} \\ & \quad S(Y,u) + S(U \setminus Y,u) = 0 \\ \frac{\sum_{u \in X} \nu_S(\{u\},Y)}{card(X)} & \text{if } card(X) > 1 \end{cases},$$

where for an established set Y and object u the weights $S(Y,u)$ and $S(U \setminus Y,u)$ are computed using the decision rule set generated for table $\mathbf{A} = (U, A, d_Y)$, where attribute d_Y describes the membership of objects from U to the set Y.

The rough inclusion function defined above may be used to construct the classifier as it is done in Equation (5). Such a classifier executes a simple negotiation method between the rules classifying the tested object to the concept and rules classifying the tested object to the complement of the concept (see Section 2.6). It simply is based on classifying tested object u to concept C only when with the established weight of rule sets S the value $\nu_S(\{u\}, C)$ is bigger than $\nu_S(\{u\}, C')$. Otherwise, object u is classified to the complement of concept C.

In this paper, the weight *BasicStrength* is used in experiments related to construction of classifiers based on decision rules to resolve conflicts between rule sets.

2.9 Evaluation of Classifiers

In order to evaluate the classifier quality in relation to the data analyzed, a given decision table is partitioned into the two tables in a general case (see, *e.g.*, [11, 257, 258]):

1. *the training table* containing objects on the basis of which the algorithm learns to classify objects to decision classes,
2. *the test table*, by means of which the classifier learned on the training table may be evaluated when classifying all objects belonging to this table.

The numerical measure of the classifier evaluation is often the number of mistakes made by the classifier during classification of objects from the test table in comparison to all objects under classification (*the error rate*, see, *e.g.*, [11, 196, 198]). However, the method of the numerical classifier evaluation, used most often, is the method based on a *confusion matrix*. The *confusion matrix* (see, *e.g.*, [15, 257, 259]) contains information about actual and predicted classifications done by a classifier. Performance of such systems is commonly evaluated using the data in the matrix. The Table 1 shows the confusion matrix for a two class classifier, i.e., for a classifier constructed for a concept.

Table 1. The confusion matrix

		Predicted	
		Negative	Positive
Actual	Negative	*TN*	*FP*
	Positive	*FN*	*TP*

The entries in the confusion matrix have the following meaning in the context of our study (see, *e.g.*, [260]):

- TN (*True Negatives*) is the number of correct predictions that an object is a negative example of a concept of the test table,
- FP (*False Positives*) is the number of incorrect predictions that an object is a positive example of a concept of the test table,
- FN (*False Negatives*) is the number of incorrect predictions that an object is a negative example of a concept of the test table,
- TP (*True Positives*) is the number of correct predictions that an object is a positive example of a concept of the test table.

Several standard terms (parameters) have been defined for the two class confusion matrix:

- the *accuracy* (ACC) defined for a given classifier by the following equality:

$$ACC = \frac{TN + TP}{TN + FN + FP + TP},$$

- the *accuracy for positive examples* or the *sensitivity* (see, *e.g.*, [260]) or the *true positive rate* (TPR) (see, *e.g.*, [257]) defined for a given classifier by the following equality:

$$TPR = \frac{TP}{TP + FN},$$

- the *accuracy for negative examples* or the *specificity* (see, *e.g.*, [260]) or the *true negative rate* (TNR) (see, *e.g.*, [257]) defined for a given classifier by the following equality:

$$TNR = \frac{TN}{TN + FP}.$$

An essential parameter is also the number of classified objects from the test table in comparison to the number of all objects from this table since classifiers may not always be able to classify the objects. This parameter, called the *coverage* (see, *e.g.*, [11, 15]), may be treated as an extension measure of the classifier. Thus, in order to evaluate classifiers, also the following numerical parameters are applied in this paper:

1. the *coverage* (COV) defined for a given classifier by the following equality:

$$COV = \frac{TN + FP + FN + TP}{\text{the number of all objects of the test table}},$$

2. the *coverage for positive examples* ($PCOV$) defined for a given classifier by the following equality:

$$PCOV = \frac{FN + TP}{\text{the number of all positive examples of a concept of the table}},$$

3. the *coverage for negative examples* ($NCOV$) defined for a given classifier by the following equality:

$$NCOV = \frac{TN + FP}{\text{the number of all negative examples of a concept of the table}},$$

4. the *real accuracy* defined for a given classifier by: $ACC \cdot COV$,
5. the *real accuracy for positive examples* or the *real true positive rate* defined for a given classifier by: $TPR \cdot PCOV$,
6. the *real accuracy for negative examples* or the *real true negative rate* defined for a given classifier by: $TNR \cdot NCOV$.

Besides that, in order to evaluate classifiers still different parameters are applied. These are, for instance, time of construction of a classifier on the basis of a training table or the complexity degree of the classifier under construction (*e.g.*, the number of generated decision rules).

In summary, in this paper the main parameters applied to the evaluation of classifiers are: the accuracy, the coverage, the real accuracy, the accuracy for positive examples, the coverage for positive examples, the real accuracy for positive examples, the accuracy for negative examples, the coverage for negative examples and the real accuracy for negative examples.

They are used in experiments with AR schemes (see Section 5.8) and experiments related to detecting behavioral patterns (see Section 6.25 and Section 6.26). However, in experiments with automated planning another method of classifier quality evaluation was applied (see Section 7.21). It results from the fact that this case is about automated generating the value of complex decision that is a plan which is a sequence of actions alternated with states. Hence, to compare this type of complex decision values the above mentioned parameters may not be used. Therefore, to compare the plans generated automatically with the plans available in the data set we use a special classifier based on concept ontology which shows the similarity between any pair of plans (see Section 7.18).

It is worth noticing that in literature there may be found another, frequently applied method of measuring the quality of created classifiers. It is a method based on ROC curve (*Receiver Operating Characteristic curve*) (see, *e.g.*, [260, 261, 262]). This method is available, for instance, in system ROSETTA (see, *e.g.*, [259, 263, 264]). It is also worthwhile mentioning that the author of this paper participated in construction of programming library RSES-lib, creating the computational kernel of system ROSETTA (see [230, 259] for more details).

In order not to make the value of the determined parameter of the classifier evaluation depending on a specific partitioning the whole decision table into a training and test parts, a number of methods are applied which perform tests to determine which parameter values of the classifier evaluation are creditable.

The methods of this type applied most often are *train-and-test* and *cross-validation* (see, *e.g.*, [11, 258, 265]). The *train-and-test* method is usually applied to decision tables having at least 1000 objects (see, *e.g.*, [11]). It consists in a random isolation of two test subtables from the whole data available, treating one of them as a training subtable and the other as a test subtable. The training and test subtables are usually separated (although not always) and altogether make the available decision table. It is crucial, however, that at least some part of the objects from the test subtable does not occur in the training subtable. The proportion between the number of objects in the test and training subtables depends on a given experiment but it is usually such that the number of objects in the test part constitutes from 20 to 50 percent of the number of objects in the whole data available (see, *e.g.*, [11]). The *cross-validation* method is applied to evaluate a classifier when the number of objects in the decision table is less than 1000 (see, *e.g.*, [11]). This method consists in partitioning data in a random way into m equal parts and, then performing m experiments with them. In each of these experiments, a local coefficient of the classifier evaluation is calculated for a situation when one of the parts into which the data was divided is a set of tested objects, and the remaining $m - 1$ parts (temporarily combined) are treated as a set of training objects. Finally, a coefficient of the classifier evaluation as an average arithmetical coefficient of all experiments is calculated. The number m is determined depending on the specific data and should be selected in such a way that the test parts not to have too few objects. In practice, m is an integer ranging from 5 to 15 (see, *e.g.*, [11]).

All decision tables used in experiments have more than 1000 objects, in this paper. That is why in order to determine the parameter of the classifier quality the *train-and-test* method is always applied. Moreover, each experiment is repeated 10 times for ten random partitions into two separate tables (training and test). Hence, the result of each experiment is the arithmetical mean obtained from the results of its repetitions. Additionally, the standard deviation of the received result is given.

2.10 Problem of Low Coverage

If a given tested object matches the predecessor of a certain basic decision rule (that is the values of condition attributes of this object are the same as the values of the descriptors from the rule predecessor corresponding to them), then this rule may be used to classify this object, that is, the object is classified to the decision class occurring in the rule successor. In this case we also say that a given tested object is recognized by a certain decision rule. However, if a given tested object is recognized by different decision rules which classify it to more than one decision classes, then negotiation methods between rules are applied (see Section 2.6 and Section 2.8).

In practice, it may happen that a given tested object does not match the predecessor of any of the available decision rules. We say that this object is not recognized by a given classifier based on decision rules and what follows it cannot be classified by this classifier. It is an unfavorable situation, for we often expect

from the classifiers to classify all or almost all tested objects. If there are many of the unclassified objects, then we say that a given classifier has too low an extension. It is expressed numerically by a low value of the coverage parameter (see Section 2.9).

A number of approaches which enable to avoid a low coverage of classifiers based on decision rules were described in literature. They are for example:

1. The application of classifiers based on the set of all rules with a minimum number of descriptors (see Section 2.4) which usually have a high extension (see, *e.g.*, [196, 198]).
2. The application of rule classifiers constructed on the basis of covering algorithms and partial matching mechanism of the objects to the rules (see, *e.g.*, [10, 213, 214, 216, 217, 222, 223, 266]).
3. The application of classifiers based on decision rules which underwent the process of generalization of rules owing to which the classifier extension usually increases (see Section 2.5).
4. The application of classifiers based on a lazy learning which does not require preliminary computation of decision rules, for decision rules needed for object classification are discovered directly in a given decision table during the classification of the tested object (see, *e.g.*, [197, 198, 267]).

All the methods mentioned above have their advantages and disadvantages. The first method has an exponential time complexity which results from the complexity of the algorithm computing all reducts (see Section 2.4). The second method is very quick, for it is based on rules computed with the help of the covering method. However, in this method there are often applied approximation rules to classify objects (determined as a result of a partial matching objects to the rules). Therefore, the quality of classification on the basis of such rules may be unsatisfactory. The third method uses the operation of rule generalization. Owing to this operation the extension of the obtained rules increases. However, it does not lead to such a high extension as in the case of the first, second and fourth method. Apart from that the operation of rule generalization is quite time consuming. Whereas, the fourth method, although does not require preliminary computation of decision rules, its pessimistic computational time complexity of each tested object classification is of order $O(n^2 \cdot m)$, where n is the number of objects in the training table and m is the number of condition attributes. Hence, for bigger decision tables this method cannot be applied effectively.

There is one more possibility remaining to build classifiers on the basis of rules computed with the covering method without using partial matching tested objects to the rules. Obviously, classifiers based on such rules may have a low coverage. However, they usually have a high quality of the classification. It is extremely crucial in many applications (for example in medical and financial ones) where it is required that the decisions generated by classifiers be always or almost always correct. In such applications it is sometimes better for the classifier to say *I do not know* rather than make a wrong decision. That is why in this paper we use classifiers based on rules computed with covering method (without partial matching objects to the rules) agreeing on a low coverage of such

classifiers in cases when classifiers based on the set of all rules with minimum number of descriptors cannot be applied (too large analyzed decision tables).

3 Methods of Constructing Stratifying Classifiers

The algorithm of concept approximation, presented in Subsection 2.8, consists in classifying the tested objects to the lower approximation of this concept, the lower approximation of complement of this concept or its border. Many methods enabling increase of the extension of classifiers under construction, in rough set theory are proposed (see Section 2.8). Discretization of attribute values (see Section 2.2), methods of calculating and modifying decision rules (see Sections 2.3, 2.4, 2.5), and partial matching method (see Section 2.10) are examples of such methods. As a result of applying these methods, there are constructed classifiers able to classify almost every tested object to the concept or its complement.

At first glance this state of affairs should dispose optimistically for approximation methods can be expanded for tested objects from beyond a given decision table, which is necessary in inductive learning (see Section 2.8). Unfortunately, such a process of generalizing concept approximation encounters difficulties in classifying new (unknown during the classifier learning) tested objects. Namely, after expanding the set of objects U of a given information system with new objects, equivalence classes of these objects are often disjoint with U. This means that if such objects match the description of a given concept C constructed on the basis of set U, this match is often incidental. Indeed due to the unfamiliarity the process generalization of decision rules may go too far (*e.g.*, decision rules are too short) because of absence of these new objects when the concept description was created. It may happen that the properties (attributes) used to describe a concept are chosen in wrong way. So, if a certain tested object from outside the decision table is classified, then it may turn out that, in the light of the knowledge gathered in a given decision table, this object should be classified neither to the concept nor to its complement but to the concept border, which expresses our uncertainty about the classification of this object. Meanwhile, most of the classifiers currently constructed classify the object to the concept or its complement. A need of use the knowledge from a given table arises in order to determine the coefficient of certainty that the object under testing belongs to the approximated concept. In other words, we would like to determine, with reference to the tested object, how certain the fact is that this object belongs to the concept. And at the same time it would be the best to express if the certainty coefficient by a number, *e.g.*, from $[0, 1]$. In literature such a numerical coefficient is expressed using different kinds of rough membership functions (see Section 2.8). If a method of determining such a coefficient is given, it may be assumed that the coefficient values are discretized which leads to a sequence of concept layers arranged linearly. The first layer in this sequence represents objects which, without any doubt do not belong to the concept (the lower approximation of the concept complement). The next layers in the sequence represent objects belonging to the

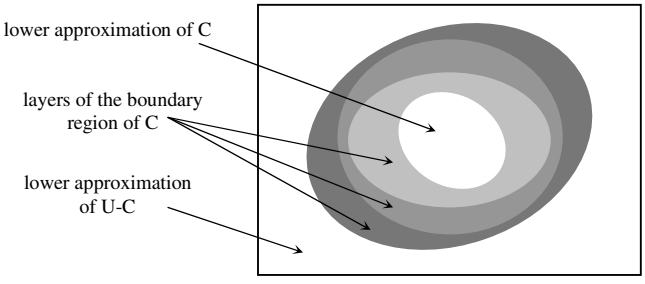


Fig. 4. Layers of a given concept C

concept more and more certainly (border layers of the concept). The last layer in this sequence represents objects certainly belonging to the concept, that is, the ones belonging to the lower concept approximation (see Fig. 4).

Let us add that this type of concept layers may be defined both on the basis of the knowledge gathered in data tables and using additional domain knowledge provided by experts.

3.1 Stratifying Classifier

In order to examine the membership of tested objects to individual concept layers, such classifiers are needed that can approximate all layers of a given concept at the same time. Such classifiers are called in this paper *stratifying classifiers*.

Definition 1 (*A stratifying classifier*)**.** *Let* $\mathbf{A} = (U, A, d)$ *be a decision table whose objects are positive and negative examples of a concept* C *(described by a binary attribute* d*).*

1. *A partition of the set* U *is a family* $\{U_1, ..., U_k\}$ *of non-empty subsets of the set* U *(where* $k > 1$*) such that the following two conditions are satisfied:*
   (a) $U = U_1 \cup ... \cup U_k$*,*
   (b) $\forall_{i \neq j}\ U_i \cap U_j = \emptyset$*.*
2. *A partition of the set* U *into a family* $U_C^1, ..., U_C^k$ *we call the partition of* U *into layers in relation to concept* C *when the following three conditions are satisfied:*
   (a) set U_C^1 *includes objects which, according to an expert, certainly do not belong to concept* C *(so they belong to a lower approximation of its complement),*
   (b) for every two sets U_C^i, U_C^j *(where* $i < j$*), set* U_C^i *includes objects which, according to an expert, belong to concept* C *with a degree of certainty lower than the degree of certainly of membership of objects of* U_C^j *in* U*,*
   (c) set U_C^k *includes objects which, according to an expert, certainly belong to concept* C*, viz., to its lower approximation.*
3. *Each algorithm which assigns (classifies) tested objects into one of the layers belonging to a partition of the set* U *in relation to the concept* C*, is called a stratifying classifier of the concept* C*.*

4. *In practice, instead of using layer markings* $U_C^1, ..., U_C^k$*, elements of the set* $E = \{e_1, ..., e_k\}$ *are used to label layers, whereas the stratifying classifier constructed for the concept* C *which classifies each tested object into one of the layers labeled with labels from the set* E*, is denoted by* μ_C^E*.*
5. *If the stratifying classifier* μ_C^E *classifies a tested object* u *into the layer labeled by* $e \in E$*, then this fact is denoted by the equality* $\mu_C^E(u) = e$*.*

An expert may divide the set of objects U into layers in two following ways:

1. by an assignment of weight labels to all training objects arbitrary (see Section 3.2),
2. by providing heuristics which may be applied in construction of a stratifying classifier (see Section 3.3).

Stratifying classifiers can be very useful when we need to estimate realistically what the certainty of membership of a tested object to a concept is, without determining whether the object belongs to the concept or not. Apart from that, stratifying classifiers may be used to construct the so-called production rules (see Section 5.3).

In the paper, two general ways of construction of stratifying classifiers are presented. The first one is the *expert approach* consisting in defining by an expert an additional attribute in data which describes the membership of objects to particular layers of the concept. Next, a classifier differentiating layers as decision classes is built (see Section 3.2).

The second approach is called the *automatic approach* and is based on designing algorithms which are extensions of classifiers enabling the classification of objects into layers of a concept on the basis of certain premises and experimental observations (see Section 3.3).

3.2 Stratifying Classifiers Based on the Expert Approach

In construction of stratifying classifiers using expert knowledge, it is assumed that for all training objects not only the binary classification of training objects to a concept or outside the concept is known but we also know the assignment of all training objects into the specific concept layers. Using this approach an additional knowledge needs to be gained from a domain knowledge. Owing to that a classical classifier may be built (*e.g.*, the one based on a set of rules with a minimal number of descriptors) which directly classifies the objects to different concept layers. This classifier is built on the basis of a decision attribute which has as many values as many concept layers are there, and each of these values is a label of one of the layers.

3.3 Stratifying Classifiers Based on the Automatic Approach

In construction of stratifying classifiers using the automatic approach, the assignment of all training objects to specific concept layers is unknown but we only know the binary classification of training objects to a concept or its complement. However, the performance of a stratifying classifier is, in this case,

connected with a certain heuristics which supports discernibility of objects belonging to a lesser or greater degree to the concept, that is, objects belonging to different layers of this concept. Such a heuristic determines the way an object is classified to different layers and, thus, it is called a *stratifying heuristic*.

Many different types of heuristics stratifying concepts may be proposed. These may be, *e.g.*, heuristics based on the difference of weights of decision rules classifying tested objects to concept and its complement or heuristics using a k-NN algorithm of k nearest neighbors (compare with [78, 200, 268]). In this paper, however, we are concerned with a new type of stratifying heuristics using the operation of decision rule shortening (see Section 2.5).

The starting point of the presented heuristics is the following observation. Let us assume that for a certain consistent decision table $\mathbf{A}$ whose decision is a binary attribute with values 1 (objects belonging to the concept C) and 2 (objects belonging to the complement of concept C which is denoted by C'), a set of decision rules, $RUL(\mathbf{A})$ was calculated. The set $RUL(\mathbf{A})$ is the sum of two separate subsets of rules $RUL_1(\mathbf{A})$ (classifying objects to C) and $RUL_2(\mathbf{A})$ (classifying objects to C'). Now, let us shorten the decision rules from $RUL_1(\mathbf{A})$ to obtain the coefficient of consistency equal to 0.9 by placing the shortened decision rules in the set $RUL_1(\mathbf{A}, 0.9)$. Next, let $RUL'(\mathbf{A}) = RUL_1(\mathbf{A}, 0.9) \cup RUL_2(\mathbf{A})$. In this way, we have increased the extension of the input decision set of rules $RUL(\mathbf{A})$ in relation to the concept C, viz., as a result of shortening of the rules, the chance is increased that a given tested object is recognized by the rules classifying to the concept C. In other words, the classifier based on the set of rules $RUL'(\mathbf{A})$ classifies objects to the concept C more often.

Now, if a certain tested object u, not belonging to table $\mathbf{A}$, is classified to C' by the classifier based on the rule set $RUL'(\mathbf{A})$, then the chance that object u actually belongs to C' is much bigger than in the case of using the set of rules $RUL(\mathbf{A})$. The reason for this assumption is the fact that it is harder for a classifier based on the set of rules $RUL'(\mathbf{A})$ to classify objects to C' for the rules classifying objects to C are shortened in it and owing to that they recognize the objects more often. If, however, an object u is classified to C', then some of its crucial properties identified by the rules classifying it to C' must determine this decision. If shortening of the decision rules is greater (to the lower consistency coefficient), then the change in the rule set extension will be even bigger.

Summing up the above discussion, we conclude that rule shortening makes it possible to change the extensions of decision rule sets in relation to chosen concepts (decision classes), and owing to that one can obtain a certain type of approximation based on the certainty degree, concerning the membership of tested objects to the concept under consideration where different layers of the concept are modeled by applying different coefficients of rule shortening.

In construction of algorithms producing stratifying classifiers based on shortening of decision rules, there occurs a problem with the selection of accuracy coefficient threshold to which decision rules are shortened. In other words, what we mean here is the range and the step with which the accuracy coefficient threshold must be selected in order to obtain sets of rules enabling an effective

description of the actual layers of the concept approximated. On the basis of previous experimental experience (see, *e.g.*, [196, 198]), in this paper, we establish that the shortening thresholds of decision rule consistency coefficient are selected from the range 0.5 to 1.0. The lower threshold limit (that is, 0.5) results from the experimental observation that if we shorten rules classifying objects to a certain concept C below the limit 0.5 (without simultaneous shortening of rules classifying objects to C'), then although their extension increases dramatically (they classify objects to the concept C very often), their certainty falls to an absolutely unsatisfactory level. However, the upper limit of the threshold (that is 1.0) simply means leaving only accurate rules in the set of rules, and rejecting other approximation rules which could have occurred for a given decision table.

If it comes, however, to the change step of the chosen threshold of the rule coefficient of consistency, we set it at 0.1. This change step is dictated by the fact that it enables a general search of thresholds from 0.5 to 1.0 and, at the same time, the number of rule shortening operations is not too high which is essential for keeping the time needed to conduct computer experiments within acceptable bounds.

Now we present an algorithm of a stratifying classifier construction based on rule shortening (see Algorithm 3.1).

Let us notice that after the above algorithm completes its performance on the list L, there are eleven decision rule sets. The first classifier on this list contains the

Algorithm 3.1. Stratifying classifier construction

Input: decision table $\mathbf{A} = (U, A, d)$ and concept $C \subseteq U$
Output: classifier list L representing a stratifying classifier

```
1  begin
2  |  Calculate decision rules for table A, denoted by RUL(A) =
   |  RUL_1(A) ∪ RUL_2(A)
3  |  Create empty classifier list L
4  |  for a := 0.5 to 0.9 with step 0.1 do
5  |  |  Shorten rules RUL_1(A) to the consistency coefficient a and place
   |  |  the shortened decision rules in RUL_1(A, a)
6  |  |  RUL := RUL_1(A, a) ∪ RUL_2(A)
7  |  |  Add RUL to the end of the list L
8  |  end
9  |  Add RUL(A) to the end of the list L
10 |  for a := 0.9 to 0.5 with step 0.1 do
11 |  |  Shorten rules RUL_2(A) to the consistency coefficient a and place
   |  |  the shortened decision rules the RUL_2(A, a)
12 |  |  RUL := RUL_1(A) ∪ RUL_2(A, a)
13 |  |  Add RUL to the end of the list L
14 |  end
15 |  return L
16 end
```

Algorithm 3.2. Classification using the stratifying classifier

Input:
1. classifier list L representing a stratifying classifier,
2. set of labels of layers $E = \{e_1, ..., e_{size(L)+1}\}$,
3. tested object u

Output: The label of the layer to which the object u is classified

```
1 begin
2 |  for i := size(L) down to 1 do
3 |  |  Classify u using the classifier L[i]
4 |  |  if u is classified by L[i] to the concept C then
5 |  |  |  return e_{i+1}
6 |  |  end
7 |  end
8 |  return e_1
9 end
```

most shortened rules classifying to C. That is why, if it classifies an object to C', the degree of certainty is the highest that this object belongs to concept C', whereas the last classifier on the list L, contains the most shortened rules classifying to C'. That is why the classification of an object to the concept C using this classifier gives us the highest degree of certainty of that the object really belongs to C.

The time complexity of Algorithm 3.1 depends on the time complexity of the chosen algorithm of decision rules computing and on the algorithm of approximate rules synthesis (see Section 2.5).

On the basis of the classifier constructed according to Algorithm 3.1, the tested object is classified to a specific layer with the help of successive classifiers starting from the last to the first one, and if the object is classified by the i-th classifier to C, then we learn about membership of the object under testing to the $(i+1)$-th layer of C. However, if the object is not classified to C by any classifier, we learn about membership of the tested object to the first layer (layer number 1), that is, to the complement of concept C. We present a detailed algorithm for classification of the object using the stratifying classifier (see Algorithm 3.2).

Let us notice that if the size of the list L is equal to 11 (generated by Algorithm 3.1), the above classifier classifies objects to 12 concept layers where the number 12 layer is the layer of objects with the highest degree of certainty of membership to the concept and the layer number 1 is the layer with the lowest degree of certainty of membership to this concept.

4 General Methodology of Complex Concept Approximation

Many real-life problems may be modeled with the help of the so-called *complex dynamical systems* (see, *e.g.*, [92, 93, 94, 95, 96, 97]) or, putting it in an other

way, *autonomous multiagent systems* (see, *e.g.*, [98, 101]) or *swarm systems* (see, *e.g.*, [104]). These are sets consisting of complex objects which are characterized by the constant change of parameters of their components over time, numerous relationships among the objects, the possibility of cooperation/competition among the objects and the ability of objects to perform more or less complicated actions. Examples of systems of these kind are: *traffic*, *a patient observed during treatment*, *a team of robots during performing some task*. The description of the dynamics of such a system is often impossible using purely classical analytical methods, and the description itself contains many vague concepts. For instance, in order to monitor complex dynamical systems effectively, complex spatio-temporal concepts are used very often, concerning dynamic properties of complex objects occurring in these systems. These concepts are expressed in natural language on a much higher level of abstraction than the so-called sensor data, which have mostly been applied to approximation of concepts so far. Examples of such concepts are *safe car driving*, *safe overtaking*, *patient's behavior when faced with a life threat*, *ineffective behavior of robot team*.

Much attention has been devoted to spatio-temporal exploration methods in literature (see, *e.g.*, [63, 64]). The current experience indicates more and more that approximation of such concepts requires a support of knowledge of the domain to which the approximated terms are applied, i.e., the *domain knowledge*. It usually means the knowledge about concepts occurring in a given domain and various relations among these concepts. This knowledge exceeds significantly the knowledge gathered in data sets; it is often represented in a natural language, and it is usually obtained in a dialogue with an expert from a given domain (see, *e.g.*, [41, 42, 43, 44, 45, 46, 52, 269]). One of the methods of representing this knowledge is recording it in the form of the so-called *concept ontology*. The concept ontology is usually understood as a finite set of concepts creating a hierarchy and relationships among these concepts which connect concepts from different hierarchical levels (see next section). In this subsection, we present a general methodology of approximating complex spatio-temporal concepts on the basis of experimental data and a domain knowledge represented mainly by a concept ontology.

4.1 Ontology as a Representation of Domain Knowledge

The word *ontology* was originally used by philosophers to describe a branch of metaphysics concerned with the nature and relations of being (see, *e.g.*, [270]). However, the definition of ontology itself has been a matter of dispute for a long time, and controversies concern mainly the thematic scope to be embraced by this branch. Discussions on the subject of ontology definition appear in the works of Gottfried Leibniz, Immanuel Kant, Bernard Bolzano, Franz Brentano, or Kazimierz Twardowski (see, *e.g.*, [271]). Most of them treat ontology as a field of science concerning types and structures of objects, properties, events, processes, relations, and reality domains (see, *e.g.*, [106]). Therefore, ontology is neither a science concerning functioning of the world nor the ways a human being perceives it. It poses questions: *How do we classify everything?*, *What*

classes of beings are inevitable for describing and concluding on the subject of ongoing processes?, *What classes of being enable to conclude about the truth?*, *What classes of being enable to conclude about the future?* (see, *e.g.*, [106, 270]).

Ontology in Informatics. The term ontology appeared in the information technology context at the end of the sixties of the last century as a specific way of knowledge formalization, mainly in the context of database development and artificial intelligence (see, *e.g.*, [53, 272]). The growth in popularity of database systems caused avalanche increase of their capacity. The data size, multitude of tools used both for storing and introducing, or transferring data caused that databases became difficult in managing and communication with the outside world. Database schemes are determined to high extent not only by the requirements on an application or database theory but also by cultural conditions, knowledge, and the vocabulary used by designers. As the result, the same class of objects may possess different sets of attributes in various schemes termed differently. These attribute sets are identical terms but often describe completely different things. A solution to this problem are supposed to be ontologies which can be treated as tools for establishing standards of database scheme creation.

The second pillar of ontology development is artificial intelligence (AI), mainly because of the view according to which making conclusions requires knowledge resources concerning the outside world, and ontology is a way of formalizing and representing such knowledge (see, *e.g.*, [7, 273]).

It is worth noticing that, in the recent years, one of the main applications of ontologies has been their use for an intelligent search of information on the Internet (see, *e.g.*, [53] and [54] for more details).

Definition of Ontology. Philosophically as well as in information technology, there is a lack of agreement if it comes to the definition of ontology. Let us now consider three definitions of ontology, well-known from literature. Guarino states (see [53]) that in the most prevalent use of this term, an ontology refers to *an engineering artifact, constituted by a specific vocabulary used to describe a certain reality (or some part of reality), plus a set of explicit assumptions regarding the intended meaning of vocabulary words*. In this approach, an ontology describes a hierarchy of concepts related by relationships, whereas in more sophisticated cases, suitable axioms are added to express other relationships among concepts and to constrain the interpretation of those concepts.

Another well-known definition of ontology has been proposed by Gruber (see [105]). He defines an ontology as *an explicit specification of a conceptualization*. He explains that for AI systems, what *exists* is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the *universe of discourse*. This set of objects and the describable relationships among them are reflected in the representational vocabulary with which a knowledge-based program represents knowledge. Thus, according to Gruber, in the the context of AI, we can describe the ontology of a knowledge-based program by defining a set of representational terms. In such an ontology, definitions associate the names of

entities in the universe of discourse (*e.g.*, classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and the well-formed use of these terms.

Finally, we present a view of ontology recommended by *the World Wide Web Consortium* (W3C) (see [107]). W3C explains that *an ontology defines the terms used to describe and represent an area of knowledge.* Ontologies are used by people, databases, and applications that need to share domain information (a domain is just a specific subject area or area of knowledge such as medicine, tool manufacturing, real estate, automobile repair, financial management, etc.). Ontologies include computer-usable definitions of basic concepts in the domain and the relationships among them. They encode knowledge in a domain and also knowledge that spans domains. In this way, they make that knowledge reusable.

Structure of Ontology. Concept ontologies share many structured similarities, regardless of the language in which they are expressed. However, most ontologies describe individuals (objects, instances), concepts (classes), attributes (properties), and relations (see, *e.g.*, [53, 54, 105, 107]).

Individuals (objects, instances) are the basic, "ground level" components of an ontology. They may include concrete objects such as people, animals, tables, automobiles, and planets, as well as abstract individuals such as numbers and words.

Concepts (classes) are abstract groups, sets, or collections of objects. They may contain individuals or other concepts. Some examples of concepts are vehicle (the class of all vehicles), patient (the class of all patients), influenza (the class of all patients suffering from influenza), player (the class of players), team (the class of all players from some team).

Objects belonging to concepts in an ontology can be described by assigning attributes to them. Each attribute has at least a name and a value, and is used to store information that is specific to the object the attribute is attached to. For example, an object from the concept *participant* (see ontology from Fig. 5[2]) has attributes such as first name, last name, address, affiliation. If you did not define attributes for concepts, you would have either a taxonomy (if concept relationships are described) or a controlled vocabulary. These are useful, but are not considered true otologies.

There are three following types of relations between concepts from ontology: *a subsumption relation* (written as *is-a* relation), a *meronymy relation* (written as *part-of* relation), and a *domain-specific* relation.

The first type of relations is the *subsumption relation* (written as *is-a*). If a class A subsumes a class B, then any member of the class A *is-a* member of the class B. For example, the class *author* subsumes the class *participant.* It means that anything that is a member of the class *author* is a member of the class *Participant* (see ontology from Fig. 5). Where A subsumes B, A is called the *superclass*, whereas B is the *subclass.* The subsumption relation is very similar to the notion of *inheritance*, well-known from the *object-oriented programming*

[2] This example has been inspired by Jarrar (see [54]).

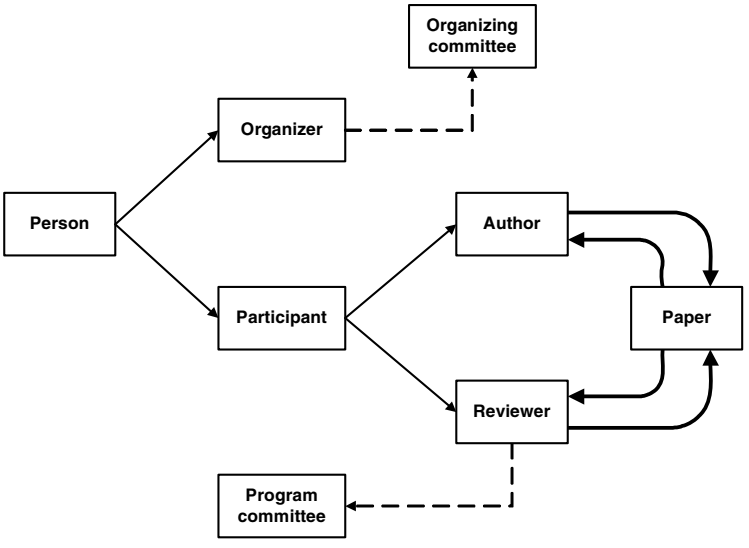


Fig. 5. The graph of a simple ontology

(see, *e.g.*, [274, 275]). Such relation can be used to create a hierarchy of concepts, typically with a maximally general concept like *person* at the top, and more specific concepts like *author* or *reviewer* at the bottom. The hierarchy of concepts is usually visualized by *a graph of ontology* (see Fig. 5) where any subsumption relation is represented by a thin solid line with an arrow in the direction from the superclass to the subclass.

Another common type of relations is the *meronymy relation* (written as *part-of*) that represents how objects combine together to form composite objects. For example, in the ontology from Fig. 5, we would say that any reviewer is-part-of the *program committee*. Any meronymy relation is represented graphically by a broken line with an arrow in the direction from the part to the composite object (see Fig. 5). From the technical point of view this type of relation between ontology terms is represented with the help of object attributes belonging to concepts. It is done in such a way that the value of an attribute of an object u, which is to be a part of some object u' belonging to different concept, informs about u'.

Apart from the standard *is-a* and *part-of* relations, ontologies often include additional types of relations that further refine the semantics modeled by the ontologies. These relations are often *domain-specific* and are used to answer particular types of questions. For example, in the domain of conferences, we might define a *written-by* relation between concepts *paper* and *author* which tells us who is the author of a paper. In the domain of conferences, we define also a *writes* relation between concepts *author* and *paper* which tells us which paper has been written by each author. Any domain-specific relation is represented by

a thick solid line with an arrow. From the technical point of view this type of relations between ontology concepts is also represented with the help of object attributes belonging to the concepts.

In this paper, we use many ontologies, constructed on the basis of domain knowledge concerning the analyzed data sets, to approximate complex concepts. In these ontologies there occur all types of relations mentioned above. However, the relations of individual types do not occur in these ontologies simultaneously but in each of them there occurs only one type of relation. The reason for this is the fact that individual relation types serve us to approximate different types of complex concepts. For example, relations of the type *is-a* occur in ontology from Fig. 6 which is an example of an ontology used to approximate spatial concepts (see Section 5). Ontologies showing dependencies between temporal concepts for structured objects and temporal concepts for constituent parts of these objects (used to approximate temporal concepts for structured objects) are examples of ontologies in which there occur relations of type *part-of* (see Section 6). On the other hand, *domain-specific* relations occur in numerous examples of behavior graphs presented in Section 6 and are used to approximate behavioral patterns. The planning graphs presented in Section 7 are also examples of ontologies in which there occur *domain-specific* relations. Incidentally, planning graphs are, in a way, ontologies even more complex than the mentioned above, because two types of concepts occur in them simultaneously. Namely, there occur concepts representing states of complex objects and concepts representing actions performed on complex objects.

Obviously, there are many ways of linking the ontologies mentioned above provided they concern the same domain. For example, an ontology describing the behavior graph of a group of vehicles may be linked with ontologies describing dependencies of temporal concepts for such groups of vehicles and temporal concepts describing behavior of individual vehicles or changes of relationships among these vehicles. Then, in such an ontology, there would occur relations of two types simultaneously, that is, *domain-specific* and *part-of* relations. Although these types of linking different ontologies are not essential for complex concept approximation methods presented in this paper, they cause a significant increase of complexity of the ontologies examined.

General Recommendations Concerning Building of an Ontology. Currently there are many papers which describe various designer groups' experience obtained in the process of ontology construction (see, *e.g.*, [276]). Although they do not constitute formal frames enabling to create an integral methodology yet, general recommendations how to create an ontology may be formed on their basis. Each project connected with an ontology creation has the following phases:

- Motivation for creating an ontology.
- Definition of the ontology range.
- Ontology building.
  - Building of a lexicon.
  - Identification of concepts.

  - Building of the concept structure.
  - Modeling relations in ontology.
- The evaluation of the ontology obtained.
- Ontology implementation.

Motivation for creating an ontology is an initial process resulting from arising inside a certain organization, a need to change the existing ontology or to create a new one. Extremely crucial for the whole further process is, at this stage, clarity of the aim for which the ontology is built. It is the moment when potential sources of knowledge needed for the ontology construction should be defined. They are usually sources which may be divided into two groups those requiring human engagement (*e.g.*, interviews, discussions) and those in which a human does not appear as a knowledge source (*e.g.*, documents, dictionaries and publications from the modeled domain, intranet and Internet, and other ontologies).

By *the ontology range* we understand this part of the real world which should be included into the model under creation in the form of concepts and relations among them. One of the easier, and at the same time very effective, ways to determine the ontology range accurately is using the so-called "competency questions" (see, *e.g.*, [277]). The starting point for this method is defining a list of questions to which the database built on the basis of the ontology under construction should give an answer.

Having the range defined, the process of *ontology building* should be started. The first step in ontology building is defining a list of expressions, phrases, and terms crucial for a given domain and a specific context of application. A *lexicon* should be composed that is a dictionary containing terms used by the ontology as well as their definitions, from the list.

The lexicon is a starting point for the most difficult stage in the ontology building, that is, construction of concepts (classes) of the ontology and relations among these concepts. It should be remembered that it is not possible to perform these two activities one after the other. They have to be performed in parallel. We should bear in mind that each relation is also a concept. Thus, finding the answer to the question *What should constitute a concept and what should constitute a relation?* is not easy and depends on the target application and, often, the designer's experience.

If it comes to building hierarchical classes, three approaches to building such a hierarchy are given in the paper [278]:

1. *Top-down.* We start with a concept superior to all concepts included in the knowledge base and we come to the next levels of inferior concepts by applying atomization.
2. *Bottom-up.* We start with the most inferior concept contained in the knowledge base and we come to the concepts on higher levels of hierarchy by applying generalization.
3. *Middle-out.* We start with concepts which are the most crucial in terms of the project and we perform atomization or generalization when needed.

In order to evaluate the obtained ontology it should be checked if the ontology possesses the following qualities ([277]):

- *Consistency.* The ontology is consistently integral, that is, contradictory conclusions cannot be drawn from it.
- *Completeness.* The ontology is complete if all expected elements are included in the model (concepts, relations, etc.).
- *Conciseness.* All information gathered in the ontology is concise and accurate.
- The possibility of answering the *"competency questions"* posed previously.

Summing up, an ontology building is a laborious process requiring a huge amount of knowledge concerning the modeling process itself, the tools used, and the domain being modeled.

Ontology Applications. Practical ontology applications relate to the so-called *general ontologies* which have a rather general character and may be applied in a knowledge base building from different domains and *domain ontologies* meaning ontologies describing knowledge about a specific domain or a specific fragment of the real world. Many such ontologies have been worked out and they are often available on the Internet. They are, *e.g.*, Dublin Core (see [279]), GFO (General Formal Ontology [280]), OpenCyc/ResearchCyc (see [281]), SUMO (Suggested Upper Merged Ontology [282]), WordNet (see [283]), DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering [284]) and others.

Generally, ontologies are applied when the semantics of the data gathered is crucial. It turns out that such a situation takes place quite often, particularly when intelligent methods of data analysis are supposed to act effectively. That is why ontologies more and more are useful in information technology projects. Some examples of applications of ontologies are e-commerce, bioinformatics, geographical, information systems, regulatory and legal information systems, digital libraries, e-learning, agent technology, database design and integration, software engineering natural language processing, information access and retrieval, the Semantic Web, Web services, medicine (see, *e.g.*, [53] and [54] for more details).

Computer Systems for Creating and Using Ontologies. There is a series of formal languages to represent ontologies. These are such languages as *Web Ontology Language* (OWL [107]), *Resource Description Framework* (RDF [285]), *Ontology Inference Layer* (OIL [286]), *DARPA Agent Markup Language* (DAML [287]), CycL (see [288]), etc. However, the most dynamically developed one is OWL which came to the existence as an improvement of the DAML, OIL and RDF languages.

There are also many computer systems for creating and using ontologies. They are, *e.g.*, Cyc (see [288]), OpenCyc (see [289]), Protege (see [290]), OntoStudio (previously OntoEdit [291]), Ontolingua (see [292]), Chimaer (see [293]), OilEd (see [294]), and others. Within these systems, the ontology is usually created using convenient graphical tools which make it possible to enter all the elements of ontology as well as their further edition and visualization.

Ontological systems very often possess mechanisms of concluding on the basis of ontology constructed. These mechanisms work in such a way that after creating an ontology the system may be asked quite complex questions. They concern

the existence of an instance of a concept which satisfies certain logical conditions, defined using concepts, attributes, and relations occurring in the ontology. For instance, in the ontology in Fig. 5, we could pose the following questions:

- *Who is the author of a given paper?*
- *Which papers have been reviewed by a given reviewer?*
- *Which persons belong to the programming committee?*

From the technical point of view, information searching based on ontology is performed with the help of questions formed in a formal language used to represent ontology or its special extension. For instance, the language RDQL (*RDF Data Query Language* [295]) is a question language similar to the language SQL extending the RDF language. Usually, the ontological systems also enable to form questions using graphical interface (see, e.g, [290, 291]).

4.2 Motivations for Approximation of Concepts and Relations from Ontology

In current systems operating on the basis of ontology it is assumed that we possess complete information about concepts, that is, for each concept all objects belonging to this concept are known by us. This assumption causes that, in order to examine the membership of an object to the concept, it is enough to check if this object occurs as an instance of this concept or not. Meantime, in practical applications we often possess only incomplete information about concepts, that is, for each concept, certain sets of objects constituting examples and counterexamples, respectively are given. It causes the necessity of approximating concepts with the help of classifiers. For instance, using the ontology in Fig. 6 which concerns safe vehicle driving on a road, it cannot be assumed that all concept instances of this ontology are available. For example, for the concept *safe driving*, it cannot be assumed that the information about all possible cars driving safely is available. That is why for such a concept, a classifier is constructed which is expected to be able to classify examples of vehicles into those belonging and those which do not belong to the concept.

Apart from that, the relations between concepts, defined in current systems based on ontology, are usually precise (exact, crisp). For example, for the relation *is-a* in ontology from Fig. 5, if the relation between concepts *author* and *participant* is to be precise (exact, crisp), then each author of a paper at a conference is a participant of this conference. In practice, however, it does not always have to be that way. It is possible that some authors of papers are not conference participants, particularly in the case of articles having many coauthors. So, a relation between concepts can be imprecise (inexact, vague). Besides, on the grounds of classical systems based on ontology, when we possess complete information about concepts, the problem of vagueness of the above relation may be solved by adding to the ontology an additional concept representing these authors who are not conference participants and binding this new concept with the concept *person* by the *is-a* relation. However, in practical applications, when the available information about concepts is not complete, we are even not able to

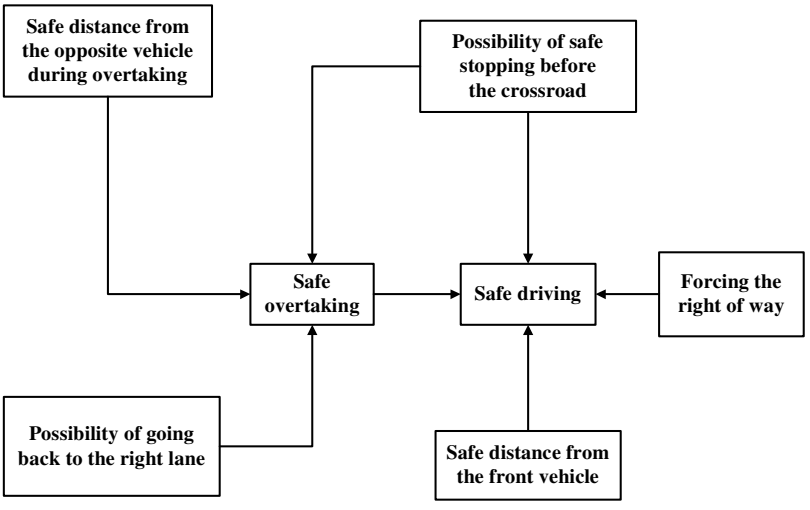


Fig. 6. An ontology for safe driving

check whether the relations under consideration are precise (exact, crisp). That is why relations among concepts also require approximation.

In approximation of concepts occurring in ontology, there often appears the following problem. In practical applications, usually is the so-called sensor data available only (that is, data obtained by measurement using sensors, thus obtained on a low level of abstraction). For example, by observing a situation on a road, i.e., such data as speed, acceleration, location, the current driving lane, may be obtained. Meanwhile, some concepts occurring in ontology are so complex that they are separated by a considerable semantical distance from the sensor data, i.e., they are defined and interpreted on very different levels of abstraction. Hence, approximation of such concepts using sensor data does not lead to classifiers of satisfying quality (see, *e.g.*, [42, 44, 45, 46, 48]). For instance, in ontology from Fig. 6, such a complex concept is without a doubt the concept *safe driving* because it is not possible to directly determine whether a given vehicle goes safely on the basis of simple sensor data only.

If, however, apart from complex concepts there are simple concepts in ontology, that is, those which may be approximated using sensor data, and they are directly or indirectly linked by relations to complex concepts, then appears a need to use the knowledge about the concepts and relations among them to approximate complex concepts more effectively. For example, in order to determine if a given vehicle drives safely, other concepts from ontology from Fig. 6, linked by relations to the concept *safe driving*, may be used. For example, one of such concepts is the *possibility of safe stopping before the crossroad*.

The aim of this paper is to present set of methods for approximating complex spatio-temporal concepts and relations among them assuming that the

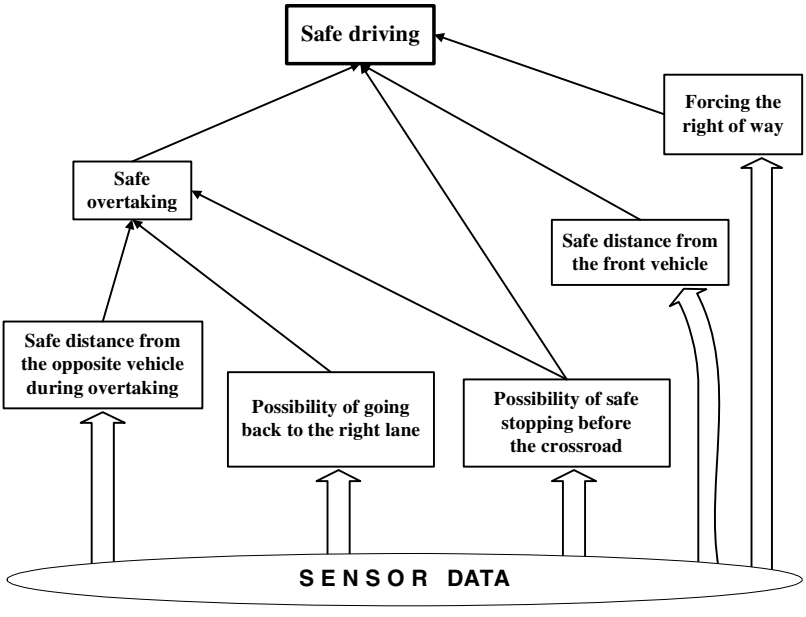


Fig. 7. The ontology for safe driving revisited

information about concepts and relations is given in the form of ontology. To meet these needs, by ontology we understand a finite set of concepts creating a hierarchy and relations among these concepts which link concepts from different levels of the hierarchy. At the same time, on top of this hierarchy there is always the most complex concept whose approximation we are interested in aiming at practical applications. For example, ontology from Fig. 6 may be presented hierarchically as in Fig. 7. At the same time, we assume that the ontology specification contains incomplete information about concepts and relations occurring in ontology, particularly for each concept, sets of objects constituting examples and counterexamples for these concepts are given. Additionally, for concepts from the lowest hierarchical level (sensor level) it is assumed that there are also *sensor attributes* available which enable to approximate these concepts on the basis of the examples and counterexamples given. This fact is marked in Fig. 7 by block arrows.

4.3 Unstructured, Structured, and Complex Objects

Every concept mentioned in this paper is understood as a subset of a certain set called the universe. Elements of the universe are called objects and they are interpreted as states, incidents, vehicles, processes, patients, illnesses and sets or sequences of entities mentioned previously. If such objects come from the real-life world, then their perception takes place by detecting their structure. Discovery

of relevant object structure for particular tasks is a complex problem strongly related to perception, that is usually understood as the process of acquiring, interpreting, selecting, and organizing sensory information (see, *e.g.*, [45, 86, 145, 146, 147, 148, 149, 150, 151, 152, 296, 297, 298, 299]). Many interdisciplinary research has been conducted in this scope in the overlapping areas of such fields as cognitive science, psychology and neuroscience, pattern recognition (see, *e.g.*, [26, 27, 35, 300, 301, 302, 303]).

Structure of objects is used to define compound patterns over objects with the simple or structured structures. The construction of such compound patterns may be hierarchical. We search for patterns relevant for approximation of some complex concepts. Notice, that together with the granularity of patterns one should consider the computational complexity of satisfiability testing for such patterns.

The structure of the perceived objects may be more or less complex, because the objects may differ in complexity. It means both the degree of spatial as well as the spatio-temporal complexity. When speaking about spatial complexity we mean not only the fact that the objects differ in the features such as location, size, shape, color, weight, but also that objects may consist of parts related with each other in terms of dependencies (*e.g.*, one may examine objects which are object groups in the traffic). However, the spatio-temporal complexity results from the fact that the perception of objects may be extended over time (*e.g.*, one may examine objects which are single vehicles observed at a single time point and objects which are also single vehicles, but they are observed over a certain period of time). Both of these aspects of object complexity may cumulate which additionally increases the diversity of appearing objects (*e.g.*, objects which are vehicle groups observed over a certain period of time are more complex than both the objects which are vehicle groups observed at a single time point and the objects which are single vehicles observed over a certain period of time).

However, in practice the perception of objects always takes place on an established level of perceiving detail. This means that depending on the needs, during perceiving objects only such details concerning their structure are taken into account that are necessary to conduct effective reasoning about the objects being perceived. For example, if we want to identify vehicles driven dangerously on the road, then we are not interested in the internal construction of each vehicle but rather the behavior of each vehicle as a certain whole. Hence, in the paper, we examine objects of two types. The first type of objects are *unstructured* objects, meaning those which may be treated as indivisible wholes. We deal with this type of objects when we analyze *patients*, *bank clients* or *vehicles*, using their parameters observed at the single time point.

The second type of objects which occurs in practical applications are *structured* objects which cannot be treated as indivisible wholes and are often registered during some period. Examples of this type of objects may be *a group of vehicles driving on a highway*, *a set of illnesses occurring in a patient*, *a robot team performing a task*.

In terms of spatiality, structured objects often consist of disjunctive parts which are objects of uniform structure connected with dependencies. However, generally, the construction of structured objects is hierarchical, that is, their parts may also be structured objects. Additionally, a great spatial complexity of structured objects causes that conducting effective reasoning about these objects usually requires their observation over a certain period of time. Thus, the hierarchy of such objects' structure may concern not only their spatial, but also spatio-temporal structure. For example, to observe simple behaviors of a single vehicle (*e.g.*, speed increase, a slight turn towards the left lane) it is sufficient to observe the vehicle over a short period of time, whereas to recognize more complex behaviors of a single vehicle (*e.g.*, acceleration, changing lanes from right to the left one), the vehicle should be observed for a longer period of time, at the same time a repeated observation of the above mentioned simple behaviors may be extremely helpful here (*e.g.*, if over a certain period the vehicle increased speed repeatedly, it means that this vehicle probably accelerates). Finally, behavior observation of a vehicle group requires its observation for an even longer period of time. It happens that way because the behavior of a vehicle group is usually the aggregation or consequence of vehicle behaviors which belong to the group (*e.g.*, observation of an overtaking maneuver of one vehicle by another requires following specific behaviors of both the overtaking and overtaken vehicle for a certain period of time).

Obviously, each of structured objects usually may be treated as an unstructured object. If we treat any object as an unstructured object at a given moment, it means that its internal structure does not interest us from the point of view of the decision problems considered. On the other hand, it is extremely difficult to find real-life unstructured objects, that is, objects without parts. In the real-life world, almost every object has some kind of internal structure and consists of certain spatial, temporal or spatio-temporal parts. Particularly, objects which are examples and counterexamples of complex concepts (both spatial and spatio-temporal), being more or less semantically distant from sensor data, have a complex structure. Therefore, one can say that they are *complex objects*. That is why the division of complex objects into unstructured and structured ones is of a symbolic character only and depends on the interpretation of these objects. If we are interested in their internal structure, then we treat them as structured objects; otherwise we treat them as unstructured ones.

4.4 Representation of Complex Object Collections

If complex objects are gathered into a collection, then in order to represent the available information about these objects, one may use information systems. Below, we present an example of such an information system whose objects are vehicles and attributes describe the parameters of the vehicle recorded at a given time point.

Example 1. Let us consider an information system $\mathbf{A} = (U, A)$ such that $A = \{x, y, l, v, t, id\}$. Each object of this system represents a condition of a considered

vehicle at one time moment. The attributes x and y provide the current location of a vehicle, the l and v attributes provide us with current traffic lane on which the vehicle is and the current vehicle speed respectively. The attribute t represents time in a number of seconds which has passed since the first observation of the vehicle (V_t is a subset of the set of positive integer numbers). The attribute id provides identifiers of vehicles. □

The second, extremely crucial, example of the information system used in this paper is an information system whose objects represent patient conditions at different time points.

Example 2. Let us consider an information system $\mathbf{A} = (U, A)$ such that $U = \{u_1, ..., u_n\}$ and $A = \{a_1, ..., a_m, a_t, a_{id}\}$. Each object of this system represents medical parameters of a certain patient during one day of his/her hospitalization. Attributes $a_1, ..., a_m$ describe medical parameters of the patient (examination results, diagnoses, treatments, medications, etc.), whereas the attribute a_t represents time in a number of days which has passed since the first observation of the patient (V_{a_t} is a subset of the set of positive integer numbers). Finally, the attribute a_{id} provides identifiers of patients. □

Similarly to the two examples above, the attributes of complex objects may be based on sensor data. However, in a general case the properties of complex objects may be defined in languages which are defined specifically for a given purpose (see Section 4.7).

4.5 Relational Structures

As we have written before, structured objects consist of parts which are structured objects of lesser complexity (hierarchical structure) or unstructured objects connected by dependencies. Additionally, a great spatial complexity of structured objects causes that conducting effective conclusions about these objects usually requires their observation for a certain period of time. Hence, there is a need to follow the spatio-temporal dependencies between parts of complex objects. Therefore, the effective description of the structure of objects requires not only providing spatial properties of individual parts of these objects, but also describing the spatio-temporal relations between the parts of these objects. Therefore, in order to describe the structure of complex objects and relations between complex objects in this paper we will use relational structures (see, *e.g.*, [5, 89]).

In order to define the relational structure using language and semantics of *first-order logic* we assume that a set of relation symbols $REL = \{R_i : i \in I\}$ and function symbols $FUN = \{f_j : j \in J\}$ are given, where I, J are some finite sets (see, *e.g.*, [89]). For any functional or relational symbol there is assigned a natural number called the *arity* of the symbol. Functional symbols and relations of arity 0 are called *constants*. The set of constants is denoted by *CONST*. Symbols of arity 1 are called *unary* and of arity 2 are called *binary*. In the case of binary relational or functional symbols we usually use traditional infix notation. For instance we write $x \leq y$ rather than $\leq (x, y)$. The set of functional

and relational symbols together with their arities is called the *signature*. The interpretation of a functional symbol f_i (a relational symbol R_i) over the set A is a function (a relation) defined over the set A and denoted by f_i^A (R_i^A). The number of arguments of a function f_i^A (a relation R_i^A) is equal the arity of f_i (R_i). Now, we can define the relational structure of a given signature (see, *e.g.*, [5, 89]).

Definition 2 (*A relational structure of a given signature*)**.** *Let $\Sigma = REL \cup FUN$ be a signature, where $REL = \{R_i : i \in I\}$ is a set of relation symbols and $FUN = \{f_j : j \in J\}$ is a set of function symbols, where I, J are some finite sets.*

1. *A relational structure of signature Σ is a triple $(D, \mathbf{R}, \mathbf{F})$ where*
   - *D is a non-empty finite set called the domain of the relational structure,*
   - *$\mathbf{R} = \{R_1^D, ..., R_k^D\}$ is a finite (possibly empty) family of relations defined over D such that R_i^D corresponds to symbol $R_i \in REL$ and $R_i^D \subseteq D^{n_i}$ where $0 < n_i \leq card(D)$ and n_i is the arity of R_i, for $i = 1, ..., k$,*
   - *$\mathbf{F} = \{f_1^D, ..., f_l^D\}$ is finite (possibly empty) family of functions such that f_j^D corresponds to symbol $f_j \in FUN$ and $f_j^D : D^{m_j} \longrightarrow D$ where $0 \leq m_j \leq card(D)$ and m_j is the arity of f_j, for $j = 1, ..., l$.*
2. *If for any $f \in \mathbf{F}$, $f : D^0 \longrightarrow D$, then we call such a function constant and we identify it with one element of the set D, corresponding to f.*
3. *If $(D, \mathbf{R}, \mathbf{F})$ is a relational structure and F is empty, then such relational structure is called pure relational structure and is denoted by $(D, \mathbf{R})$.*

A classical example of a relational structure is a set of real numbers with operations of addition and multiplications and ordering relation.

A typical example of a pure relational structure is a directed graph whose domain is set of graph nodes and the family of relations consists of one relation described by a set of graph edges.

The example below illustrates how relational structures may be used to describe the spatial structure of a complex object.

Example 3. Let us examine the complex object which is perceived as an image in Fig. 8. In this image one may notice a group of six cars: A, B, C, D, E, F. In order to define the spatial structure of this car group, the most crucial thing is defining the location of cars towards each other and defining the diversity of the driving directions of individual cars. That is why the spatial structure of such a group may be described with the help of relational structure $(\mathcal{S}, \mathbf{R})$, where:

- $\mathcal{S} = \{A, B, C, D, E, F\}$,
- $\mathbf{R} = \{R_1, R_2, R_3, R_4\}$, where:
  - $\forall (X, Y) \in \mathcal{S} \times \mathcal{S} : (X, Y) \in R_1$ iff X is driving directly before Y,
  - $\forall (X, Y) \in \mathcal{S} \times \mathcal{S} : (X, Y) \in R_2$ iff X is driving directly behind Y,
  - $\forall (X, Y) \in \mathcal{S} \times \mathcal{S} : (X, Y) \in R_3$ iff X is coming from the opposite direction in comparison with Y,
  - $\forall (X, Y) \in \mathcal{S} \times \mathcal{S} : (X, Y) \in R_4$ iff X is driving in the same direction as Y.

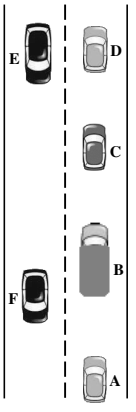


Fig. 8. An example of spatial complex object

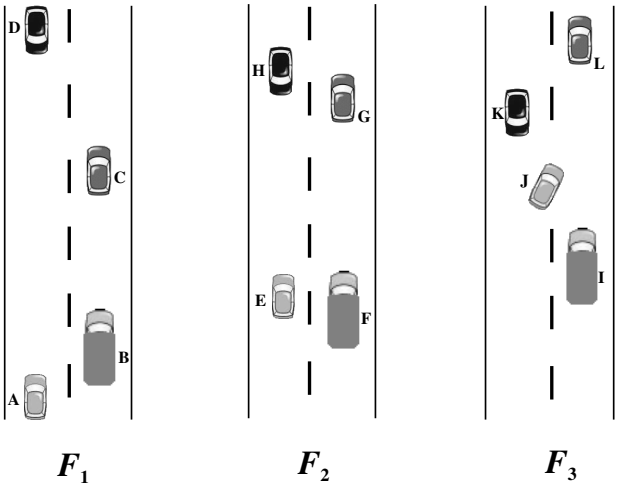


Fig. 9. An example of spatio-temporal complex object

For instance, it is easy to see that (B, A), (C, B), (D, C), $(F, E) \in R_1$, (A, B), (B, C), (C, D), $(E, F) \in R_2$, (E, C), (E, B), $(F, A) \in R_3$ and (A, C), (B, D), $(E, F) \in R_4$. □

Complex objects may also have spatio-temporal structure. The example below shows this type of a structured object.

Example 4. Let us examine the complex object which is represented with the help of three images F_1, F_2 and F_3 recorded at three consecutive time points (see Fig. 9). In image F_1 one may notice cars A, B, C and D, whereas in image

F_2 we see cars E, F, G and H. Finally, in image F_3 we see cars I, J, K and L (see Fig. 9). It is easy to notice that pictures F_1, F_2 and F_3 may be treated as three frames chosen from a certain film made *e.g.*, from an unmanned helicopter conducting a road observation, and at the same time each consecutive frame is distant in time from the previous one by about one second. Therefore, in all these pictures we see the same four cars, at the same time the first car is perceived as car A, E or J, the second car is perceived as car B, F or I, the third car is perceived as car C, G or L and the fourth car is perceived as car D, H or K. The spatial structure of complex object $\mathcal{ST} = \{A, B, C, D, E, F, G, H, I, J\}$ may be described with the help of relational structure similar to the one in Example 3. However, object $\mathcal{ST}$ has spatio-temporal structure which should be reflected in relational structure describing complex object $\mathcal{ST}$. That is why, to the relation family $\mathbf{R}$ from Example 3 we add relation R_t determined in the following way:

$$\forall (X,Y) \in \mathcal{ST} \times \mathcal{ST} : (X,Y) \in R_t \text{ iff } X \text{ represents the same vehicle as } Y \text{ and } X \text{ was recorded earlier than } Y.$$

For instance, it is easy to see that (A, E), $(H, K) \in R_t$, but (G, C), $(I, F) \notin R_t$ and (C, H), $(F, K) \notin R_t$.

Moreover, we modify the definition of the remaining relations from family $\mathbf{R}$:

- $\forall (X,Y) \in \mathcal{ST} \times \mathcal{ST} : (X,Y) \in R_1$ iff X, Y were noticed in the same frame and X is going directly before Y,
- $\forall (X,Y) \in \mathcal{ST} \times \mathcal{ST} : (X,Y) \in R_2$ iff X, Y were noticed in the same frame and X is driving directly behind Y,
- $\forall (X,Y) \in \mathcal{ST} \times \mathcal{ST} : (X,Y) \in R_3$ iff X, Y were noticed in the same frame and X is coming from the opposite direction in comparison with Y,
- $\forall (X,Y) \in \mathcal{ST} \times \mathcal{ST} : (X,Y) \in R_4$ iff X, Y were noticed in the same frame and X is driving in the same direction as Y.

□

If some set of complex objects is perceived as an unstructured object (its parts are not distinguished) and these objects belong to the object set of a certain information system, then a structure of such set of complex objects is described by relational structure, that we call *a trivial relational structure*.

Definition 3. *Let $\mathbf{A} = (U, A)$ be an information system. For any set of objects $U' \subseteq U$ we define a relational structure $(Dom, \mathbf{R}, \mathbf{F})$ such that $Dom = \{U'\}$, $\mathbf{R}$ and $\mathbf{F}$ are empty families. Such relational structure is called a trivial relational structure.*

The above trivial relational structures are used to approximate spatial concepts (see Section 5).

In each collection of complex objects there may occur relations between objects belonging to this collection. That is why each collection of complex objects may be treated as a complex object whose parts are objects belonging to the collection. Hence, the structure of complex object collection may be described using relational structure, where the domain elements of this structure are objects which belong to this collection (see Section 4.7).

4.6 Languages and Property Systems

In the paper, we use many special languages to define features of complex objects. Any language $\mathcal{L}$ is understood as a set of formulas over a given finite alphabet and is constructed in the following way.

1. First, we define an alphabet of $\mathcal{L}$, some atomic formulas and their semantics by means of some satisfiability relation $\models_{\mathcal{L}}$. The satisfiability relation is a binary relation in $X \times \mathcal{L}$, where X denotes a universe of elements (objects). We will write $x \models_{\mathcal{L}} \alpha$ to denote the fact that $\models_{\mathcal{L}}$ holds for the pair (x, α) consisting of the object x and the formula α.
2. Next, we extend, in the standard way, the satisfiability relation $\models_{\mathcal{L}}$ on Boolean combination of atomic formulas, i.e., on the least set of formulas including atomic formulas and closed with respect to the classical propositional connectives: disjunction ($\vee$), conjunction ($\wedge$), negation ($\neg$) using the following rules:
   (a) $x \models_{\mathcal{L}} (\alpha \vee \beta)$ iff $x \models_{\mathcal{L}} \alpha$ or $x \models_{\mathcal{L}} \beta$,
   (b) $x \models_{\mathcal{L}} (\alpha \wedge \beta)$ iff $x \models_{\mathcal{L}} \alpha$ and $x \models_{\mathcal{L}} \beta$,
   (c) $x \models_{\mathcal{L}} \neg(\alpha)$ iff $non(x \models_{\mathcal{L}} \alpha)$,
   where α, β are formulas, x is an object, and the symbol $\models_{\mathcal{L}}$ denotes the satisfiability relation of the defined language.
3. Finally, for any formula $\alpha \in \mathcal{L}$, the set $|\alpha|_{\mathcal{L}} = \{x \in X : x \models_{\mathcal{L}} \alpha\}$ can be constructed that is called the meaning (semantics) of α in $\mathcal{L}$.

Hence, in the sequel, in specifying languages and their semantics we will only define atomic formulas and their semantics assuming that the extension on Boolean combination is the standard one. Moreover, in definitions of alphabets over which languages are constructed we often omit listing parentheses assuming that the relevant parentheses are always included in alphabets.

Besides, in modeling complex objects we often use structures called *property systems*.

Definition 4 (*A property system*). *A property system is any triple* $\mathcal{P} = (X, \mathcal{L}, \models)$, *where* X *is a set of objects;* $\mathcal{L}$ *is a language over a given finite alphabet; and* $\models \subseteq X \times \mathcal{L}$ *is a satisfiability relation.*

We also use the following notation:

1. We write, if necessary, $X_{\mathcal{P}}$, $\mathcal{L}_{\mathcal{P}}$, $\models_{\mathcal{P}}$, instead of X, $\mathcal{L}$, and $\models$, respectively.
2. $|\alpha|_{\mathcal{P}} = \{x \in X : x \models_{\mathcal{P}} \alpha\}$ is the meaning (semantics) of α in $\mathcal{P}$.
3. By a_α for $\alpha \in \mathcal{L}_{\mathcal{P}}$ we denote a function (attribute) from $X_{\mathcal{P}}$ into $\{0, 1\}$ defined by $a_\alpha(x) = 1$ iff $x \models_{\mathcal{P}} \alpha$ for $x \in X_{\mathcal{P}}$.
4. Any property system $\mathcal{P}$ with a finite set of objects and a finite set of formulas defines an information system $\mathbf{A}_{\mathcal{P}} = (X_{\mathcal{P}}, A)$, where $A = \{a_\alpha\}_{\alpha \in \mathcal{L}}$.

It is worthwhile mentioning that the definition of any information system $\mathbf{A} = (U, A)$ constructed in hierarchical modeling should start from definition of the universe of objects of such an information system. For this purpose, we select

a language in which a set U^* of complex objects is defined, where $U \subseteq U^*$. For specifying the universe of objects of $\mathbf{A}$, we construct some property system $\mathcal{Q}$ over the universe U^* of already constructed objects. The language $\mathcal{L}_{\mathcal{Q}}$ consists of formulas which are used for specifying properties of the already constructed objects from U^*. To define the universe of objects of $\mathbf{A}$, we select a formula α from $\mathcal{L}_{\mathcal{Q}}$. Such a formula is called *type* of the constructed information system $\mathbf{A}$. Now, we assume that the object x belongs to the universe of $\mathbf{A}$ iff x satisfies (in $\mathcal{Q}$) the formula α, i.e., $x \models_{\mathcal{Q}} \alpha$, where $x \in U^*$. Observe, that the universe of objects of $\mathbf{A}$ can be an extension of the set U because U is usually only a sample of possible objects of $\mathbf{A}$. Notice that the type α selected for a constructed information system defines a binary attribute a_α for this system. Certainly, this attribute can be used to define the universe of the information system $\mathbf{A}$ (see Section 4.7 for more details). Notice also that the property system $\mathcal{Q}$ is constructed using property systems and information systems used in modeling the lower level of concept hierarchy.

4.7 Basic Languages of Defining Features of Complex Objects

As we have written before, the perception of each complex object coming from the real-life world takes place by detecting its structure (see Section 4.3), whereas the features of a given complex object may be determined only by establishing the features of this structure. The structures of complex objects which are the result of perception of complex objects may be modeled with the help of relational structures (see Section 4.5). Therefore, by the features of complex objects represented with relational structures we will understand the features of these structures.

Each collection of complex objects K may be represented using an information system $\mathbf{A} = (U, A)$, where the object set U is equal to collection K and the attributes from set A describe the properties of complex objects from collection K and more precisely, the properties of relational structures representing individual objects from this collection.

In the simplest case, the attributes from set A may be sensor attributes, that is, they represent the readings of sensors recorded for objects from set U (see Example 1 and Example 2 from Section 4.4).

However, in the case of structured objects whose properties usually cannot be described with the help of sensor attributes, the attributes from set A may be defined with the use of the properties of these objects' parts, the relations between the parts and information about the hierarchy of parts expressed *e.g.*, with the help of concept ontology (see Section 4.10).

In practice, apart from the properties of complex objects described above and represented using the attributes from set A, other properties of complex objects are also possible which describe the properties of these objects on a slightly higher level of abstraction than the attributes from set A. These properties are usually defined by experts on the basis of domain knowledge and are often represented with the help of concepts, that is, attributes which have only two values. For the table in Example 1, *e.g.*, "safe driving"could be such a concept.

By adding such an attribute to the information system, which is usually called decision attribute or decision and marking it with d, we obtain decision table (U, A, d). However, effective approximation of a decision attribute d using attributes from set A usually requires defining new attributes which are often binary attributes representing concepts. Such concepts may be defined in an established language on the basis of attributes available in set A.

In this paper, such language is called *a language for defining features of complex objects*. In the simplest case such a language may be the language of mathematical formulas in which formulas enabling calculating the specific properties of a complex object are formed. For example, if the complex object is a certain subset of a set of rational numbers with simple addition and multiplication and the order relation, then the attributes of such a complex object may be: the minimal value, the maximum value or the arithmetic average over this set.

However, in many cases in order to define attributes of complex objects special languages should be defined. In this paper, to define a specific language defining complex object properties Tarski's approach is used which requires the language's alphabet, set of language formulas and language formula semantics (see, *e.g.*, [304] and Section 4.6).

For example, in order to define concepts describing new properties of objects from a given information system a well known language called *generalized descriptor language* may be used (see, *e.g.*, [16, 165]).

Definition 5 (*A generalized descriptor language***).** *Let* $\mathbf{A} = (U, A)$ *be an information system. A generalized descriptor language of information system* $\mathbf{A}$ *(denoted by* $GDL(\mathbf{A})$ *or GDL-language, when* $\mathbf{A}$ *is fixed) is defined in the following way:*

- *the set* $AL_{GDL}(\mathbf{A}) = A \cup \bigcup_{a \in A} V_a \cup \{\neg, \vee, \wedge\}$ *is an alphabet of the language* $GDL(\mathbf{A})$,
- *expressions of the form* $(a \in V)$, *where* $a \in A$ *and* $V \subseteq V_a$ *are atomic formulas of the language* $GDL(\mathbf{A})$.

Now, we determine the semantics of the language $GDL(\mathbf{A})$. The language $GDL(\mathbf{A})$ formulas may be treated as the descriptions of object occurring in system $\mathbf{A}$.

Definition 6. *Let* $\mathbf{A} = (U, A)$ *be an information system. The satisfiability of an atomic formula* $\phi = (a \in V) \in GDL(\mathbf{A})$ *by an object* $u \in U$ *from table* $\mathbf{A}$ *(denoted by* $u \models_{GDL(\mathbf{A})} \phi$*) is defined in the following way:*

$$u \models_{GDL(\mathbf{A})} (a \in V) \text{ iff } a(u) \in V.$$

We still need to answer the question of defining the atomic formulas (expressions of the form $a \in V$) belonging to the set of formulas of the above language.

In the case of symbolic attributes, in practical applications the formulas of the form $a \in V$ are usually defined using relations: "=" or "$\neq$" (*e.g.*, $a = v_a$ or $a \neq v_a$ for some symbolic attribute a such that $v_a \in V_a$). However, if the attribute a is

a numeric one, then the correct atomic formulas may be $a < v_a$, $a \leq v_a$, $a > v_a$ or $a \geq v_a$. Atomic formulas may be also defined using intervals, for example: $a \in [v_1, v_2]$, $a \in (v_1, v_2]$, $a \in [v_1, v_2)$ or $a \in (v_1, v_2)$, where $v_1, v_2 \in V_a$.

We present a few examples of formulas of the language $GDL(\mathbf{A})$, where $\mathbf{A} = (U, A)$, $A = \{a_1, a_2, a_3\}$ and $v_1 \in V_{a_1}$, $v_2 \in V_{a_2}$ and $v_3, v_4 \in V_{a_3}$.

- $(a_1 = v_1) \wedge (a_2 \neq v_2) \wedge (a_3 \in [v_3, v_4))$,
- $(a_1 \neq v_1) \vee (a_2 = v_2)$,
- $((a_1 = v_1) \vee (a_2 = v_2)) \wedge (a_3 > v_3)$,
- $\neg((a_1 = v_1) \wedge (a_3 \leq v_3)) \vee ((a_2 \neq v_2) \wedge (a_3 \in (v_3, v_4]))$.

Another example of a language defining complex object properties may be *a neighborhood language*. In order to define the neighborhood language a dissimilarity function of pairs of objects of the information system is needed.

Definition 7. *Let $\mathbf{A} = (U, A)$ be an information system.*

1. *We call a function $DISM_{\mathbf{A}} : U \times U \longrightarrow [0, 1]$ the dissimilarity function of pairs of objects in the information system $\mathbf{A}$, if the following conditions are satisfied:*
   (a) for any pair $(u_1, u_2) \in U \times U$:
   $$DISM_{\mathbf{A}}(u_1, u_2) = 0 \Leftrightarrow \forall\ a \in A :\ a(u_1) = a(u_2),$$
   (b) for any pair $(u_1, u_2) \in U \times U$: $DISM_{\mathbf{A}}(u_1, u_2) = DISM_{\mathbf{A}}(u_2, u_1)$,
   (c) for any $u_1, u_2, u_3 \in U$:
   $$DISM_{\mathbf{A}}(u_1, u_3) \leq DISM_{\mathbf{A}}(u_1, u_2) + DISM_{\mathbf{A}}(u_2, u_3).$$
2. *For any $u_1, u_2, u_3, u_4 \in U$, if $DISM_{\mathbf{A}}(u_1, u_2) < DISM_{\mathbf{A}}(u_3, u_4)$ then we say that objects from the pair (u_3, u_4) are more different than objects from the pair (u_1, u_2), relatively to $DISM_{\mathbf{A}}$.*
3. *If any $u_1, u_2 \in U$ satisfies $DISM_{\mathbf{A}}(u_1, u_2) = 0$ then we say that objects from the pair (u_1, u_2) are not different, relatively to $DISM_{\mathbf{A}}$, i.e., they are indiscernible, relatively to $DISM_{\mathbf{A}}$.*
4. *If any $u_1, u_2 \in U$ satisfies $DISM_{\mathbf{A}}(u_1, u_2) = 1$ then we say that objects from the pair (u_1, u_2) are completely different, relatively to $DISM_{\mathbf{A}}$.*

Let us notice that the above dissimilarity function is not a *metric* (distance) but a *pseudometric*. The reason is that the first metric condition is not satisfied which in the case of the $DISM_{\mathbf{A}}$ function would state that the distance between the pair of objects is equal to 0 if and only if they are the same objects. This condition is not satisfied because of the possibility of existence of non-one-element abstraction classes of the relation $IND_{\mathbf{A}}(A)$, that is, because of the possibility of repetition of objects in the set U.

We present an example of dissimilarity function of pairs of objects of information system.

Example 5. Let $\mathbf{A} = (U, A)$ be an information system $\mathbf{A} = (U, A)$, where $A = \{a_1, ..., a_m\}$ is the set of binary attributes. We define the dissimilarity function of pairs of objects in the following way:

$$\forall (u_1, u_2) \in U \times U : DISM_{\mathbf{A}}(u_1, u_2) = \frac{card(\{a \in A : a(u_1) \neq a(u_2)\})}{card(A)}. \quad \square$$

Let us notice, that the dissimilarity function defined above is based on a widely known and introduced by Hamming measurement of dissimilarity of two sequences of the same length expressing number of places (positions) on which these two sequences differ.

Now, we can define the neighborhood language.

Definition 8 (*A neighborhood language***).** *Let $\mathbf{A} = (U, A)$ be an information system. A neighborhood language for the information system $\mathbf{A}$ (denoted by $NL(\mathbf{A})$ or NL-language, when $\mathbf{A}$ is fixed) is defined in the following way:*

- *the set $AL_{NL}(\mathbf{A}) = U \cup (0, 1] \cup \{\neg, \vee, \wedge\}$ is an alphabet of the language $NL(\mathbf{A})$,*
- *expressions of the form (u, ε), where $u \in U$ and $\varepsilon \in (0, 1]$ called as neighborhoods of objects, are atomic formulas of the language $NL(\mathbf{A})$.*

Now, we determine the semantics of language $NL(\mathbf{A})$. The language $NL(\mathbf{A})$ formulas may be treated as the descriptions of object occurring in system $\mathbf{A}$.

Definition 9. *Let $\mathbf{A} = (U, A)$ be an information system and $DISM_{\mathbf{A}}$ be a dissimilarity function of pairs of objects from the system $\mathbf{A}$. The satisfiability of an atomic formula $\phi = (u_0, \varepsilon) \in NL(\mathbf{A})$ by an object $u \in U$ from table $\mathbf{A}$ relative to dissimilarity function $DISM_{\mathbf{A}}$ (denoted by $u \models_{NL(\mathbf{A})} \phi$), is defined in the following way:*

$$u \models_{NL(\mathbf{A})} (u_0, \varepsilon) \Leftrightarrow DISM_{\mathbf{A}}(u_0, u) \leq \varepsilon.$$

Each of formula of languages GDL or NL describes a certain set of objects which satisfy this formula (see Fig. 10). According to Definitions 5 and 8 a set of such objects is included in a set of objects U. However, it is worth noticing that these formulas may be satisfied by objects from outside the set U, that is, belonging to an extension of the set U (if we assume that attribute values on such objects can be received) (see Fig. 10).

An explanation is needed if it comes to the issue of defining pairs of objects in an information system with a dissimilarity function. For information systems many such functions may be defined applying various approaches. A review of such approaches may be found in, *e.g.*, [162, 163, 164, 165, 166, 167, 168, 169, 170, 171]). However, the approaches known from literature usually do not take into account the full specification of a specific information system. That is why in a general case the dissimilarity function of a pair of objects should be defined by experts individually for each information system on the basis of domain knowledge. Such a definition may be given in the form of an arithmetical expression (see Example 5). Very often, however, experts in a given domain are not able to present such an expression

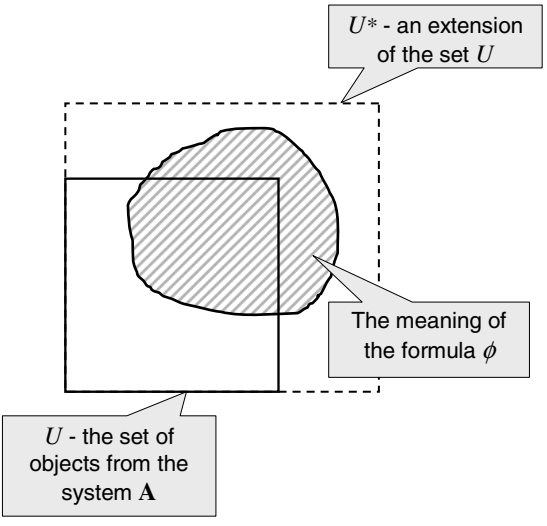


Fig. 10. The illustration of the meaning of a given formula

and content themselves with presenting a set of value examples of this function, that is, a set of pairs of objects labeled with a dissimilarity function value which exists between these objects. In this last case, defining dissimilarity function requires approximation with the help of classifiers. The classifier approximating the dissimilarity function are called dissimilarity classifier of pairs of objects for an information system.

Definition 10. *Let* $\mathbf{A} = (U, A)$ *be an information system* $\mathbf{A} = (U, A)$ *(where* $A = \{a_1, ..., a_m\}$*) and* $DISM_{\mathbf{A}}$ *is a given dissimilarity function of pairs of objects from the system* $\mathbf{A}$*.*

1. *A dissimilarity function table for the system* $\mathbf{A}$ *relatively to the dissimilarity function* $DISM_{\mathbf{A}}$ *is a decision table* $\mathbf{A}_D = (U_D, A_D, d)$*, where:*
   - $U_D \subseteq U \times U$*,*
   - $A_D = \{b_1, ..., b_m, b_{m+1}, ..., b_{2m}\}$*, where attributes from* A_D *are defined in the following way:*

$$\forall u = (u_1, u_2) \in U_D \ \forall b_i \in A_D : \ b_i(u) = \begin{cases} a_i(u_1) & i \leq m \\ a_{i-m}(u_2) & \text{otherwise.} \end{cases}$$

   - $\forall u = (u_1, u_2) \in U_D : d(u) = DISM_{\mathbf{A}}(u_1, u_2)$*.*
2. *If* $\mathbf{A}_D = (U_D, A_D, d)$ *is the dissimilarity function table for the system* $\mathbf{A}$ *then any classifier for the table* $\mathbf{A}_D$ *is called a dissimilarity classifier for the system* $\mathbf{A}$*. Such classifier is denoted by* $\mu_{DISM_{\mathbf{A}}}$*.*

Let us notice that in the dissimilarity table of the information system $\mathbf{A}$ there do not exist all the possible pairs of objects of system $\mathbf{A}$, but only a certain chosen

subset of the set of these pairs. This limitation is necessary, for the number of pairs of $U \times U$ product may be so large that the expert is not able to give all the values of decision attribute d for them. That is why in the dissimilarity table there are usually only found pairs chosen by the expert which represent typical cases of dissimilarity function determining which may be generalized with the help of a classifier.

The dissimilarity classifier may serve determining the value of dissimilarity function for the pair of objects from the information system. According to Definition 10 such pairs come from set $U \times U$, that is, they are pairs of objects from a given information system $\mathbf{A}$. However, it should be stressed that the dissimilarity classifier may determine the values of the dissimilarity function for the pairs of objects which do not belong to system $\mathbf{A}$, that is, those which belong to extension of $\mathbf{A}$. Hence, dissimilarity classifiers may be treated as a way to define concepts (new two-argument relations).

The described approach to the measure of dissimilarity is applied in this paper to the measure of dissimilarity between objects in information systems (see Section 6.7 and Section 6.19), between states in planning graphs (see Section 7.9) and plans (see Section 7.20).

4.8 Types of Complex Objects

In a given complex dynamical system there may occur many different complex objects. The collection of all such objects may be represented with the help of information system, where the set of this system's objects correspond with the objects of this collection and the attributes of this system describe the properties of complex objects from the collection and more precisely the properties of relational structures representing individual objects of this collection. Such a system for a given complex dynamical system we call in this paper a *total information system* (TIS) for a given complex dynamical system.

Attributes of the system TIS may be sensor attributes or they are defined in an established language which helps to express the properties of complex objects (see Section 4.7). To the attribute set of the system TIS one may add the binary decision attribute representing the concept describing an additional property of complex objects. The decision attribute may be further approximated with the help of attributes available from the system TIS (see Section 4.7).

However, in practice the concepts which are examined are defined only in the set of complex objects of a certain type occurring in a given complex dynamical system. In the example concerning the traffic (see Example 1) such a concept may concern only cars (*e.g.*, safe overtaking of one car by another), whereas in the example concerning patient treatment (see Example 2), the examined concepts may concern the treatment of infants only, not other people like children, adults or the elderly whose treatment differs from the treatment of infants.

Therefore, we need a mechanism which enable to appropriate selection of complex objects, and more precisely relational structures which they represent and in which we are interested at the moment. In other words, we need a method which enable to select objects of a certain type from the system TIS.

In the paper, we propose a method of adding a binary attribute to TIS to define the types of complex objects, and more precisely the types representing the objects of relational structures. The value YES of such an attribute in a given row, means that the given row represents the complex object that is of the examined type, whereas value NO means that the row represents a complex object which is not of the examined type. The attributes defining types may be defined with the help of attributes from the system TIS in the language GDL or any other language in which the attributes form the system TIS were defined.

The example below shows how the attributes defining the types of complex objects may be defined.

Example 6. Let us assume that in a certain hospital in the children's ward there was applied information system $\mathbf{A} = (U, A)$ to represent the information about patients' treatment, such that $U = \{u_1, ..., u_n\}$ and $A = \{a_1, ..., a_m, a_{age}, a_t, a_{id}\}$. Each object of this system represents medical parameters of a certain child in one day of his/her hospitalization. Attributes $a_1, ..., a_m$ describe medical parameters of the patient (examination results, diagnoses, treatments, medications, etc.), while the attribute a_{age} represents the age of patient (a number of days of life), the a_t attribute represents the value of a time unit (a number of days) which has elapsed since the first observation of the patient and the attribute a_{id} provides identifiers of patients. If system $\mathbf{A}$ is treated as the total information system for a complex dynamical system understood as a set of all patients, then the "infant" type of patient (it is a child not older than 28 days) labeled with T_{inf} may be defined with the help of formula $(a_{age} \leq 28)$. □

A slightly more difficult situation appears in the case of the information system from Example 1, when we want to define the *passenger car* type of object. A written description of the formula defining such a type may be as follows: *the object is perceived as a rectangle whose length is two to five times bigger than its width, and the movement of the object takes place in the direction parallel to the longer side of the rectangle.* It is easy to see that in order to define such a formula the information system from Example 1 would have to be complemented with sensor attributes determining the coordinates of the characteristic points of the object for determining its sizes, shape and movement direction.

If we define an additional attribute by determining the type of object in the system TIS, then we can select information subsystem in which all objects will have the same value of this attribute. Using a subsystem selected in such a way one may analyze concepts concerning the established type of objects. Obviously, during the approximation of these concepts the attribute determining the type according to which an object selection was previously performed is useless, because its value is the same for all selected objects. Therefore, the attributes defining the type of object are not used to approximate concepts, but only to an initial selection of objects for the need of concept approximation.

In a given complex dynamical system there may be observed very different complex objects. The diversity of objects may express itself both through the degree of spatial complexity and by the spatio-temporal complexity (see Section 4.3). Therefore, in a general case it should be assumed that in order to

describe the properties of all complex objects occurring in a given dynamical system, many languages must be used. For instance, to describe the properties of a single vehicle at a single time point, the information obtained directly from the sensors are usually used (*e.g.*, speed, location), to describe the properties of a vehicle observed for a certain period of time (time window), a language may be used which enables to define the so-called temporal patterns observed in time windows (see Section 6.6), whereas in order to describe the properties of groups of vehicles a language may be used which enable to define temporal patterns observed in the sequences of time windows (see Section 6.17). Moreover, it usually happens that not each of these languages is appropriate to express the properties of all complex objects occurring in a given complex dynamical system. For example, if we want to apply the language of temporal patterns to determine the properties of a vehicle at a single time point, then it is not feasible because this language requires information about the vehicle collected in the whole time window not at a single time point. Therefore, the approach to recognizing types of complex objects described above must be complemented. Namely, the attributes defining types of complex objects, apart from the values YES and NO mentioned before, may also have the *UNKNOWN* value. This value means that for a given complex object it is not possible to compute correctly the value of an attribute.

Summarizing, if we examine complex objects from a certain complex dynamical system and claim that a given complex object u is a complex object of type T, then it means that in the total information system constructed for this system there exists such attribute a_T that it takes the value YES for object u. One may also say that a given complex object u is not a complex object of type T which means that attribute a_T corresponding with type T takes the value NO for object u. The value of attribute a_T for object u may also take the value *UNKNOWN* which in practice also means that object u is not of type T.

A given complex object may be an object of many types, because there may exist many attributes identifying types in TIS which take the value YES for this object. For example, in the information system from Example 6 the type of object T_r may be defined which can be described in words as *the patient recently admitted to hospital* (that is admitted not earlier than three days ago) with the help of formula $(a_t \leq 3)$. Then, the infant admitted to hospital for treatment two days ago is a patient of both type T_{inf} and T_r.

Finally, let us notice that the above approach to determining types of objects may be applied not only to complex objects which were observed at the moment of defining the formula determining the type, but also to those complex objects which appeared later, that is, belong to the extension of the system TIS. It results from the properties of formulas of the language GDL which define the types of objects in the discussed approach.

4.9 Patterns

If an attribute of a complex object collection is a binary attribute (it describes a certain concept), then the formula enables to determine its values is usually called

a pattern for the concept. Below, we present a pattern definition assuming that there is given a language $\mathcal{L}$ defining features of complex objects of a determined type, defined using Tarski's approach (see, *e.g.*, [304]).

Definition 11 (*A pattern*)**.** *Let S be a collection of complex objects of a fixed type T. We assume $C \subseteq S$ is a concept and $\mathcal{L}$ is language of formulas defining (under a given interpretation of $\mathcal{L}$ defined by a satisfiability relation) features of complex objects from the collection S (i.e., subsets of S defined by formulas under the given interpretation).*

1. *A formula $\alpha \in \mathcal{L}$ is called a pattern for concept C explained in the language $\mathcal{L}$ if exists $s \in S$ such that $s \in C$ and $s \models_{\mathcal{L}} \alpha$ (s satisfies α in the language $\mathcal{L}$).*
2. *If $s \models_{\mathcal{L}} \alpha$ then we say that s matches pattern α or s supports pattern α. Otherwise s does not match pattern or does not support pattern α.*
3. *A pattern $\alpha \in \mathcal{L}$ is called exact relative to the concept C when for any $s \in S$, if $s \models_{\mathcal{L}} \alpha$ then $s \in C$. Otherwise, a pattern α is called inexact.*
4. *The number:*
$$support(\alpha) = card(|\alpha|_{\mathcal{L}})$$
is called the support of the pattern α.
5. *The confidence of the pattern α relatively to the concept C we denote as $confidence_C(\alpha)$ and define in the following way:*
$$confidence_C(\alpha) = \frac{card(\{s \in C : s \models_{\mathcal{L}} \alpha\})}{support(\alpha)}.$$

Thus patterns are simple but convenient way of defining complex object properties and they may be applied to information system construction representing complex object collections.

Despite the fact that according to Definition 11, patterns are supposed to describe complex object properties belonging to a given complex object collection S, they may also describe complex object properties from outside of the S collection. However, they always have to be complex objects of the same type as the objects gathered in collection S.

Patterns may be defined by experts on the basis of domain knowledge. In such a case the expert must define a needed formula in a chosen language which enables to test objects on their membership to the pattern. In a general case, patterns may be also approximated with the help of classifiers. In this case, it is required from the expert to give only examples of objects belonging to the pattern and counterexamples of objects not belonging to the pattern. Then, however, attributes which may be used to approximate the pattern are needed.

Sometimes in an information system representing a complex object collection one of the attributes is distinguished. For example, it may represent a concept distinguished by the expert which requires approximation using the rest of the attributes. Then such an information system is called a decision table (see Section 2.1).

The decision table constructed for a complex object collection may be useful in classifier construction which ensures the approximation of the distinguished decision attribute. The approximation may be performed with the help of classical classifiers (see Section 2) or stratifying classifiers (see Section 3).

As we wrote before, language formulas serving to define complex object properties may be satisfied by complex objects from outside of a given collection of complex objects. Thus, for any complex object being of the same type as complex objects from a given collection, it may be classified using the above mentioned classifier.

4.10 Approximation of Concepts from Ontology

The method of using ontology for the approximation of concepts presented in this section consists of approximating concepts from the higher level of ontology using concepts from the lower levels.

For the concepts from the lowest hierarchical level of ontology (sensor level), which are not dependent on the rest of the concepts, it is assumed that there are also available the so called *sensor attributes* which enable to approximate these concepts on the basis of applied positive and negative examples of objects.

Below, we present an example of concept approximation using sensor attributes in a certain ontology.

Example 7. Let us consider the ontology from Fig. 11. Each vehicle satisfying the established condition expressed in a natural language belongs to some concepts of this ontology. For example, to the concept of *Safe overtaking* belong vehicles which overtake safely, while to the concept of *Possibility of safe stopping before the crossroads* belong vehicles whose speed is so small that they may safely stop before the crossroads. Concepts of the lowest ontology level that is *Safe distance from the opposite vehicle during overtaking*, *Possibility of going back to the right lane*, *Possibility of safe stopping before the crossroads*, *Safe distance from the front vehicle*, *Forcing the right of way* and *Safe distance from the front vehicle* are sensor concepts, that is, they may be approximated directly using sensor data. For instance, the concept of *Possibility of safe stopping before the crossroads* may be approximated using such sensor attributes as *vehicle speed*, *vehicle acceleration*, *distance to the crossroads*, *visibility* and *road humidity*. □

On the higher levels of ontology, however, sensor attributes may not be used directly to approximate concepts because the semantical distance of approximated concepts from sensor attributes is too large and they are defined on different levels of abstraction. For example, if we wish to approximate the concept of *safe driving* on the higher level and on the sensor level we have at our disposal only attributes giving simple parameters of vehicle driving (that is, location, speed, acceleration, etc.), then it is hard to expect that these parameters allow to make the approximation of such a complex concept as *safe driving* possible. That is why in this paper we propose a method of approximating the concept from the higher level of ontology only with the help of concepts from the ontology level that is lower by one level, which are closer to the concept under approximation

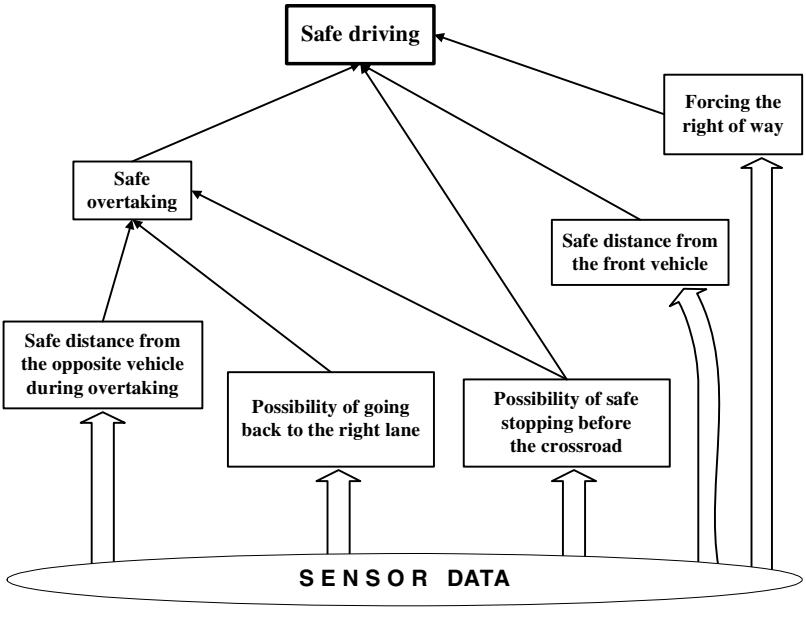


Fig. 11. An ontology as a hierarchy of concepts for approximation

than the sensor data. The proposed approach to the approximation of concepts of the lower level is based on an assumption that a concept from the higher ontology level is "not too far" semantically from concepts lying on the lower level of ontology. "Not too far" means that it can be expected that it is possible to approximate a concept from the higher level of ontology using concepts from the lower level for which classifiers have already been built.

The proposed method of approximating concepts of the higher ontology level is based on constructing a decision table for a concept on the higher ontology level whose objects represent positive and negative examples of the concept approximated on this level; and at the same time a stratifying classifier is constructed for this table. In this paper, such a table is called *a concept approximation table* of the higher ontology level concept.

One of the main problems related to construction of the concept approximation table of the higher ontology level concept is providing positive and negative examples of the approximated concept on the basis of data sets. It would seem that objects which are the positive and negative examples of the lower ontology levels concepts may be used at once (without any changes) for concept approximation on the higher ontology level. If it could be possible to perform, any ontology concepts could be approximated using positive and negative examples available from the data sets. However, in a general case, because of semantical differences between concepts and examples on different levels of ontology, objects of the lower level cannot be directly used to approximate concepts of the

higher ontology level. For example, if on a higher level of a concept hierarchy, we have a concept concerning a group of vehicles, and on a lower one concepts concerning single vehicles, then usually the properties of single vehicles (defined in order to approximate concepts of lower levels of ontology) are not sufficient to describe properties of a whole group of vehicles. Difficulties with approximation of concepts on the higher ontology level with the help of object properties from the lower ontology level also appear when on the higher ontology level there are concepts concerning another (*e.g.*, longer) period of time than concepts on the lower ontology level. For example, on the higher level we examine a concept concerning a time window (a certain time period), yet on the lower level they are concepts concerning a certain instant, i.e., a time point (see Section 6).

That is why in this paper we propose a method for constructing objects of an approximation table of the concept from the higher ontology level (that is, positive and negative examples of this concept) by arranging sets of objects which are positive and negative examples of the lower ontology level concepts. These sets must be constructed in such a way, that the properties of these sets considered together with relationships between their elements could be used for the approximation of the higher ontology level concept. However, it should be stressed here that the complex objects mentioned above (being positive and negative examples of concepts from the higher and lower ontology levels) are representation of real-life objects only. In other words, we assume that the relational structures are expressing the result of perception of real-life objects (see Section 4.5 and Fig. 12). Therefore, by the features of complex objects represented with

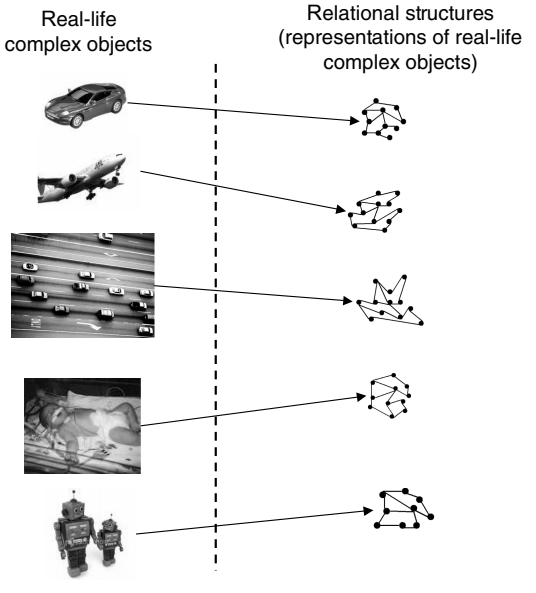


Fig. 12. Real-life complex objects and representations of their structures

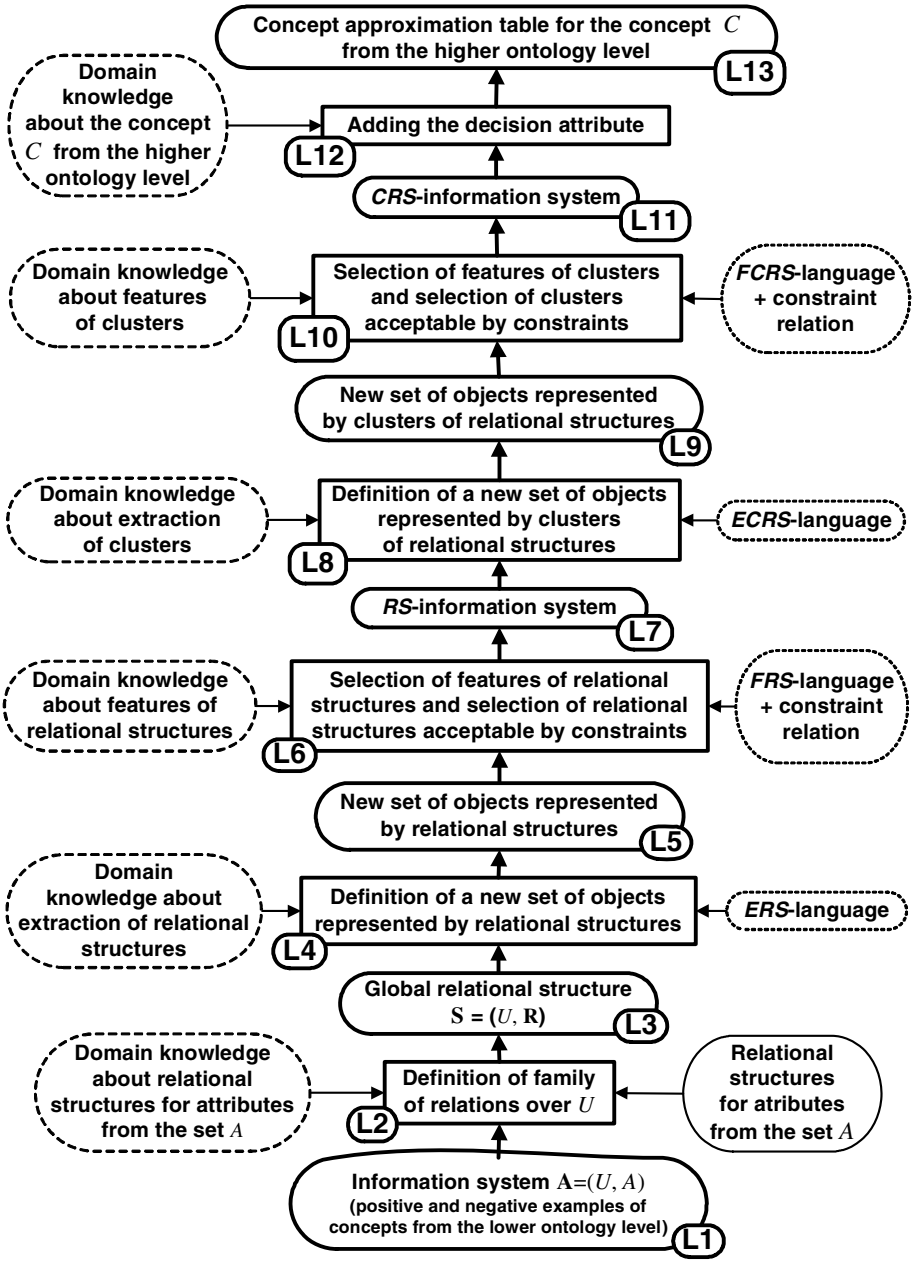


Fig. 13. The general scheme for construction of the concept approximation table

relational structures we understand the features of these structures. Such features are defined using attributes from information systems from the higher and lower ontology levels.

In Fig. 13, we illustrate the general scheme for construction of the concept approximation table for a given concept C depending in some ontology on concepts from the lower level (relatively to the concept C). In the further part of the subsection, this scheme will be explained in detail.

As we have written before, in this paper we assume that for the concepts of the lower ontology level a collection of objects which are positive and negative examples of these concepts is available. Let us also assume that they are objects of a certain information system $\mathbf{A} = (U, A)$, where attributes from set A represent all available properties of these objects (see label **L1** from the Fig. 13).

It should be stressed here that the information about the membership degree of objects from set U to the concepts from the lower ontology level may serve defining new attributes which are appended to the set A. However, providing such information for a randomly chosen object (also for an object which will appear in the future) requires previous approximation of concepts of the lower level with the help of classical or stratifying classifiers. At this point, we assume that for the concepts of the lower ontology level such classifiers were already constructed, while our aim is to approximate the concept of the higher ontology level. Incidentally, in the simplest case, the concepts of the lower ontology level may be approximated with the help of sensor attributes (see Example 7).

Apart from attributes defined on the basis of the membership of objects to the concepts or to the layers of the concepts, there may be other attributes in set A. For example, it may be an attribute identifying the recording time of values of the remaining attributes from set A for a given object from set U or an attribute unambiguously identifying individual objects or groups of objects from set U.

Objects being positive and negative examples of the lower ontology level concepts can be very often used to define new objects represented by relational structures by using available information about these objects. Relations defined in such structures may be also used to filter (extract) sets of objects or, in a more general case, sets of relational structures or their clusters as new objects for a higher level concept.

Relations among objects may be defined on the basis of attributes from the information system $\mathbf{A}$, with the use of relational structures defined on the value sets of attributes from set A (see label **L2** from the Fig. 13). For example, the value set of attribute V_{a_t} from Example 2 is a subset of the set of integer numbers. Therefore, it is a domain of a relational structure $(V_{a_t}, \{R_{a_t}\})$, where relation R_{a_t} is defined in the following way:

$$\forall (t_1, t_2) \in V_{a_t} \times V_{a_t} : t_1 R_{a_t} t_2 \Leftrightarrow t_1 \leq t_2.$$

Relation R_{a_t} may be in a natural way, generalized to the relation $R_t \subseteq U \times U$ in the following way:

$$\forall (u_1, u_2) \in U \times U : \; u_1 R_t u_2 \Leftrightarrow a_t(u_1) \; R_{a_t} \; a_t(u_2).$$

Let us notice that relation R_t orders in time the objects of the information system from Example 2. Moreover, it is also worthwhile mentioning that for any

pair of objects $(u_1, u_2) \in U^\star \times U^\star$ (where $U \subseteq U^\star$) the relation R_t is also defined (if we assume that attribute values on such objects can be received) (see Fig. 10).

Analogously, a relation ordering objects in time on the basis of attribute t from the information system from Example 1 may be obtained.

Obviously, relations defined on the basis of the attributes of information system $\mathbf{A}$ are not always related to the ordering objects in time. The example below illustrates how structural relations may be defined on the basis of the distance between objects.

Example 8. Let us consider an information system $\mathbf{A} = (U, A)$, whose object set $U = \{u_1, ..., u_n\}$ is a finite set of vehicles going from a town T_1 to a town T_2, whereas two attributes d and v belong to the attribute set A. The attribute d represents the distance of a given vehicle from the town T_2 while attribute v represents the speed of a given vehicle. Value sets of these attributes are subsets of the set of real numbers. Besides, the set V_d is a domain of relational structure $(V_d, \{R_d^\varepsilon\})$, where the relation R_d^ε is defined in the following way:

$$\forall (v_1, v_2) \in V_d \times V_d : \ v_1 \ R_d^\varepsilon \ v_2 \Leftrightarrow |v_1 - v_2| \leq \varepsilon,$$

where ε is a fixed real number greater than 0.

Relation R_d^ε may be in a natural way, generalized to the relation $R_\varepsilon \subseteq U \times U$ in the following way:

$$\forall (u_1, u_2) \in U \times U : \ u_1 R_\varepsilon u_2 \Leftrightarrow d(u_1) \ R_d^\varepsilon \ d(u_2).$$

As we see, a pair of vehicles belongs to relation R_ε when objects are distant from each other by no more than ε. Therefore, relation R_ε we call the *nearness relation* of vehicles and parameter ε is called the *nearness parameter* of vehicles. Relation R_ε may be defined for different values ε. That is why in a general case the number of nearness relations is infinite. However, if it is assumed that parameter ε takes the values from a finite set (*e.g.*, $\varepsilon = 1, 2, ..., 100$), then the number of nearness relations is finite. If R_ε is a nearness relation defined in the set $U \times U$ (where $\varepsilon > 0$), then set of vehicles U is a domain of the pure relational structure $\mathbf{S} = (U, \{R_\varepsilon\})$. The exemplary concepts characterizing the properties of individual vehicles may be *high* (*average*, *low*) speed of the vehicle or *high* (*average*, *low*) distance from the town T_2. These concepts are defined by an expert and may be approximated on the basis of sensor attributes d and v. However, more complex concepts may be defined which cannot be approximated with the help of these attributes. The example of such a concept is *vehicle driving in a traffic jam.* The traffic jam is defined by a number of vehicles blocking one another until they can scarcely move (see, *e.g.*, [305]). It is easy to notice that on the basis of observation of the vehicle's membership to the above mentioned sensor concepts (concerning a single vehicle) and even observation of the value of sensor attributes for a given vehicle, it is not possible to recognize whether the vehicle is driving in a traffic jam or not. It is necessary to examine the neighborhood of a given vehicle and more precisely to check whether there are other vehicles right after and before the examined one. Therefore, to approximate the concept *vehicle driving*

in traffic jam we need a certain type of vehicle grouping which may be performed with the help of the above mentioned relation R_ε (see Example 9). Let us add that in recognition of the vehicle's membership to the concept *vehicle driving in a traffic jam*, it is also important that the speed of the examined vehicle and the speed of the vehicles in its neighborhood are available. However, to simplify the examples, in this subsection we assume that in recognition of the vehicle's membership to the concept *vehicle driving in a traffic jam* it is sufficient to check the appearance of other vehicles in the neighborhood of a given vehicle and considering the speed of these vehicles is not necessary. □

Thus, for a given information system $\mathbf{A} = (U, A)$ representing positive and negative examples of the lower ontology levels concepts there may be defined a pure relational structure $\mathbf{S} = (U, \mathbf{R})$ (see label **L3** from the Fig. 13). Next, using the relations from family $\mathbf{R}$ a special language may be defined in which patterns are defined which describe sets of objects (new concepts) for the needs of approximation of the higher ontology level concepts (see label **L4** from the Fig. 13). The extracted sets of objects of a lower level are also usually nontrivial relational structures, for the relations determined on the whole set of objects of the lower ontology level in a natural way are defined on the extracted sets. Time windows (see Section 6.4) or sequences of time windows (see Section 6.15) may be such kind of relational structures. In modeling, we use pure relational structures (without functions) over set of objects extracted from the initial relational structures whose domains are sets of objects of lower ontology level. The reason is that these structures are defined by extension of relations structures defined on information about objects of lower ontology level and even if in the latter structures are defined functions then after the extension we obtain relations over objects rather than functions.

Example 9. Let us consider an information system $\mathbf{A} = (U, A)$ from Example 8. Let R_ε be the nearness relation defined in the set $U \times U$ for the fixed $\varepsilon > 0$. Then, the vehicle set U is the domain of relational structure $\mathbf{S} = (U, \{R_\varepsilon\})$ and the relation R_ε may be used to extract relational structures from the structure $\mathbf{S}$. In order to do this we define the family of subsets $F(\mathbf{S})$ of the set U in the following way: $F(\mathbf{S}) = \{N_\varepsilon(u_1), ..., N_\varepsilon(u_n)\}$, where:

$$N_\varepsilon(u_i) = \{u \in U : \ u_i R_\varepsilon u\}, \text{ for } i = 1, ..., n.$$

Let us notice that each set from family $F(\mathbf{S})$ is connected with one of the vehicles from set U. Therefore, each of the sets from family $F(\mathbf{S})$ should be interpreted as a set of vehicles which are distant from the established vehicle u no more than by the established nearness parameter ε. In other words each such set is a vehicle set which are in the neighborhood of a given vehicle, with the established radius of the neighborhood area. For instance, if $\varepsilon = 20$ meters then vehicles u_3, u_4, u_5, u_6, and u_7 belong to the neighborhood of vehicle u_5 (see Fig. 14). Finally, let us notice that each set $N \in F(\mathbf{S})$ is a domain of relational structure $(N, \{R_\varepsilon\})$. Thus, we obtain the family of relational structures extracted from structure $\mathbf{S}$.
□

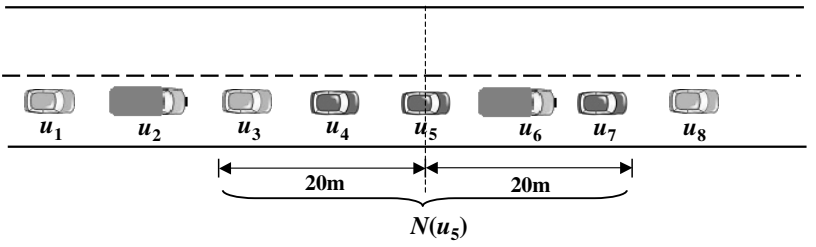


Fig. 14. A vehicle and its neighborhood

The language in which, using the relational structures, we define formulas for expressing extracted relational structures, is called *a language for extracting relational structures* (ERS-language). The formulas of ERS-language determine type of relational structures, i.e., relational structures which can appear in the constructed information system. These new relational structures represent structure of more compound objects composed out of less compound ones. We call them *extracted relational structures* (see label **L5** from the Fig. 13). In this paper, we use the three following ERS-languages:

1. the language assigned to extract trivial relational structures such as presented in Definition 3 and this method of relational structure extraction is used in the case of construction of the concept approximation table using stratifying classifiers (see Section 5.2),
2. the ETW-language assigned to extract relational structures which are time windows (see Section 6.4),
3. the $ESTW$-language assigned to extract relational structures which are sequences of time windows (see Section 6.15).

However, the above mentioned process of extracting relational structures is carried out in order to approximate the concept of the higher ontology level with the help of lower ontology level concepts. Therefore, to extract relational structures it is necessary to use information about membership of objects of the lower level to the concepts from this level. Such information may be available for any tested object thanks to the application of previously created classifiers for the lower ontology level concepts (see Section 6.4 and Section 6.15).

For relational structures extracted using ERS-language features (properties, attributes) may be defined using a specially constructed language, that we call *a language for definnig features of relational structures* (see label **L6** from the Fig. 13). The FRS-language leads to an information system whose objects are extracted relational structures and the attributes are the features of these structures. Such system will be called *an information system of extracted relational structures* (RS-information system) (see label **L7** from the Fig. 13). However, from the point of view of domain knowledge, not all objects (relational structures) extracted using ERS-language are appropriate to approximation of a given concept of the higher level of ontology. For instance, if we approximate the

concept of *safe overtaking*, it is reasonable to use objects representing vehicles examples that are in the process of overtaking maneuver, for using objects representing vehicles which are not in the process of an overtaking maneuver, nothing help to recognize the pairs of vehicles which take part in a safe overtaking with the pairs of vehicles which overtake unsafely.

For the above reason, that is, to eliminate objects which are unreal or are unreasonable, there are defined the so-called *constraints* which are formulas defined on the basis of object features used to create attributes from the *RS*-system. The constraints determine which objects may be used in order to obtain a concept example from the higher level and which cannot be used (see label **L6** from the Fig. 13). In this paper constraints are represented by a constraint relation and are defined as a formula of the language GDL (see Definition 5) on the basis of attributes appearing in the system *RS*-system.

The example below illustrates how *RS*-information systems may be defined.

Example 10. Let us consider an information system $\mathbf{A} = (U, A)$, a relational structure $\mathbf{S} = (U, \{R_\varepsilon\})$ and a family $F(\mathbf{S})$ extracted from relational structure $\mathbf{S}$ (see Example 9). We construct an information system $\overline{\mathbf{F}} = (F(\mathbf{S}), \overline{A})$ such that $\overline{A} = \{\overline{a}_f, \overline{a}_b\}$, where for any $\overline{u} = N_\varepsilon(u) \in F(\mathbf{S})$ a value $\overline{a}_f(\overline{u})$ is the number of vehicles in the neighborhood $N_\varepsilon(u)$ going in the right lane before vehicle u and $\overline{a}_b(\overline{u})$ is the number of vehicles in the neighborhood $N_\varepsilon(u)$ going in the right lane behind vehicle u. Let us notice that attributes of set $\overline{A}$ were chosen in such a way that the objects from information system $\overline{\mathbf{F}}$ are relevant to approximate the concept *vehicle driving in a traffic jam*. For example, if $\varepsilon = 20$ meters and for the object $\overline{u} \in F(\mathbf{S})$ values $\overline{a}_f(\overline{u}) = 2$ and $\overline{a}_b(\overline{u}) = 2$, then vehicle u is driving in a traffic jam (see vehicle u_4 from Fig. 15). Whereas, if $\overline{a}_f(\overline{u}) = 0$ and $\overline{a}_b(\overline{u}) = 0$, then vehicle u is not driving in a traffic jam (see vehicle u_7 from Fig. 15). For the system $\overline{\mathbf{F}}$ we define the following formula:

$$\phi \;=\; ((\overline{a}_f > 0) \;\vee\; (\overline{a}_b > 0)) \in GDL(\overline{\mathbf{F}}).$$

It is easy to notice that formula ϕ is not satisfied only by neighborhoods related to vehicles which definitely not driving in a traffic jam. Therefore, in terms of neighborhood classification to the concept *driving in a traffic jam* these neighborhoods may be called trivial ones. Hence, formula ϕ may be treated as a constraint formula which is used to eliminate the above mentioned trivial neighborhoods from $\overline{\mathbf{F}}$. After such reduction we obtain an *RS*-information system $\overline{\mathbf{A}} = (\overline{U}, \overline{A})$, where

$$\overline{U} = \{\overline{u} \in F(\mathbf{S}) :\; \overline{u} \models_{GDL(\overline{\mathbf{F}})} \phi\}. \qquad \square$$

Let us notice that the definition of attributes of extracted relational structures leads to granulation of relational structures. For example, we obtain granules of relational structures defined by the indiscernibility relation defined by new attributes.

A question arises, how to construct languages defining features of relational structures, particularly when it comes to approximation of spatio-temporal concepts, that is, those whose recognition requires following the changes of complex objects over time. One of more developed languages of this type is a temporal

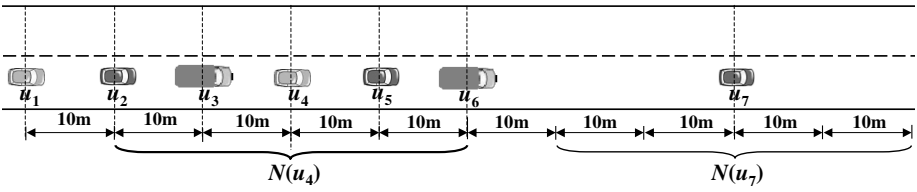


Fig. 15. Two vehicle neighborhoods

logic language. In literature there are many systems of temporal logics defined which offer many useful mechanisms (see, *e.g.*, [183, 184, 185]). Therefore, in this paper, we use temporal logics to define our own languages describing features of relational structures. Especially interesting for us are the elements appearing in definitions of *temporal logics of linear time* (*e.g.*, *Linear Temporal Logic*) and *branching time logic* (*e.g.*, *Branching Temporal Logic*).

Temporal logic of linear time assumes that time has a linear nature, that is, one without branches. In other words, it describes only one world in which each two events are sequentially ordered. In linear time logics there are the following four temporal operators introduced: $\Box$, $\Diamond$, $\bigcirc$ and $\mathcal{U}$. Generally speaking, these operators enable us to determine the satisfiability of temporal formulas in a certain time period. Operator $\Box$ (often also marked as G) determines the satisfiability of a formula at all instants (states) of the time period under observation. Operator $\Diamond$ (often marked as F) determines the satisfiability of a formula at least at one instant (state) of the time period under observation. Operator $\bigcirc$ (often marked as X) determines the satisfiability of a formula at an instant (state) right after the instant of reference. Finally, operator $\mathcal{U}$ (often marked as U) determines the satisfiability of a formula until another formula is satisfied. Therefore, linear time temporal logics may be used to express object properties which aggregate behavior of complex objects observed over a certain period of linear time, *e.g.*, features of time windows or features of temporal paths in behavioral graphs (see Section 6.6 and Section 6.17).

Temporal logic of branching time, however, assumes that time has a branching nature, that is, at a given instant it may branch itself into parallel worlds representing possible various future states. In branching time logics there are two additional path operators A and E introduced. They enable us to determine the satisfiability of temporal formulas for various variants of the future. The first operator means that the temporal formula, before which the operator occurs, is satisfied for all variants of the future. The second, however, means the formula is satisfied for a certain future. Path operators combined with the three G, F and X temporal logics operators give six possible combinations: AG, AF, AX, EG, EF and EX. These combinations give opportunities to describe multi-variant, extended over time behaviors. Therefore, temporal logics of branching time may be used to express such complex object properties that aggregate multi-variant behaviors of objects changing over time (*e.g.*, features of clusters of time windows

or features of clusters of temporal paths in behavioral graphs) (see Section 6.8 and Section 6.19).

We assume, that in extracted relational structures the time flow has a linear character. Therefore, languages using elements of temporal logics with linear time are applied to define their features. In this paper, we use the three following languages defining features of extracted relational structure:

1. the language assigned to define features of trivial relational structure such as in Definition 3 - this method of defining features of relational structures is applied together with extraction of trivial relational structure (see Definition 3) and is based on the usage of features of objects taken from information system as features of relational structures after extraction (objects in a given information system and elements of domains of extracted from this system relational structures are the same) (see Section 5.2),
2. the language FTW using elements of temporal logic language and is assigned to define relational structure properties, which are time windows (see Section 6.6),
3. the language FTP also using elements of temporal logic language, assigned to define relational structure properties, which are paths in behavioral graphs (see Section 6.17).

However, objects of RS-information systems are often not suitable to use their properties for approximating concepts of the higher ontology level. It happens this way because the number of these objects is too large and their descriptions are too detailed. Hence, if they are applied to approximate the concept from the higher ontology level, the coverage of the constructed classifier would be too little, that is, the classifier could classify too small number of tested objects. Apart from that, there would appear a problem of computational complexity which means that due to the large number of objects of such information system, the number of objects in the concept approximation table for the structured objects (see further part of this subsection) would be too large in order to construct a classifier effectively.

That is why, a clustering such objects is applied leading to obtaining a family of object clusters (see label **L8** from the Fig. 13).

The example below illustrates in a very simple way how it is possible to define clusters of relational structures.

Example 11. Let $\overline{\mathbf{A}} = (\overline{U}, \overline{A})$ be an RS-information system from Example 10. We are going to define clusters of the vehicles' neighborhoods. For this purpose we propose a relation $\overline{R}_\sigma \subseteq \overline{U} \times \overline{U}$, that is defined in the following way:

$$\forall_{(\overline{u}_1,\overline{u}_2)\in\overline{U}\times\overline{U}}\ \overline{u}_1\overline{R}_\sigma\overline{u}_2 \ \Leftrightarrow\ |\overline{a}_f(\overline{u}_1) - \overline{a}_f(\overline{u}_2)| \leq \sigma \ \wedge\ |\overline{a}_b(\overline{u}_1) - \overline{a}_b(\overline{u}_2)| \leq \sigma,$$

where σ is a fixed integer number greater than 0. As we see, to relation $\overline{R}_\sigma$ belong such pairs of vehicle neighborhoods which differ only slightly (no more than by σ) in terms of attribute values $\overline{a}_f$ and $\overline{a}_b$. Therefore, relation $\overline{R}_\sigma$ is called the *nearness relation of vehicle neighborhoods* and parameter σ is called the *nearness*

parameter of vehicle neighborhoods. The relation $\overline{R}_\sigma$ may be defined for different values σ. That is why in a general case the number of such nearness relations is infinite. However, if it is assumed that parameter σ takes the values from a finite set (*e.g.*, $\sigma = 1, 2, ..., 10$), then the number of nearness relations is finite. Let $\overline{R}_\sigma$ be nearness relation of neighborhoods determined for the established $\sigma > 0$. Then the set of neighborhood of vehicles $\overline{U}$ is the domain of a pure relational structure $\overline{\mathbf{S}} = (\overline{U}, \{\overline{R}_\sigma\})$. The relational structure $\overline{\mathbf{S}}$ is the starting point to extract clusters of vehicle neighborhoods. In order to do this we define the family of subsets $F(\overline{\mathbf{S}})$ of the set $\overline{U}$ in the following way: $F(\overline{\mathbf{S}}) = \{N_\sigma(\overline{u}_1), ..., N_\sigma(\overline{u}_n)\}$,
where:

$$N_\sigma(\overline{u}_i) = \{\overline{u} \in \overline{U} : \ \overline{u}_i \overline{R}_\sigma \overline{u}\}, \ \text{for } i = 1, ..., n.$$

Let us notice that each of the set from family $F(\overline{\mathbf{S}})$ is connected with one vehicle neighborhood from the set $\overline{U}$. For any $\overline{u} \in \overline{U}$ the set $N_\sigma(\overline{u})$ will be also denoted by $\overline{\overline{u}}$, for short. Moreover, these sets are interpreted as neighborhood clusters which are distant from the central neighborhood in the cluster no more than the established nearness parameter. In other words, each such family is a vehicles' neighborhood cluster which are close to a given neighborhood, with their established nearness parameter. For instance, if $\varepsilon = 20$ meters and $\sigma = 1$, then neighborhoods $N_\varepsilon(u_3)$, $N_\varepsilon(u_5)$ and obviously neighborhood $N_\varepsilon(u_4)$ belong to the neighborhood cluster $N_\sigma(\overline{u_4})$ (see Fig. 16), whereas the neighborhood $N_\varepsilon(u_7)$ does not belong to this neighborhood cluster. Finally, let us notice that each set $\overline{X} \in F(\overline{\mathbf{S}})$ is a domain of relational structure $(\overline{X}, \{\overline{R}_\sigma\})$. Hence, we obtain the family of relational structures extracted from structure $\overline{\mathbf{S}}$. □

Grouping of objects in system *RS*-system may be performed using chosen by an expert language of extraction of clusters of relational structures, which in this case is called *a language for extracting clusters of relational structures* (*ECRS*-language). The formulas of *ECRS*-language express families of clusters of relational structures from the input *RS*-information systems (see label **L9** from the Fig. 13). Such formulas can be treated as a type of clusters of relational structures which will create objects in a new information system. In *ECRS*-language we may define a family of patterns corresponding to a family of expected clusters. In this paper, the two following *ECRS*-languages are used:

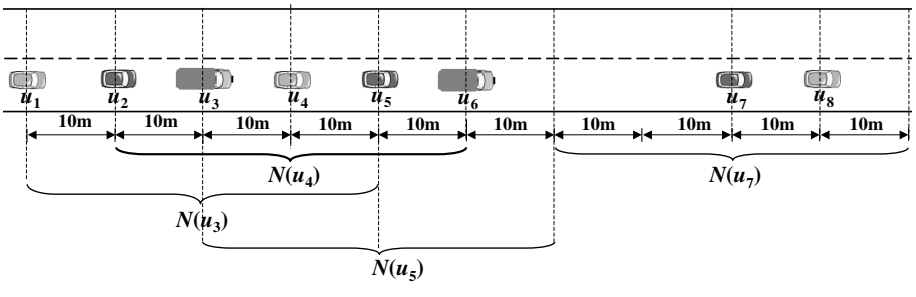


Fig. 16. Four vehicle neighborhoods

1. the language $ECTW$ assigned to define relational structure clusters which are time window families (see Section 6.8),
2. the language $ECTP$ assigned to define relational structure clusters which are path families in complex object behavioral graphs (see Section 6.19).

For clusters of relational structures extracted in such a way features may be defined using a specially constructed language, that we call *a language for defining features of clusters of relational structures* ($FCRS$-language) (see label **L10** from the Fig. 13). A formula from this language is satisfied (or unsatisfied) on a given clusters of relational structures if and only if it is satisfied for all relational structures from this clusters. The $FCRS$-language leads to an information system whose objects are extracted clusters of relational structures and the attributes are the features of these clusters (see label **L11** from the Fig. 13). Such information system we call *an information system of clusters of relational structures* (CRS-information system).

Similarly to the case of the relational structures extracted using ERS-language, not all objects (relational structures) extracted using $ECRS$-language are appropriate to approximation of a given concept of the higher level of ontology. Therefore in this case we also define constraints which are formulas defined on the basis of object features used to create attributes from the CRS-information system. Such constraints determine which objects may be used in order to obtain a concept example from the higher level and which cannot be used.

The example below illustrates how CRS-information systems may be defined.

Example 12. Let $F(\overline{\mathbf{S}})$ be the family extracted from relational structure $\overline{\mathbf{S}}$ (see Example 11). One can construct an information system $\overline{\overline{\mathbf{F}}} = (F(\overline{\mathbf{S}}), \overline{\overline{A}})$, where $\overline{\overline{A}} = \{\overline{\overline{a}}_f, \overline{\overline{a}}_b\}$ and for any $\overline{\overline{u}} \in F(\overline{\mathbf{S}})$ values of attributes $\overline{\overline{a}}_f$ and $\overline{\overline{a}}_b$ are computed as the arithmetical average of values of attributes $\overline{a}_f$ and $\overline{a}_b$ for neighborhoods belonging to the cluster represented by $\overline{\overline{u}}$. The attributes of set $\overline{\overline{A}}$ were chosen in such a way that the objects from set $\overline{\overline{U}}$ are appropriate for approximation of the concept *vehicle driving in a traffic jam*. For example, if $\varepsilon = 20$ meters, $\sigma = 1$ and values $\overline{\overline{a}}_f(\overline{\overline{u}})$ and $\overline{\overline{a}}_b(\overline{\overline{u}})$ are close to 2 then the neighborhoods from cluster represented by object $\overline{\overline{u}}$ contain vehicles which definitely drive in a traffic jam. Whereas, if $\overline{\overline{a}}_f(\overline{\overline{u}})$ and $\overline{\overline{a}}_b(\overline{\overline{u}})$ are close to 0 then the neighborhoods from cluster represented by object $\overline{\overline{u}}$ contain vehicles which definitely do not drive in a traffic jam. For the system $\overline{\overline{\mathbf{F}}}$ we define the following formula:

$$\Phi \;=\; ((\overline{\overline{a}}_f > 0.5) \;\vee\; (\overline{\overline{a}}_b > 0.5)) \in GDL(\overline{\overline{\mathbf{F}}}).$$

It is easy to notice that formula Φ is not satisfied only by such clusters to which belong vehicle neighborhoods definitely not driving in a traffic jam. Therefore, in terms of cluster classification to the concept *driving in a traffic jam* these clusters may be called trivial ones. Hence, formula Φ may be treated as a constraint formula which is used to eliminate the above mentioned trivial clusters from $\overline{\overline{\mathbf{F}}}$.

After such reduction we obtain an CRS-information system $\overline{\overline{\mathbf{A}}} = (\overline{\overline{U}}, \overline{\overline{A}})$, where

$$\overline{\overline{U}} = \{\overline{\overline{u}} \in F(\overline{\mathbf{S}}) : \ \overline{\overline{u}} \models_{GDL(\overline{\overline{\mathbf{F}}})} \Phi \ \}. \qquad \square$$

Unlike the single relational structures in relational structure clusters the time flow has a branching character because in various elements of a given cluster we observe various variants of dynamically changing reality. Therefore, to define relational structure cluster properties we use elements of temporal logics of branching time language. In this paper, we use the two following languages defining cluster properties:

1. the language $FCTW$ using elements of temporal logics language and assigned to define cluster features which are families of time windows (see Section 6.8),
2. the language $FCTP$ also using elements of temporal logics language assigned to define cluster families which are families of temporal paths in behavioral graphs, that is, sub-graphs of behavioral graphs (see Section 6.19).

Finally, we assume that to each object, acceptable by constraints, an expert adds a decision value determining whether a given object belongs to a higher level approximated concept or not (see label **L12** from the Fig. 13). After adding the decision attribute we obtain the concept approximation table for a concept from the higher ontology level (see label **L13** from the Fig. 13).

The notion of concept approximation table concerning a concept from the higher ontology level for an unstructured complex object may be generalized in the case of concept approximation for structured objects (that is, consisting of parts).

Let us assume that the concept is defined for structured objects of type T which consist of parts being complex objects of types T_1,...,T_k. In Fig. 17 we illustrate the general scheme for construction of the concept approximation table for such structured objects. We see that in order to construct a table for approximating a concept defined for structured objects of type T, CRS-systems are constructed for all types of structured object parts, that is, types T_1,...,T_k (see labels **L3−1**,..., **L3−k** from the Fig. 17). Next, these systems are joined in order to obtain a table of approximating concept of the higher ontology level determined for structured objects. Objects of this table are obtained by arranging (linking) all possible objects of linked information systems (see label **L4** from the Fig. 17). From the mathematical point of view such an arrangement is a Cartesian product of sets of objects of linked information systems. However, from the point of view of domain knowledge not all objects links belonging to such a Cartesian product are possible and reasonable (see [78, 84, 186, 187]). For instance, if we approximate the concept of *overtaking*, it is reasonable to arrange objects of such pairs of vehicles that drive close to each other. For the above reason, there are defined *constraints* which are formulas defined on the basis of properties of arranged objects. The constraints determine which objects may be arranged in order to obtain a concept example from the higher level and which

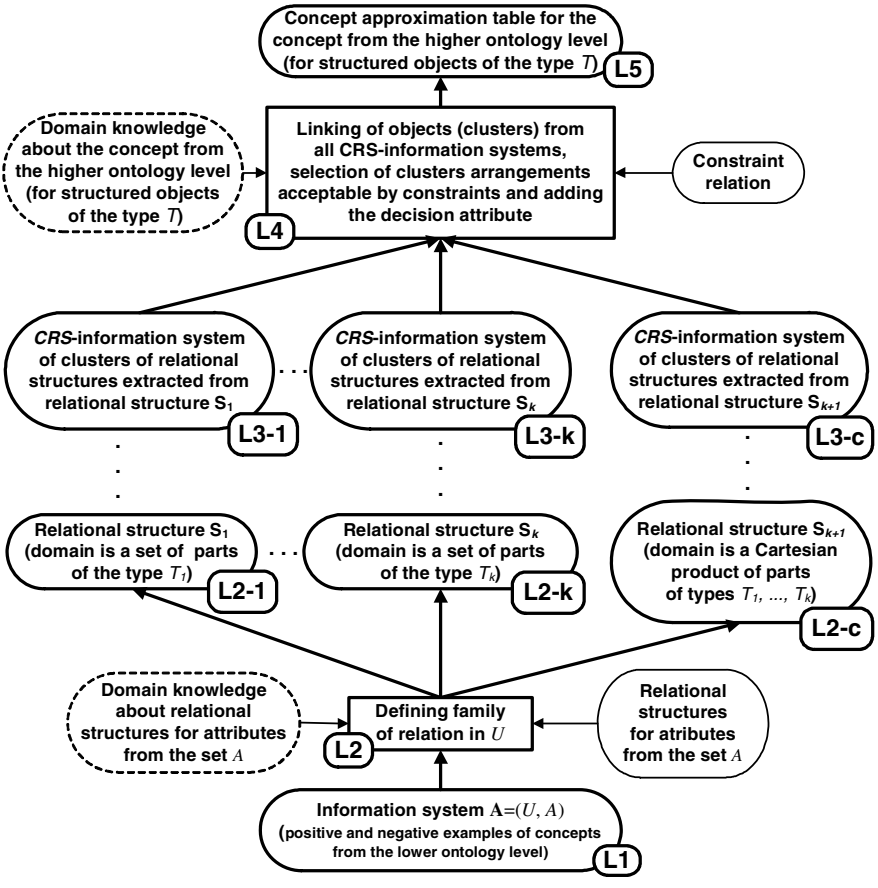


Fig. 17. The general scheme for construction of the concept approximation table for structured objects

cannot be arranged. Additionally, we assume that to each object arrangement, acceptable by constraints, an expert adds a decision value determining whether a given arrangement belongs to a higher level approximated concept or not (see label **L4** from the Fig. 17).

A table constructed in such a way is to serve a concept approximation determined on a set of structured objects (see label **L5** from the Fig. 17). However, it frequently happens that in order to describe a structured object, apart from describing all parts of this object, a relation between the parts of this object should be described. Therefore, in constructing a table of concept approximation for a structured object, there is constructed an additional CRS-information system whose attributes entirely describe the whole structured object in terms of relations between the parts of this object (see label **L3**−**c** from the Fig. 17). In approximation of the object concerning structured objects, this system is

arranged together with other CRS-information systems constructed for individual parts of the structured objects (see label **L4** from the Fig. 17).

Similarly to the case of the concept approximation table for unstructured objects, the constraint relation is usually defined as a formula in the language GDL (see Definition 5) on the basis of attributes appearing in the obtained table. However, constraint relation may also be approximated using classifiers. In such a case providing examples of objects belonging and not belonging to constraint relation is required (see, *e.g.*, [78]).

The construction of a specific approximation table of a higher ontology level concept requires defining all elements appearing in Figs. 13 and 17. A fundamental problem connected with construction of an approximation table of the higher ontology level concept is, therefore, the choice of four appropriate languages used during its construction. The first language serves the purpose of defining patterns in a set of lower ontology level concept examples which enable the relational structure extraction. The second one enables defining the features of these structures. The third one enables to define relational structure clusters and finally the fourth one the properties of these clusters. All these languages must be defined in such a way as to make the properties of created relational structure clusters useful on a higher ontology level for approximation of the concept occurring there. Moreover, in the case when the approximated concept concerns structured objects each of the parts of this type of objects may require another four of the languages mentioned above.

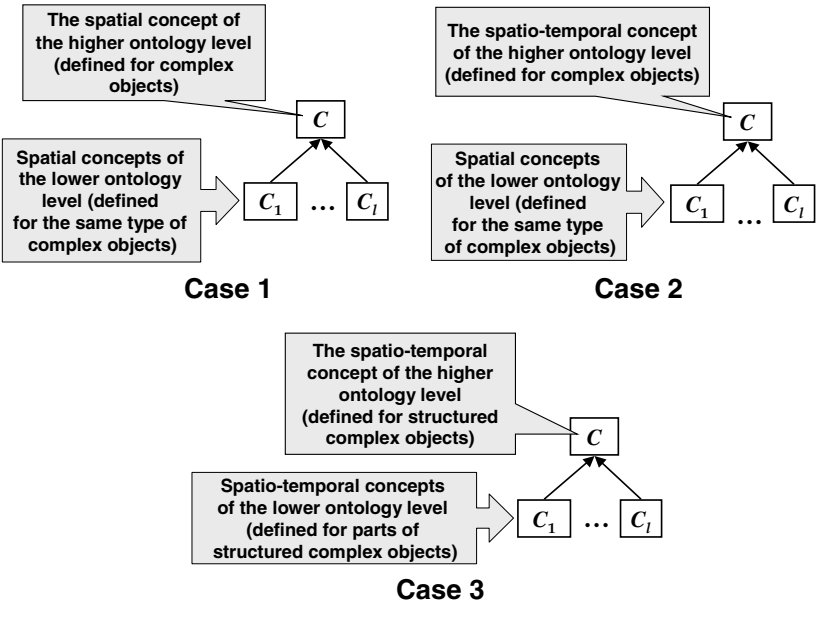


Fig. 18. Three cases of complex concepts approximation in ontology

However, the definition of these languages depends on semantical difference between concepts from both ontology levels. In this paper, we examine the following three situations in which the above four languages are defined in a completely different way (see Fig. 18).

1. The approximated concept C of the higher ontology level is a spatial concept (it does not require observing changes of objects over time) and it is defined on a set of the same objects as lower ontology level concepts (see **Case 1** from Fig. 18). On the lower level we have a concept family: $\{C_1, ..., C_l\}$, that are also spatial concept. Apart from that the concepts $\{C_1, ..., C_l\}$ are defined for unstructured objects without following their changes over time. That is why these concepts are defined on the basis of an object state observation at a single time point or time period established identically for all concepts. For example, the concept C and the concepts C_1,...,C_l may concern the situation of the same vehicle while concept C may be the concept of *Safe overtaking*. On the other hand, to the family of concepts C_1,...,C_l may belong such concepts as: *Safe distance from the opposite vehicle during overtaking*, *Possibility of going back to the right lane* and *Possibility of safe stopping before the crossroads*. The methods of approximation of the concept C for this case are described in Section 5.
2. The concept C under approximation is a spatio-temporal one (it requires observing object changes over time) and it is defined on the set of the same objects as the lower ontology level concepts (see **Case 2** from Fig. 18). On the lower level we have a concept family: $\{C_1, ..., C_l\}$, that are spatial concept. The concept C concerns object property defined in a longer time period than the concepts from the family $\{C_1, ..., C_l\}$. This case concerns a situation when following an unstructured object in order to capture its behavior described by the concept C, we have to observe it longer than it is required to capture behaviors described by concepts from the family $\{C_1, ..., C_l\}$. For example, concepts C_1,...,C_l may concern simple behaviors of a vehicle such as *acceleration*, *deceleration*, *moving towards the left lane*, while the concept C may be a more complex concept: *accelerating in the right lane*. Let us notice that determining whether a vehicle accelerates in the right lane requires its observation for some time which is called a time window. However, determining whether a vehicle increased its speed requires only the vehicle's speed registration at two neighboring instants. Such a case of the concept C approximation is described in Section 6.
3. The approximated concept C is a spatio-temporal one (it requires observing object changes over time) and it is defined on a set of structured objects, while concepts from the family $\{C_1, ..., C_l\}$ are determined on the set of parts of these objects; and at the same time the concept C concerns the structured object's behavior over a longer period of time than concepts from the family $\{C_1, ..., C_l\}$ (see **Case 3** from Fig. 18). This case concerns a situation when following a structured object in order to capture its behavior described by the concept C, we have to observe this object longer than it is required to capture behaviors of single part of this object described by concepts from

the family $\{C_1, ..., C_l\}$. For example, concepts from the family $\{C_1, ..., C_l\}$ may concern complex behaviors of a single vehicle such as *acceleration in the right lane*, *acceleration and changing lanes from right to left*, *decelerating in the left lane*. However, the concept C may be even more complex concept describing a behavior of a group of two vehicles (overtaking and overtaken) over a certain period of time, for example *the overtaking vehicle changes lanes from the right to left one while the overtaken vehicle drives in the right lane*. Let us notice that the behavior described by the concept C is an essential fragment of overtaking maneuver and determining if the group of two vehicles under observation behaved exactly that way requires observation for a certain time of behavior sequence of vehicles taking part in maneuvers such as *accelerating in the right lane*, *accelerating and changing lanes from right to left*, *maintaining a stable speed in the right lane*. This most complex case of the approximation of the concept C also is described in Section 6.

5 Approximating Spatial Concepts from Ontology

In the present subsection, we describe the case of approximating the concept C from the higher ontology level using concepts C_1,...,C_k from the lower ontology level when approximated concept C is defined on the set of the same objects as concepts C_1,...,C_k. Moreover, both concept C and concepts C_1,...,C_k concern object properties without observing their changes over time. In this paper such concepts are called spatial concepts. The example below describes a classic situation of this type resulting from an ontology obtained from a road traffic simulator (see Appendix A).

Example 13. Let us consider a situation when all ontology concepts concern the same type of objects, that is, vehicles. We deal with this type of situation in the case of ontology from Fig. 7. To each concept of this ontology there belong vehicles satisfying a specific condition expressed in a natural language. For example, to the concept of *Safe overtaking* there belong all vehicles which overtake safely, whereas to the concept of *Possibility of safe stopping before the crossroads* these vehicles whose speed is low enough to safely stop before the crossroads. The concepts of the lowest ontology level, that is, *Safe distance from the opposite vehicle during overtaking*, *Possibility of driving back to the right lane*, *Possibility of safe stopping before the crossroads*, *Safe distance from the front vehicle*, *Forcing the right of way* and *Safe distance from the front vehicle* are sensor concepts, that is, they may be approximated directly using sensor data. For example, the concept of *Possibility of safe stopping before the crossroads* may be approximated using such sensor attributes as *the speed of the vehicle*, *the acceleration of the vehicle*, *the distance from the crossroads*, *visibility* and *humidity*. However, concepts of the higher ontology level, that is, *Safe overtaking* and *Safe driving* should be approximated using concepts from the lower ontology level. For example, the concept of *Safe overtaking* may be approximated using the three following concepts: *Safe distance from the opposite vehicle during overtaking*, *Possibility of going back to the right lane* and *Possibility of safe stopping before the crossroads*. □

In order to approximate the higher ontology level concept (for example, the concept of *Safe overtaking* in the above example), an approximation table should be constructed for this concept according to Fig. 13. In order to do this a special language PEC is necessary, whose definition we provide in the next subsection.

5.1 Language of Patterns Extracted from a Classifier

If the approximation of the lower level ontology concepts is performed, then for each of these concepts we have at our disposal a classifier which is an algorithm returning for any tested object (that is, relational structure of the lower ontology level) the information about whether this object belongs to the concept or not. This type of information coming from all classifiers approximating lower ontology level concepts may serve the construction of binary attributes which describe crucial properties of the object of the higher ontology level. However, it should not be expected that in a general case the membership of objects to the concept of lower ontology level determines the membership of objects to the concept of higher ontology level. For example, if we assume that the concept of *safe overtaking* depends on the three following concepts: *safe distance from the opposite vehicle during overtaking*, *possibility of driving back to the right lane* and *possibility of safe stopping before the crossroads* (see Fig. 7), then it is hard to expect that a given vehicle overtakes safely only when it belongs to these three concepts. On the other hand, it is hard to expect that if the vehicle does not belong to one of these three concepts, then it definitely does not overtake safely. For example, if the distance from the oncoming vehicle is not safe, that is, a head-on collision of the overtaking and oncoming vehicles is possible, then it cannot be determined that the overtaking is safe. However, there are probably situations when the precise membership of the vehicle to the three concepts above cannot be acknowledged, but the expert will still claim in the natural language that *the overtaking is safe*, or that *the overtaking is almost safe* or *that the overtaking is safe to some degree*. Therefore, in this paper to construct attributes describing object properties from the lower ontology level, we propose stratifying classifiers which must certainly be constructed previously for lower ontology level concepts. This type of attributes inform in a more detailed way about the membership of objects to the lower ontology level concepts and because of that they are more useful to approximate higher level concepts.

Let us, now, define a *language of patterns extracted from a classifier* (PEC) which are used to describe object properties which are positive and negative examples of ontology concepts.

Definition 12 (*A language of patterns extracted from a classifier*)**.** *Let us assume that:*

- $\mathbf{A} = (U, A, d)$ *is a decision table, whose objects are relational structures and examples (positive and negative) for some concept* C*, described by a binary attribute* d*,*
- μ_C^E *is a stratifying classifier for the concept* C*, which classifies objects from* U *to* l*-layers, denoted be labels from the set* $E = \{e_1, ..., e_l\}$*, where the following three conditions are satisfied (see also Section 3.1):*

1. *layer e_1 includes objects which, according to an expert, certainly do not belong to concept C (so they belong to a lower approximation of its complement),*
2. *for every two layers e_i, e_j (where $i < j$), layer e_i includes objects which, according to an expert, belong to concept C with a degree of certainty lower the degree of certainly of membership of objects of e_j in U,*
3. *layer e_l includes objects which, according to an expert, certainly belong to concept C, viz., to its lower approximation.*

1. *The language of patterns extracted from a classifier μ_C^E (denoted by $PEC(\mu_C^E)$ or PEC-language, when μ_C^E is fixed) is defined in the following way:*
   - *the set $AL_{PEC}(\mu_C^E) = \{\mu, \in, \neg, \wedge, \vee\} \cup (2^E \setminus \emptyset)$ is an alphabet of the language $PEC(\mu_C^E)$,*
   - *expressions of the form $(\mu \in B)$, for any $B \subseteq E$, are atomic formulas of the language $PEC(\mu_C^E)$.*
2. *The semantics of atomic formulas from the language $PEC(\mu_C^E)$ is defined for any $B \subseteq E$ in the following way:*

$$|\mu \in B|_{PEC(\mu_C^E)} = \left\{u \in U : \mu_C^E(u) \in B\right\}.$$

The issue of defining atomic formulas themselves (expressions of type $\mu \in B$, where $B \subseteq E$) belonging to the sets of formulas of the language mentioned above also requires an explanation. Because concept layers from the set E are ordered, then the formulas of the form $\mu \in B$ are defined with the help of relation $=$, $\neq$, $<$, $\leq$, $>$ and $\geq$. For example, the $\mu = e_2$ formula describes these objects from the set U which the stratifying classifier μ_C^E classifies to the layer marked by e_2, however, the formula of the form $\mu \geq e_3$ describes these objects from the set U which the stratifying classifier μ_C^E classifies to the e_3 layer or higher, that is, to one of the $e_3, e_4, ..., e_l$ layers.

If it is known which stratifying classifier is used to define the language PEC for a specific concept C and if the set of layers E of the approximated concept is known, then we often use simplification of pattern description consisting in replacing (in formulas of the language PEC) the μ symbol with the name of the approximated concept, which enable to simplify the records in pattern presentation for several concepts at the same time. For example, in approximation of concept C using the stratifying classifier μ_C^E (where $E = \{e_1, ..., e_l\}$), pattern $(\mu \geq e_3)$ are recorded as $C \geq e_3$.

Because the language PEC is the language for construction of the structural relation properties, then each formula of that language in a given information system can be called a pattern. Some of these patterns are of great significance in practical applications. Therefore, we give them special names. The first pattern of this type is the so-called *concept layer pattern* which describes objects belonging to one of the concept layers.

Definition 13. *Let C be a concept and μ_C^E be a stratifying classifier for the concept C, which classifies objects to l-layers, denoted be labels from the set $E = \{e_1, ..., e_l\}$ (see conditions from Definition 12). Any pattern of the form $(\mu = e_i)$, where $i \in \{1, ..., l\}$, is called a layer pattern of concept C.*

Each layer pattern may be treated as a simple classifier which can classify objects matching this pattern. Thus, it is possible to select objects which belong to one of the concept layers. However, frequently in practice such accuracy of indicating one layer for a tested object is often not necessary. For example, we may be interested in patterns which describe such layers which certainly do not precede the established layer. These patterns correspond to the situation when we wish to recognize such concepts that belong to the concept with certainty at least equal to the previously established certainty level. For example, if we consider the concept of *safe overtaking* which has six linearly organized layers *"certainly NO"*, *"rather NO"*, *"possibly NO"*, *"possibly YES"*, *"rather YES"* and *"certainly YES"*, then the $\mu \geq$ "*possibly YES*" pattern describes such vehicles that perhaps overtake safely, rather overtake safely and certainly overtake safely. Hence, this pattern may be useful as a classifier which is not too certain. It is easy to change it to $\mu \geq$ "*rather YES*" by this increasing the certainty of its classification.

Due to practical applications of the above patterns we use a special term to call them, which is given in the definition below.

Definition 14. *Let C be a concept and μ_C^E be a stratifying classifier for the concept C, which classifies objects to l-layers, denoted be labels from the set $E = \{e_1, ..., e_l\}$ (see Definition 12). Any pattern of the form $(\mu \geq e_i)$, where $i \in \{1, ..., l\}$, is called a production pattern of concept C.*

The term from the above definition results from the fact that these types of patterns find application in production rule construction (see Section 5.3).

5.2 Concept Approximation Table Using Stratifying Classifiers

Currently, we present a definition of the approximation table of the higher ontology level concept with the help of stratifying classifiers.

Definition 15 (*A concept approximation table using stratifying classifiers*)**.** *Let us assume that:*

- $\mathbf{A} = (U, A)$ *is a given information system of unstructured objects of the fixed type,*
- C *is a concept, dependent in some ontology on concepts C_1,...,C_k, where $C \subseteq U$ and $C_i \subseteq U$, for $i = 1, ..., k$,*
- $\mathbf{T}_i = (U, A_i, d_{C_i})$ *is a decision table constructed for approximation of the concept C_i such that $A_i \subseteq A$ for $i = 1..., k$ and d_{C_i} is a decision attribute which values describe the membership of objects from U to the concept C_i,*
- $\mu_{C_i}^{E_i}$ *is a stratifying classifier for the concept C_i constructed on the basis the table $\mathbf{T}_i$, which classifies objects from the set U to layers, denoted be labels from the set E_i, for $i = 1, ..., k$,*
- $\Phi_i = \{\phi_i^1, ..., \phi_i^{l_i}\}$ *is a family of patterns defined by formulas from the language $PEC(\mu_{C_i}^{E_i})$, which can be used to define new attributes (features) for objects from the set U, for $i = 1..., k$,*

- $\mathcal{P}_{PEC} = (U, \Phi, \models_{PEC})$ *is a property system, where* $\Phi = \bigcup_{i=1}^{k} \Phi_i$ *and the satisfiability relation* $\models_{PEC}$ *is defined in the following way:*

$$\forall (u, \phi) \in U \times \Phi : \; u \models_{PEC} \phi \;\Leftrightarrow$$

$$u \models_{PEC(\mu_{C_i}^{E_i})} \phi, \quad \textit{for } i \in \{1, .., k\} \textit{ such that } \phi \in \Phi_i,$$

- $\mathbf{A}_\Phi = (U, A_\Phi)$ *is an information system defined be the property system* $\mathcal{P}_{PEC}$,
- $\mathbf{R}_C \subseteq U$ *is a relation of constraints defined by a formula* $\Psi \in GDL(\mathbf{A}_\Phi)$ *that is* $\forall_{u \in U} \; u \in \mathbf{R}_C \Leftrightarrow u \models_{GDL(\mathbf{A}_\Phi)} \Psi$.

A concept approximation table using stratifying classifiers for the concept C *relatively to concepts* C_1,...,C_k *is a decision table* $\mathbf{A}_C = (U_C, A_C, d_C)$, *where:*

- $U_C = \mathbf{R}_C$,
- $A_C = A_\Phi$,
- *the attribute* d_C *describes membership of objects from the set* U *to the concept* C.

According to the above definition, the conditional attributes of concept approximation table are constructed on the basis of stratifying classifiers $\mu_{C_1}^{E_1}, ..., \mu_{C_k}^{E_k}$ which were generated for concepts of the lower ontology level C_1,...,C_k and for layer sets E_1,...,E_k. It ought to be stressed, however, that the number of layers and the layout of layers in sets E_1,...,E_k should be chosen in such a way as to serve effective approximation of complex concept C. In order to do this the layers are chosen by an expert on the basis of the domain knowledge or are obtained on the basis of suitably designed layering heuristics (see Section 3).

It is easy to notice that the table of concept approximation from ontology using stratifying classifiers defined above is a special case of a concept approximation table mentioned in Section 4.10.

Let us now go back to Example 7 which concerned approximation of the concept of *Safe overtaking*.

Example 14 (The continuation of Example 13). For concept *Safe overtaking* approximation we wish to construct an approximation table according to Definition 15. First, stratifying classifiers for concepts *Safe distance from the opposite vehicle during overtaking* (C_{SDOV}), *Possibility of going back to the right lane* (C_{PGBR}) and *Possibility of safe stopping before the crossroads* (C_{SSBC}) should be constructed. Next, conditional attributes are constructed which are defined as patterns in the language PEC, individually for each of the three concepts. The choice of appropriate patterns takes place on the basis of domain knowledge. In the simplest case they may be layer patterns for all layers of concepts C_{SDOV}, C_{PGBR} and C_{SSBC}. Next, on the basis of domain knowledge a relation of constraints is established and used to arrange an approximation table for the *Safe overtaking* concept, leaving only objects which belong to this relation. Finally, also on the basis of domain knowledge values of decision attribute from the *Safe overtaking* concept approximation table is established. □

The concept approximation table using stratifying classifiers may be used for building a classifier which ensures approximation of this concept. The approximation may take place using classical classifiers (see Section 2) or stratifying classifiers (see Section 3).

Slightly similar approaches have been successfully applied to approximate concepts in different ontologies (see, *e.g.*, [80, 81, 179, 306, 307]). They have also been applied in ontology from Fig. 7 (see [179, 306]) obtained from the road simulator (see Appendix A). However, in this paper we are more interested in other methods of classifier construction using language PEC which use *production rules*, *productions* and *approximate reasoning schemes*. These methods are described in the next few subsections.

5.3 Production Rules

The production rule (see, *e.g.*, [77, 89, 172, 188, 189, 190, 191, 192, 193]) is a kind of decision rule which is constructed on two adjacent levels of ontology. In the predecessor of this rule there are patterns for the concepts from the lower level of ontology while in the successor the pattern for one concept from the higher level of ontology (connected by relationships with concepts from the rule predecessor).

Definition 16 (*A production rule*)**.** *Let us assume that:*

- $\mathbf{A} = (U, A)$ *is a given information system of unstructured objects of the fixed type,*
- C *is a concept, dependent in some ontology on concepts* C_1,...,C_k, *where* $C \subseteq U$ *and* $C_i \subseteq U$, *for* $i = 1, ..., k$,
- $\mathbf{T}_i = (U, A_i, d_{C_i})$ *is a decision table constructed for approximation of the concept* C_i *such that* $A_i \subseteq A$ *for* $i = 1..., k$ *and* d_{C_i} *is a decision attribute which values describe the membership of objects from* U *to the concept* C_i,
- $\mu_{C_i}^{E_i}$ *is a stratifying classifier for the concept* C_i *constructed on the basis the table* $\mathbf{T}_i$, *which classifies objects from the set* U *to layers, denoted be labels from the set* E_i, *for* $i = 1, ..., k$,
- $\mathbf{A}_C = (U_C, A_C)$ *is a concept approximation table for the concept* C *using stratifying classifiers* $\mu_{C_i}^{E_i}$, *for* $i = 1, ..., k$,
- μ_C^E *is a stratifying classifier for the concept* C, *which classifies objects from the set* U_C *to layers, denoted be labels from the set* E.

1. *If* $p_i \in PEC(\mu_{C_i}^{E_i})$ *is a production pattern for the concept* C_i *(for* $i = 1, ..., k$*) and* $p \in PEC(\mu_C^E)$ *is a production pattern for the concept* C *then any formula of the form:*

$$p_1 \wedge ... \wedge p_k \Rightarrow p \tag{6}$$

is called a production rule for the concept C *relatively to concepts* $C_1, ..., C_k$ *if and only if the following conditions are satisfied:*
(a) exists at least one object $u \in U_C$ *such that:*

$$u \models_{PEC(\mu_{C_i}^{E_i})} p_i \text{ for } i = 1, ..., k,$$

(b) for any object $u \in U_C$*:*

$$u \models_{PEC(\mu_{C_1}^{E_1})} p_1 \wedge ... \wedge u \models_{PEC(\mu_{C_k}^{E_k})} p_k \Rightarrow u \models_{PEC(\mu_C^E)} p$$

2. *The first part of production rule (i.e.,* $p_1 \wedge ... \wedge p_k$*) is called a predecessor of production rule, whilst the second part of production rule (i.e.,* p*) is called a successor of production rule.*
3. *The concept from the upper level of production rule (from successor of rule) is called a target concept of production rule, whilst the concepts from the lower level of production rule (from predecessor of rule) are called source concepts of production rule.*

Below, we present an example of production rule.

Example 15. We consider the concept C which depends on concepts C_1 and C_2 (in some ontology). Besides, concepts C, C_1 and C_2 have six linearly organized layers *"certainly NO"*, *"rather NO"*, *"possibly NO"*, *"possibly YES"*, *"rather YES"* and *"certainly YES"*. In Fig. 19 we present an example of production rule for concepts C_1, C_2 and C. This production rule has the following

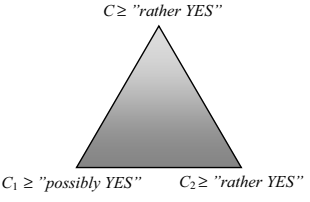


Fig. 19. The example of production rule

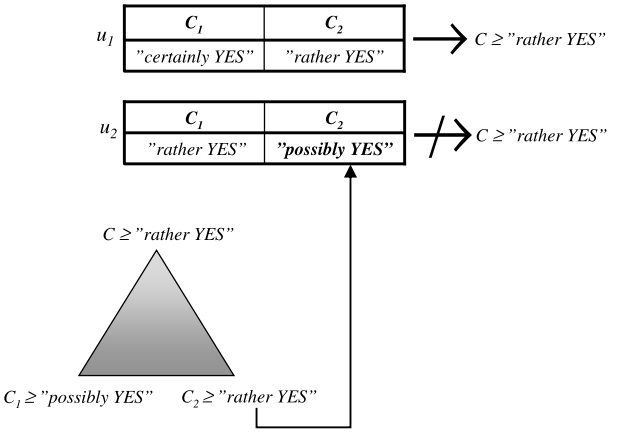


Fig. 20. Classifying tested objects by single production rule

interpretation: if inclusion degree to a concept C_1 is at least *"possibly YES"* and to concept C_2 at least *"rather YES"* then the inclusion degree to a concept C is at least *"rather YES"*. □

A rule constructed in such a way may serve as a simple classifier enabling the classification of objects matching the patterns from the rule predecessor into the pattern from the rule successor. The tested object may be classified by a production rule if it matches all patterns from the production rule predecessor. Then the production rule classifies a tested object to the target (conclusion) pattern.

For example, the object u_1 from Fig. 20 is classified by production rule from Fig. 19 because it matches both patterns from the left hand side of the production rule whereas, the object u_2 from Fig. 20 is not classified by production rule because it does not match the second source pattern of production rule (the value of attribute C_2 is less than *"rather YES"*).

The domain of a given production rule is a set of all objects matching all patterns from the predecessor of this rule.

5.4 Algorithm for Production Rules Inducing

Production rules can be extracted from data using domain knowledge. In this section we present an exemplary algorithm for the production rule inducing. The basic structure of this algorithm's data is a special table called *a layer table* which we define for the approximated concept in ontology.

Definition 17 (*A layer table*)**.** *Let us assume that:*

- $\mathbf{A} = (U, A)$ *is a given information system of unstructured objects of the fixed type,*
- C *is a concept, dependent in some ontology on concepts* C_1,...,C_k*, where* $C \subseteq U$ *and* $C_i \subseteq U$*, for* $i = 1, ..., k$*,*
- $\mathbf{T}_i = (U, A_i, d_{C_i})$ *is a decision table constructed for approximation of the concept* C_i *such that* $A_i \subseteq A$ *for* $i = 1..., k$ *and* d_{C_i} *is a decision attribute which values describe the membership of objects from* U *to the concept* C_i*,*
- $\mu_{C_i}^{E_i}$ *is a stratifying classifier for the concept* C_i *constructed on the basis the table* $\mathbf{T}_i$*, which classifies objects from the set* U *to layers, denoted be labels from the set* E_i*, for* $i = 1, ..., k$*,*
- $\mathbf{A}_C = (U_C, A_C)$ *is a concept approximation table for the concept* C *using stratifying classifiers* $\mu_{C_i}^{E_i}$*, for* $i = 1, ..., k$*,*
- μ_C^E *is a stratifying classifier for the concept* C*, which classifies objects from the set* U_C *to layers, denoted be labels from the set* E*.*

A layer table for the concept C *relatively to concepts* C_1,...,C_k *is a decision table* $\mathbf{LT}_C = (U, A, d)$*, where:*

- $U = U_C$
- $A = \{a_{C_1}, ..., a_{C_k}\} \cup \{a_C\}$*, where for any object* $u \in U$ *attributes from the set* A *are defined in the following way:*

- $a_{C_i}(u) = \mu_{C_i}^{E_i}(u)$ *for* $i = 1, ..., k$,
- $a_C(u) = \mu_C^E(u)$.

The layer table for a given concept C, which depends on concepts C_1, ..., C_k in ontology, stores layer labels of objects belonging to the $\mathbf{A}_C$ table.

Example 16. Let us assume that in a certain ontology the concept C depends on concepts C_1 and C_2. Moreover, each of these six concepts has six linearly organized layers: *"certainly NO"*, *"rather NO"*, *"possibly NO"*, *"possibly YES"*, *"rather YES"* and *"certainly YES"*. The Fig. 21 presents a sample table of layers for these concepts. □

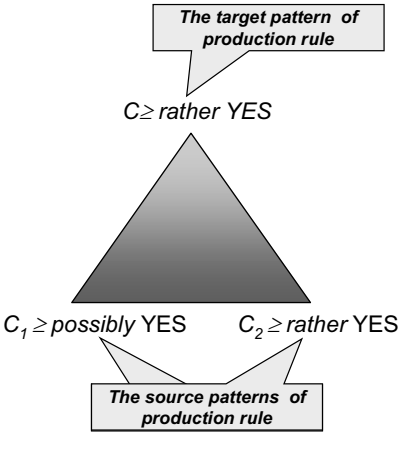
Now, an algorithm of production rule searching may be presented (see Algorithm 5.1). It works on the basis of a layer table and as a parameter requires providing a layer which occurs in the successor of the production rule.

In Fig. 21 we illustrate the process of extracting production rule for concept C and for the approximation layer "*rather YES*" of concept C. Is is easy to see that if from the table $\mathbf{LT}_C$ we select all objects satisfying $a_C =$ "*rather YES*", then for selected objects minimal value of the attribute a_{C_1} is equal to "*possibly YES*" and minimal value of the attribute a_{C_2} is equal to "*rather YES*". Hence, we obtain the production rule:

$$(C_1 \geq \text{"}possibly\ \ YES\text{"}) \wedge (C_2 \geq \text{"}rather\ \ YES\text{"}) \Rightarrow (C \geq \text{"}rather\ \ YES\text{"}).$$

The method of extracting production rule presented above can be applied for various values of attribute a_C. In this way, we obtain a collection of production rules, that we mean as a production (see Section 5.6).

a_{C_1}	a_{C_2}	a_C
certainly YES	certainly YES	**certainly YES**
certainly NO	certainly NO	certainly NO
rather YES	certainly YES	**rather YES**
possibly YES	possibly NO	possibly YES
possibly YES	possibly NO	rather NO
possibly YES	**rather YES**	**rather YES**
possibly YES	certainly NO	possibly NO
certainly YES	**rather YES**	**certainly YES**
certainly NO	possibly NO	certainly NO



certainly NO < rather NO < possibly NO < possibly YES < rather YES < certainly YES

Fig. 21. The illustration of production rule extracting

Algorithm 5.1. Extracting of production rule

Input:

1. concept C, dependent on concepts C_1,...,C_k (in some ontology).
2. layer table $\mathbf{LT}_C$ for concept C,
3. label e of layer, that can be placed in the successor of computed production rule.

Output: The production rules with the pattern $C \geq e$ placed in its successor.

begin
  Select all rows from the table $\mathbf{LT}_C$ in which values of column a_C is not less than e.
  Find minimal values $e_1, ..., e_k$ of attributes $a_{C_1}, ..., a_{C_k}$ from table $\mathbf{LT}_C$ for selected rows in the previous step.
  Set sources patterns of new production rule on the basis of minimal values $e_1, ..., e_k$ of attributes that were found in the previous step.
  Set the target pattern of new production, i.e., concept C with the value e.
  return $(C_1 \geq e_1) \wedge ... \wedge (C_k \geq e_k) \Rightarrow (C \geq e)$
end

5.5 Relation of Production Rules with DRSA

In 1996 Professor Greco, Professor Matarazzo and Professor Słowiński proposed a generalization of rough set theory for the need of multi-criteria decision problems (see, *e.g.*, [308, 309, 310, 311]). The main idea of this generalization is replacing the indiscernibility relation with the dominance relation. This approach is known under the *Dominance-based Rough Set Approach* (DRSA). In DRSA it is assumed that the values of all attributes of a given decision table (together with the decision attribute) are organized in a preferential way, that is, they are the so-called *criteria*. This means that for each attribute a from a given decision table a two-argument outranking relation is defined on the set of objects from this table, and at the same time a pair (x, y) belongs to this relation if the object x is *at least as good as* object y with regard to the criterion a. Using the outranking relation the dominance relation of one object on another is defined with respect to the established criteria set (attributes). Namely, the object x dominates object y with respect to the criteria set B (attributes), when x outranks y with respect to all criteria from B.

In DRSA it is possible to construct specific decision rules which is are called *dominance-based decision rules* (see, *e.g.*, [312]). Elementary conditions in the conditional part of these rules represent the statement that the object satisfy a criterion a (attribute) at least (or at most) as good as a certain established value of attribute a. Moreover, decision parts of the rules indicate that the object belongs to at least (or at most) to a given decision class.

Let us assume that there is given the layer table $\mathbf{LT}_C$ for the concept C which depends on concepts C_1,...,C_k in a certain ontology. It is easy to notice that each production rule (see Section 5.3) computed for the table $\mathbf{LT}_C$ by Algorithm 5.1 is a specific case of dominance-based rule (in the DRSA approach) established for the table $\mathbf{LT}_C$. Namely, it is such a case of dominance-based rule that when in the dominance-based rule predecessor, we consider the expression "the object is at least as good as" in relation the concept layers C_1,...,C_k represented by conditional attributes of $\mathbf{LT}_C$ table, and in the dominance-based rule successor "the object belongs at least to a given decision class" where decision classes are layers of the concept C. Moreover, in the rule predecessor of such a dominance-based rule there occur all conditional attributes.

On account of that, to establish production rules we may successfully apply algorithms known from literature for the induction of rules in the DRSA approach (see, *e.g.*, [312, 313, 314, 315]).

Using this approach to establish production rules, it should be remembered that the calculated dominance-based rules do not often have all conditional attributes in the predecessor. Meanwhile, according to the definition, each production rule has all conditional attributes from the table $\mathbf{LT}_C$ in the predecessor. However, with an appropriate interpretation each dominance-based rule may be treated as a production rule. It is enough to add to the predecessor all descriptors corresponding to the rest of conditional attributes; and at the same time each of the new descriptors must be constructed in such a way as to make all tested objects match it. This effect may be achieved by placing in the descriptor the attribute value representing the smallest preference possible.

5.6 Productions

Although a single production rule may be used as a classifier for the concept appearing in a rule predecessor, it is not yet a complete classifier, i.e., allowing to classify all objects belonging to an approximated concept, and not only those which match concepts of a rule predecessor. Therefore, in practice production rules are grouped into the so called *productions* (see, *e.g.*, [77, 89, 172, 188, 189, 190, 191, 192, 193]), i.e., production rule collections, in a way that to each production there belong rules having patterns for the same concepts in a predecessor and successor, but responding to their different layers.

Definition 18 (*A production*)**.** *Let us assume that:*

- $\mathbf{A} = (U, A)$ *is a given information system of unstructured objects of the fixed type,*
- C *is a concept, dependent in some ontology on concepts* C_1,...,C_k*, where* $C \subseteq U$ *and* $C_i \subseteq U$*, for* $i = 1, ..., k$*,*
- $\mathbf{T}_i = (U, A_i, d_{C_i})$ *is a decision table constructed for approximation of the concept* C_i *such that* $A_i \subseteq A$ *for* $i = 1..., k$ *and* d_{C_i} *is a decision attribute which values describe the membership of objects from* U *to the concept* C_i*,*
- $\mu_{C_i}^{E_i}$ *is a stratifying classifier for the concept* C_i *constructed on the basis the table* $\mathbf{T}_i$*, which classifies objects from the set* U *to layers, denoted be labels from the set* E_i*, for* $i = 1, ..., k$*,*

- $\mathbf{A}_C = (U_C, A_C)$ *is a concept approximation table for the concept* C *using stratifying classifiers* $\mu_{C_i}^{E_i}$*, for* $i = 1, ..., k$*,*
- μ_C^E *is a stratifying classifier for the concept* C*, which classifies objects from the set* U_C *to layers, denoted be labels from the set* E*.*

1. *A family of production rules* $P = \{r_1, ..., r_m\}$ *is a production if and only if for any pair of production rules* $r_i, r_j \in P$ *such that* $r_i = (p_1 \wedge ... \wedge p_k \Rightarrow p)$ *and* $r_j = (q_1 \wedge ... \wedge q_k \Rightarrow q)$ *and* $i < j$ *the following two conditions are satisfied:*
   - $|p_i|_{PEC(\mu_{C_i}^{E_i})} \subseteq |q_i|_{PEC(\mu_{C_i}^{E_i})}$ *for* $i = 1, ..., k$*,*
   - $|p|_{PEC(\mu_C^E)} \subseteq |q|_{PEC(\mu_C^E)}$.
2. *The domain of a given production is a sum of all domains of its production rules.*

Bellow, we present an example of production.

Example 17. In Fig. 22 we present three production rules constructed for some concepts C_1, C_2 and C approximated by three linearly ordered layers *"certainly NO"*, *"rather NO"*, *"possibly NO"*, *"possibly YES"*, *"rather YES"* and *"certainly YES"*. This collection of production rules is an exemplary production for concepts C_1, C_2 and C. Moreover, production rules from Fig. 22 have the following interpretation:

1. if inclusion degree to a concept C_1 is at least "*rather YES*" and to concept C_2 at least "*certainly YES*" then the inclusion degree to a concept C is at least "*certainly YES*";
2. if the inclusion degree to a concept C_1 is at least "*possibly YES*" and to a concept C_2 at least "*rather YES*" then the inclusion degree to a concept C is at least "*rather YES*";
3. if the inclusion degree to a concept C_1 is at least "*possibly YES*" and to a concept C_2 at least "*possibly YES*" then the inclusion degree to a concept C is at least "*possibly YES*". □

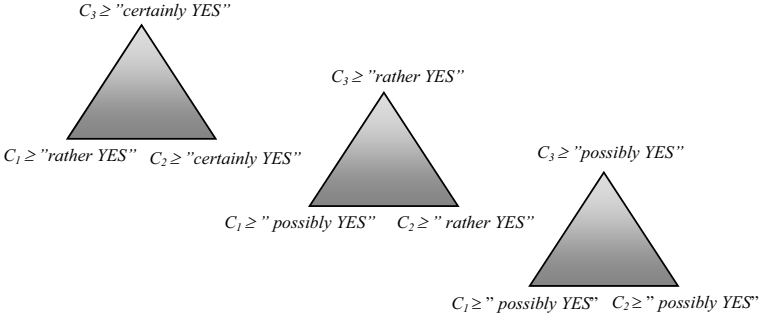


Fig. 22. The example of production as a collection of three production rules

Algorithm 5.2. Classifying objects by production

Input: Tested object u and production P
Output: The membership of the object u to the concept C (YES or NO)

```
1  begin
2  |  Select a complex concept C from an ontology (e.g., Safe overtaking).
3  |  if the tested object should not be classified by a given production P
   |  extracted for the selected concept C then
4  |  |  return HAS NOTHING TO DO WITH // The object does
   |  |     not satisfy the production guard
5  |  end
6  |  Find a rule from production P that classifies object with the maximal
   |  degree to the target concept of rule
7  |  if such a rule of P does not exist then
8  |  |  return I DO NOT KNOW
9  |  end
10 |  Generate a decision value for object from the degree extracted in the
   |  previous step
11 |  if the extracted degree is greater than fixed threshold (e.g., possibly
   |  YES) then
12 |  |  return YES // the object is classified to C
13 |  else
14 |  |  return NO // the object is not classified to C
15 |  end
16 end
```

In the case of production from Fig. 22 concept C is the target concept and C_1, C_2 are the source concepts.

Any production can be used as a classifier. The method of object classification based on production can be described as follows:

1. Preclassify object to the production domain.
2. Classify object by production.

We assume that for any production *a production guard* is given. Such a guard describes the production domain and is used in preclassification of tested objects. The production guard definition is usually based on the relation of constraints (see Section 4.10) and its usage consists in checking whether a given object satisfies the constraints, that is, if it belongs to the relation of constraints.

For example, let us assume that the production P is generated for the concept: *Is the vehicle overtaking safely?*. Then an object-vehicle u is classified by production P iff u is overtaking. Otherwise, it is returned a message *"HAS NOTHING TO DO WITH (OVERTAKING)"*.

Now, we can present an exemplary algorithm for classifying objects by production (see Algorithm 5.2).

It is worth noticing that for objects which went through preclassification positively, two cases should be distinguished: object classification through production and recognizing the object through production. Classification of object through production means that such a production rule is found in the production that the tested object matches all patterns of its predecessor. However, not classifying the object through production means that such a production rule is not found. There also exists a third possibility that the tested object is not recognized by production. It means that relying on production rules in production, it is neither possible to state whether the tested object is classified by production nor that it is not. This case concerns the situation when stratifying classifiers realizing in practice the production patterns in the production rule predecessor are not able to recognize a tested object. This difficulty may be greatly decreased or even removed by applying to production patterns defining such classifiers that always or almost always classify objects.

A questions arises whether Algorithm 5.2 is universal enough to serve not only classifying tested objects from a given concept approximation table $\mathbf{A}_C$ (see Definition 15), but also to classify objects belonging to the extension of this table. Algorithm 5.2 works on the basis of production P, which is a family of production rules generated for concept approximation table $\mathbf{A}_C$. The application of each production rule only requires computation of the values of conditional attributes of table $\mathbf{A}_C$. It is done with the use of stratifying classifiers which were generated for lower ontology level concepts C_1,...,C_k. These classifiers are based on decision rules, therefore they may effectively classify tested objects outside a given information system $\mathbf{A}$ (see Definition 15 and Section 2.8). It would seem that this property transfers to Algorithm 5.2 where tested objects are classified with the help of production rules. Unfortunately, although to classify tested objects outside table $\mathbf{A}_C$ production rules may be applied, it is quite natural that production rules classify tested objects incorrectly. It results from the fact that production rules were constructed on the basis of dependencies observed between the attribute values of table $\mathbf{A}_C$, whereas in the extension of this table these dependencies may not occur. Therefore, similarly to the case of classifiers based on decision rules, while using production rules to classify objects, arguments for and against the membership of the tested object to a given concept should be taken into consideration. Obviously, such duality of arguments leads to conflicts in classifying tested objects and these conflicts must be appropriately resolved. In practice, it means that apart from production rules classifying a tested object to a given concept C, also production rules classifying tested objects to the complement of concept C should be taken into consideration. The complement of a given concept may be treated as a separate concept $C' = U \setminus C$. Production rules may also be generated for concept C' with the help of Algorithm 5.1. It requires, however, a suitable redefining layers of concepts C_1,...,C_k and sometimes using another ontology to approximate C'. As a result we obtain table $\mathbf{A}_{C'}$, which may serve generating production rules. Having production P_C generated for concept C and production $P_{C'}$ for concept C', the Algorithm 5.2 may be modified in such a way as to be able to resolve conflicts which may occur

between production rules from P_C and $P_{C'}$. In this paper, we propose the following way of resolving these conflicts. If p_C and $p_{C'}$ are production rules chosen by Algorithm 5.2 from productions P_C and $P_{C'}$ respectively, then the tested object is classified to concept C only when the degree of certainty of classification by p_C is higher than the degree of certainty of classification by $p_{C'}$. Otherwise, the tested object is classified to C'.

5.7 Approximate Reasoning Schemes

Both productions and production rules themselves are only constructed for the two adjacent levels of ontology. Therefore, in order to use the whole ontology fully there are constructed the so called *approximate reasoning schemes* which are hierarchical compositions of production rules (see, *e.g.*, [77, 89, 172, 188, 189, 190, 191, 192, 193]).

The synthesis of AR-scheme is carried out in a way that to a particular production rule r lying on a lower hierarchical level of AR-scheme under construction another production rule r' on a higher level may be attached. However, this may be done only if one of the concepts for which the pattern occurring in the predecessor of r' was constructed is the concept corresponding to the successor pattern of the rule r. Additionally, it is required that the pattern occurring in a rule predecessor from the higher level is a pattern superset occurring in a rule successor from the lower level (in the sense of inclusion object sets matching both patterns). To the two combined production rules some other production rules can be attached (from above, from below or from the side) and in this way a multilevel structure is made which is a composition of many production rules.

In Fig. 23 we have two productions. The target concept of the first production is C_5 and the target concept of the second production is the concept C_3. We select one production rule from the first production and one production rule from the second production. These production rules are composed and then a simple AR-scheme is obtained that can be treated as a new two-levels production rule. Notice, that the target pattern of lower production rule in this AR-scheme is the same as one of the source patterns from the higher production rule. In this case, the common pattern is described as follows: inclusion degree (of some pattern) to a concept C_3 is at least *"possibly YES"*.

In this way, we can compose AR-schemes into hierarchical and multilevel structures using productions constructed for various concepts. AR-scheme constructed in such a way can be used as a hierarchical classifier whose input is given by predecessors of production rules from the lowest part of AR-scheme hierarchy and the output is a successor of a rule from the highest part of the AR-scheme hierarchy.

In this paper, there are proposed two approaches for constructing AR-schemes. The first approach is based on determining productions for a given ontology with the use of available data sets. Next, on the basis of these productions many AR-schemes are arranged which classify objects to different patterns on different ontology levels. All these productions and AR-schemes are stored in memory and their modification and potential arrangement of new AR-schemes is possible.

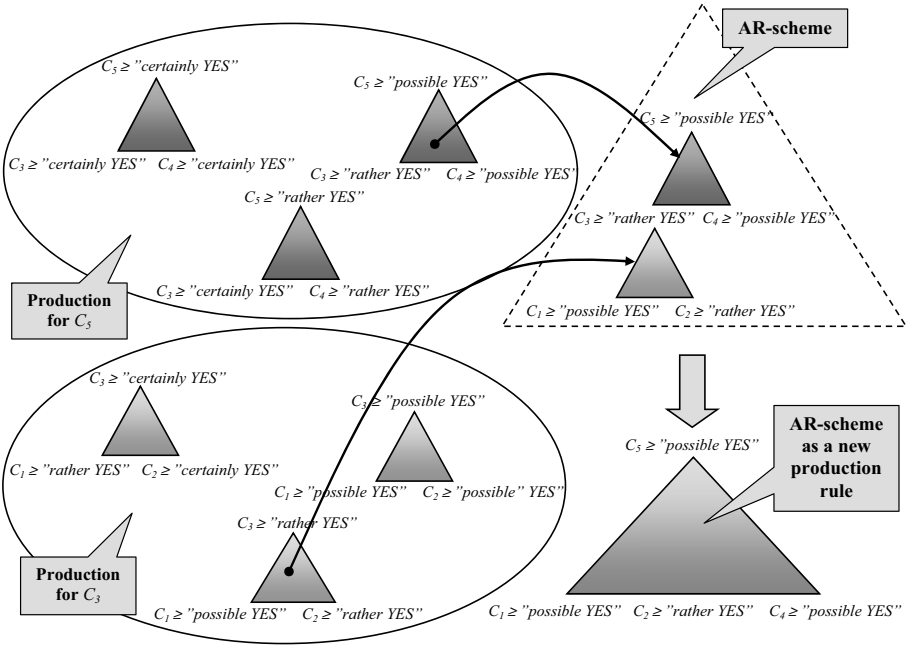


Fig. 23. Synthesis of approximate reasoning scheme

Hence, if a certain tested object should be classified, it is necessary to search in memory an AR-scheme appropriate for it and use it to classify the object. This approach enables steering the object classification depending on the expected certainty degree of the obtained classification. The drawback of this approach is a need of a large memory with a quick access to production and AR-schemes storage.

The second approach is based on a dynamic construction of AR-schemes. It is realized in a way that only during tested object classification itself, having been given different productions, an appropriate AR-scheme for classifying this particular object is built. Hence, this approach does not require so much memory as the previous approach. However, to its application we need the method of production method selection in dynamic construction of AR-schemes for tested object classification. A certain proposal of such a method is given by Algorithm 5.2 which suggests selecting from production such a production rule that recognizes the object, i.e., the object matches all patterns from the rule predecessor and production pattern from the successor of such a rule is based on a possibly highest layer, i.e., such a rule classifies the object possibly in the most certain way.

However, similarly to the case of a single production rule one AR-scheme is not yet a full classifier. That is why in practice there are many AR-schemes constructed for a particular concept which approximate different layers or

concept regions. For example, on the basis of two productions from Fig. 23 three AR-schemes may be created which we show in Fig. 24. Each of these schemes is a classifier for one of production patterns constructed for the concept C_5.

The possibility of creating many AR-schemes for one concept is of a great practical significance, because tested objects may be classified to different production patterns which enable to capture the certainty degree with regard to membership of the tested object to the concept. For example, let us assume that we constructed five AR-schemes for the concept C_{SD} (safe driving) corresponding to production patterns: $C_{SD} \geq$ "*certainly YES*", $C_{SD} \geq$ "*rather YES*", $C_{SD} \geq$ "*possibly YES*", $C_{SD} \geq$ "*possibly NO*" and $C_{SD} \geq$ "*rather NO*". If a certain tested object is not classified by the AR-scheme constructed for the $C_{SD} \geq$ "*certainly YES*" pattern, then we cannot conclude with certainty that this object is driving safely. However, what also should be checked is the fact if this object is not classified by the ARscheme constructed for the $C_{SD} \geq$ "*rather YES*" pattern (then we may conclude that the object rather drives safely) or by the AR-scheme constructed by the $C_{SD} \geq$ "*possibly YES*" pattern (which means that the vehicle perhaps drives safely). Only if the tested object is not classified by any of these three AR-schemes, we may conclude that the tested object is not going safely. Then the question arises, how dangerously

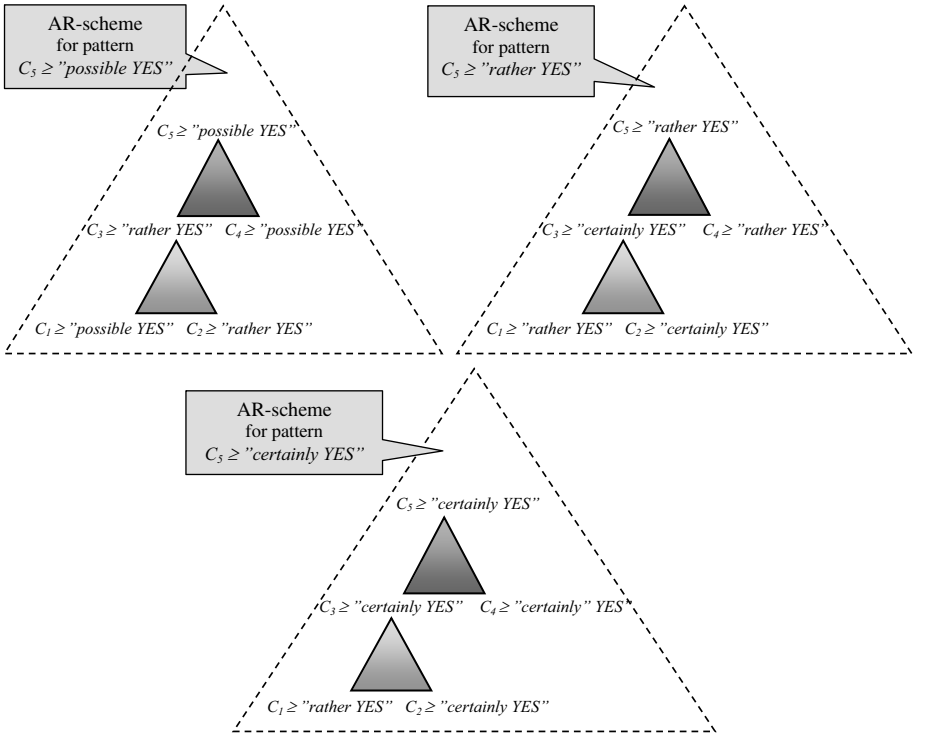


Fig. 24. Three approximate reasoning schemes for concept C_5

that object is behaving? To solve this it should be checked if the object is classified by the AR-scheme constructed for the $C_{SD} \geq$ "*possibly NO*" and then $C_{SD} \geq$ "*rather NO*" pattern. Only if none of these AR-schemes classifies the object, we may conclude that the tested object certainly does not go safely.

It is worth noticing that similarly to the case of production rule, two cases should be distinguished here: object classification by the AR-scheme and object recognition by the AR-scheme. Object classification by the AR-scheme means that the tested object belongs to all patterns lying at the bottom of the AR-scheme and this object is classified to the pattern lying at the top of the AR-scheme. However, not classifying the object means that the tested object does not belong to at least one of the patterns lying at the bottom of the AR-scheme. There is a third possibility that the tested object is not recognized by the AR-scheme. This means that, relying on a given AR-scheme, it is not possible to state that the tested object belongs to the pattern lying at the top of the AR-scheme. This case concerns the situation when stratifying classifiers executing, in practice, patterns lying at the bottom of the AR-scheme are not able to recognize the tested object. In such a situation with regard to classifying the tested object, there are two ways of procedure. Firstly, it may be acknowledged that the tested object cannot be classified using available AR-schemes. However, this approach frequently causes that the number unclassified objects is too large. Therefore, in practice the other approach is applied which consists in trying to classify the tested object with the AR-schemes classifying objects to patterns representing smaller certainty of the concept belonging, counting on the fact that such AR-schemes have a greater extension. The drawback of this approach is, however, the fact that a false resulting in decrease of the certainty of the tested object's membership to the concept is possible. This difficulty may be greatly diminished or even removed by applying, in the production pattern, such classifiers that always or almost always classify objects.

It is worth noticing that similarly to the case of production rules, in the case of using AR-schemes to construct classifiers, arguments for and against the membership of the tested object to a given concept should be taken into consideration. Thus, the obtained classifier will be able to serve effective classification not only of tested objects from a given concept approximation table $\mathbf{A}_C$ (see Definition 15) but also to classify objects belonging to the extension of this table. In practice it means that apart from AR-schemes which classify the tested object to a given concept, also AR-schemes which classify tested objects to the complement of this concept should be taken into consideration. Obviously, conflicts occurring at this point in classification of tested objects should be appropriately resolved, for instance like in the case of conflicts between production rules (see Section 5.6).

5.8 Experiments with Data

To verify effectiveness of classifiers based on AR schemes, we have implemented our algorithms in the AS-lib programming library. This is an extension of the

RSES-lib programming library creating the computational kernel of the RSES system (see Section 2).

The experiments have been performed on the data set obtained from the road simulator (see Appendix A). Data set consists of 18101 objects generated by the road simulator. We have applied the train and test method. The data set was randomly divided into two parts: training and test ones (50% + 50%). In order to determine the standard deviation of the obtained results each experiment was repeated for 10 random divisions of the whole data set.

In our experiments, we compared the quality of two classifiers: RS and ARS. For inducing RS we use RSES system generating the set of decision rules by algorithm LEM2 (see Section 2.4) that are next used for classifying situations from testing data. ARS is based on AR schemes.

During ARS classifier construction, in order to approximate concepts occurring in ontology we used the LEM2 algorithm (see Section 2.4).

For production rule construction we used the expert method of stratifying classifier construction (see Section 3.2). However, to classify objects using the ARS classifier we used the method of dynamic construction of the AR-schemes for specific tested objects (see Section 5.7).

We compared RS and ARS classifiers using the accuracy, the coverage, the accuracy for positive examples (the sensitivity or the true positive rate), the accuracy for negative examples (the specificity or the true negative rate), the coverage for positive examples and the coverage for negative examples, the learning time and the rule set size (see Section 2.9).

Table 2 shows the results of the considered classification algorithms for the concept *Is the vehicle driving safely?* (see Fig. 6). Together with the results we present a standard deviation of the obtained results.

One can see that accuracy of algorithm ARS for the decision class *NO* is higher than the accuracy of the algorithm RS for analyzed data set. The decision

Table 2. Results of experiments for the concept: *Is the vehicle driving safely?*

Decision class	Method	Accuracy	Coverage	Real accuracy
YES	RS	0.977 ± 0.001	0.948 ± 0.003	0.926 ± 0.003
	ARS	0.967 ± 0.001	0.948 ± 0.003	0.918 ± 0.003
NO	RS	0.618 ± 0.031	0.707 ± 0.010	0.436 ± 0.021
	ARS	0.954 ± 0.016	0.733 ± 0.018	0.699 ± 0.020
All classes	RS	0.963 ± 0.001	0.935 ± 0.003	0.901 ± 0.003
(YES + NO)	ARS	0.967 ± 0.001	0.937 ± 0.004	0.906 ± 0.004

Table 3. Learning time and the rule set size for concept: *Is the vehicle driving safely?*

Method	Learning time	Rule set size
RS	488 ± 21 seconds	975 ± 28
ARS	33 ± 1 second	174 ± 3

class NO is smaller that the class YES. It represents atypical cases in whose recognition we are most interested in (dangerous driving a vehicle on a highway).

Table 3 shows the learning time and the number of decision rules induced for the considered classifiers. In the case of the algorithm ARS we present the average number of decision rules over all concepts from the relationship diagram (see Fig. 6).

One can see that the learning time for ARS is much shorter than for RS and the average number of decision rules (over all concepts from the relationship diagram) for ARS algorithm is much lower than the number of decision rules induced for RS.

The experiments showed that classification quality obtained through classifiers based on AR-schemes is higher than classification quality obtained through traditional classifiers based on decision rules (especially in the case of the class NO). Apart from that the time spent on classifier construction based on AR-schemes is shorter than when constructing classical rule classifiers. Also, the structure of a single rule classifier (inside the ARS classifier) is less complicated than the structure of RS classifier (a considerably smaller average number of decision rules). It is worth noticing that the the performance of the ARS classifier is much more stable than the RS classifier because of the differences in data in samples supplied for learning (*e.g.*, to change the simulation scenario).

6 Behavioral Pattern Identification

An efficient complex dynamical systems monitoring very often requires the identification of the so-called *behavioral patterns* or a specific type of such patterns called *high-risk patterns* or *emergent patterns* (see, *e.g.*, [93, 99, 100, 132, 138, 139, 140, 141, 142, 143, 144, 173, 174, 175, 176]). They are complex concepts concerning dynamic properties of complex objects, dependent on time and space and expressed in a natural language. Examples of behavioral patterns may be *overtaking on a road*, *behavior of a patient faced with a serious life threat*, *ineffective behavior of robot team*. These types of concepts are much more difficult to approximate than complex concepts whose approximation does not require following object changes over time and may be defined for unstructured or structured objects. Identification of some behavioral patterns can be very important for recognition or prediction of behavior of a complex dynamical system, *e.g.*, some behavioral patterns correspond to undesirable behaviors of complex objects. In this case we call such behavioral patterns as *risk patterns* and we need some tools for identifying them. If in the current situation some risk patterns are identified, then the control object (a driver of the vehicle, a medicine doctor, a pilot of the aircraft, etc.) can use this information to adjust selected parameters to obtain the desirable behavior of the complex dynamical system. This can make it possible to overcome dangerous or uncomfortable situations. For example, if some behavior of a vehicle that cause a danger on the road is identified, we can try to change its behavior by using some suitable means such as road traffic signalling, radio message or police patrol intervention. Another

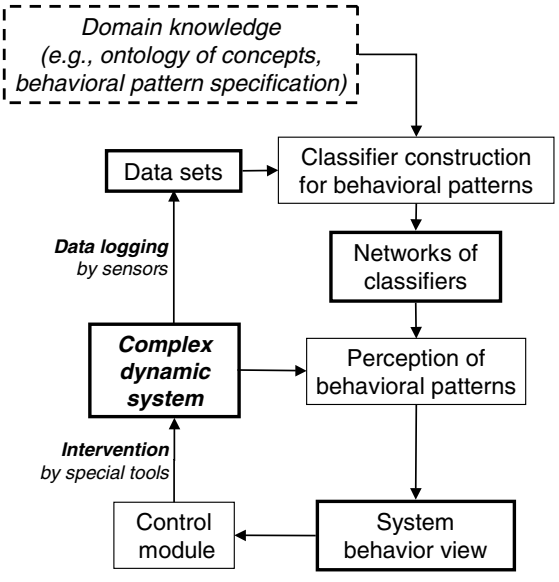


Fig. 25. Complex dynamical systems monitoring using behavioral patterns

example can be taken from medical practice. A very important element of the treatment of the infants with respiratory failure is appropriate assessment of the risk of death. The appropriate assessment of this risk leads to the decision of particular method and level of treatment. Therefore, if some complex behavior of an infant that causes a danger of death is identified, we can try to change its behavior by using some other methods of treatment (may be more radical) in order to avoid the infant's death (see Section 6.26).

In the Fig. 25 a scheme of complex dynamical system monitoring with the help of behavioral patterns is presented. This monitoring takes place in the following way. At the beginning, as a result of complex dynamical system observation, there are registered data sets describing changing over time parameter values of complex objects occurring in the system under observation. For a given complex dynamic system domain knowledge is gathered concerning, among others, complex behaviors of objects occurring in this system. Next, classifier nets are constructed on the basis of this knowledge and gathered data sets which enable perception of these patterns' behaviors whose detection is crucial for the correct functioning of the complex dynamical system. Identification of such patterns enable to find out important facts about the current system situation. This knowledge may be used by a control module which may perform a sequence of intervening actions aiming at restoring or maintaining the system in a safe, correct or convenient condition. Moreover, during complex dynamical systems monitoring data sets may still be collected. On the basis of these data sets a classifier structure identifying behavioral patterns is updated. This enable a certain type of adaptation of applied classifiers.

In this section a methodology of complex object's behavior monitoring is proposed which is to be used for approximating behavioral patterns on the basis of data sets and domain knowledge.

6.1 Temporal Information System

The prediction of behavioral patterns of a complex object evaluated over time is usually based on some historical knowledge representation used to store information about changes in relevant futures or parameters. This information is usually represented as a data set and has to be collected during long-term observation of a complex dynamical system (see, *e.g.*, [173, 174, 175, 176, 316, 318]). For example, in the case of road traffic, we associate the object-vehicle parameters with the readouts of different measuring devices or technical equipment placed inside the vehicle or in the outside environment (*e.g.*, alongside the road, in a helicopter observing the situation on the road, in a traffic patrol vehicle). Many monitoring devices serve as informative sensors such as Global Positioning System (GPS), laser scanners, thermometers, range finders, digital cameras, radar, image and sound converters (see, *e.g.*, [97, 153]). Hence, many vehicle features serve as models of physical sensors. Here are some exemplary sensors: location, speed, current acceleration or deceleration, visibility, humidity (slipperiness) of the road. By analogy to this example, many features of complex objects are often dubbed sensors. It is worth mentioning, that in the case of the treatment of infants with respiratory failure, we associate the object parameters (sensors) mainly with values of arterial blood gases measurements and the X-ray lung examination.

Data sets used for complex object information storage occurring in a given complex dynamical system may be represented using information systems (see, *e.g.*, [318]). This representation is based on representing individual complex objects by object (rows) of information system and information system attributes represent the properties of these objects. Because in a complex dynamical system there may occur many different complex objects, the storing of information about individual complex object identifiers is necessary. This information may be represented by the distinguished information system attribute which we mark by a_{id}. For convenience of the further discussion (see Algorithm 6.2) we assume that the set of values of the a_{id} attribute is linearly ordered. Therefore, the a_{id} attribute must be enriched by the relation ordering the set of values of this attribute in a linear order. Apart from that, it should be remembered that the complex objects occurring in complex dynamical systems change over time and their properties (object states) should be registered at different time instants (in other words time points). Hence, it is also necessary to store together with the information about a given object an identifier of time in which these properties are registered. This information may also be represented by the distinguished information system attribute which we mark as a_t. Because we assume that the identifiers of a time point are linearly ordered, then attribute a_t must be enriched by a relation ordering the set of values of this attribute in a linear order.

Hence, in order to represent complex object states observed in complex dynamical systems, the standard concept of information system requires extension. Therefore, we define *a temporal information system* [318].

Definition 19 (*A temporal information system*)**.**

1. *A temporal information system is a six-element tuple:*

$$\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t}), \text{ where:}$$

   (a) (U, A) *is an information system,*
   (b) a_{id}, a_t *are distinguished attributes from the set* A*,*
   (c) $\leq_{a_{id}}$ *is a relation of linear order on the set* $V_{a_{id}}$*,*
   (d) $\leq_{a_t}$ *is a relation of linear order on the set* V_{a_t}*.*
2. *About an object* $u \in U$ *we say that it represents the current parameters of the complex object with identifier* $a_{id}(u)$ *at time point* $a_t(u)$ *in the temporal information system* $\mathbf{T}$*.*
3. *About an object* $u_1 \in U$ *we say that it precedes an object* $u_2 \in U$ *in the temporal information system* $\mathbf{T}$ *if and only if*

$$u_1 \neq u_2 \ \wedge \ a_{id}(u_1) = a_{id}(u_2) \ \wedge \ a_t(u_1) \leq_{a_t} a_t(u_2).$$

4. *About an object* $u_2 \in U$ *we say that it follows an object* $u_1 \in U$ *in the temporal information system* $\mathbf{T}$ *if and only if* u_1 *precedes* u_2*.*
5. *About an object* $u \in U$ *we say that it is situated between objects* $u_1, u_2 \in U$ *in the temporal information system* $\mathbf{T}$ *if and only if* u_1 *precedes* u *and* u *precedes* u_2*.*

A typical example of a temporal information system is an information system whose objects represent vehicles' states at different instants of their observation.

Example 18. Let us have temporal information system $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ whose objects represent vehicles' states at different instants of their observation. Attributes from the set A describe sensor parameters of the vehicle at individual instants (*e.g.*, speed, location, lane, etc.). The distinguished a_{id} attribute is a unique number identifier of each vehicle, registered in the system $\mathbf{T}$. The a_t attribute represents the number of time units (*e.g.*, seconds) which have elapsed since the starting moment of all vehicles' observation. However, the relations $\leq_{a_{id}}$ and $\leq_{a_t}$ are common relations $\leq$ on the set of natural numbers. □

6.2 Representing Spatial Properties Using Concepts

In the presented approach the first step to identify the behavior of complex objects changing over time is representing and recognizing spatial properties of complex objects, that is, those which concern a certain chosen time point and their recognition does not require observing complex object changes over time. One of the most popular ways to represent these properties is representing them

using concepts. Each concept introduces the partitioning the sets of objects into two classes, that is, the class of these objects which belong to the concept and at the same time satisfy the property connected with the concept and the class of objects not belonging to the concept and at the same time not satisfying the property connected with the concept. If complex objects changing over time are represented using temporal information systems, then the concepts representing these objects' properties may be defined using attributes available in this system. The language of defining such concepts may be for example the language $GDL(\mathbf{T})$ (see Definition 5) where $\mathbf{T}$ is a temporal information system. Using this language spatial properties of complex objects may be observed at single time points. They may be for example, such concepts as: *low vehicle speed*, *high body temperature*, *considerable turn or dangerous inclination of a robot*.

Another language allowing to define properties of complex objects is *a language of elementary changes of object parameters* using information about how at a given time instant the values of the elementary parameters of the complex object changed in relation to the previous observation of this object (see [88, 173, 174, 175, 176, 178]). This property defining language is very useful when we wish to observe complex object parameters in relation to their previous observation. Examples of such properties may be: *increasing or decreasing the speed of the vehicle*, *moving the vehicle towards the right lane*, *the increasing the patient's body temperature*.

However, the use of the two above mentioned languages for defining properties of complex objects is possible only when the concepts being defined can be defined using formulas which use attribute values representing the current or previous value of the complex object's parameter. In practice, formulas of this type may be defined by experts on the basis of domain knowledge. However, an expert is often not able to give such an accurate definition of the concept. For example, the concept expressed using an expert's statement that *the vehicle speed is low* is difficult to be described without additional clues using a formula of the language GDL based on sensor attributes, although intuitively the dependence of this concept on the sensor attributes does not raise any doubt. Similarly, an expert's statement that *the patient's body temperature has fallen slightly since the last observation* is difficult to be formally described without additional clues, using the language of elementary changes of the complex object parameters (in this case the complex object is a treated patient). Meanwhile, in everyday life we often use such statements. Therefore, in the general case describing concepts concerning complex object properties requires the approximation of these concepts with the help of classifiers. This approximation may take place on the basis of a decision table whose conditional attributes are attribute arrangement from a given information system, while the decision attribute values given by the expert (on the basis of domain knowledge) describe the membership of the objects of the table under construction to the concept being approximated. The classifier constructed for such a table allow to test the membership of any object to the concept being approximated.

6.3 Temporal Information System Based on Concepts

If we decide to represent complex object properties using concepts, then a specific type of temporal information system is necessary which we call *a temporal information system based on concepts*.

Definition 20 (*A temporal information systems based on concepts*)**.** *A temporal information system* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *we call a temporal information system based on concepts (c-temporal information system or a c-system), if all attributes from the set* A *apart from the* a_{id} *and* a_t *attributes are attributes representing concepts determined in the set of objects* U*.*

Each attribute of c-system (apart from the a_{id} and a_t attributes) is then a binary attribute (taking two values). In this paper, we assume that they are 1 and 0 values, with 1 symbolizing the membership of the objects to the concept and 0 symbolizing the membership of the object to the concept complement.

Data sets gathered for complex dynamical systems and represented using temporal information systems usually contain continuous attributes, that is, ones with a large number of values which we often associate with different sensor indications. Therefore, if we wish to use c-systems for learning complex behavior of objects changing over time, then at the beginning of the learning process a c-system must be constructed on the basis of the available temporal information system. In order to do this a family of concepts must be defined which replaces all attributes (apart from the a_{id} and a_t attributes) of the original temporal information system. It is also necessary to construct a family of classifiers which approximate concepts from the defined family of concepts. These classifiers serve as replacements of the attributes of the input system with the c-system attributes. Such an operation are called the *c-transformation*.

Definition 21. *Let us assume that:*

- $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *is a temporal information system,*
- $C_1, ..., C_k$ *is a family of concepts defined on* U*,*
- $\mu_1, ..., \mu_k$ *are a family of classifiers approximating concepts* $C_1, ..., C_k$ *based on the chosen attributes from set* $A \setminus \{a_{id}, a_t\}$*.*

1. *An operation of changing the system* $\mathbf{T}$ *to a c-system*

$$\mathbf{T}_c = (U, A_c, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$$

   is called a c-transformation of the system $\mathbf{T}$*, if* $A_c = \{a_{id}, a_t, c_1, ..., c_k\}$ *and for any* $u \in U : c_i(u) = \mu_i(u)$*, for* $i = 1, ..., k$*;*
2. *The C-system* $\mathbf{T}_c$ *is called a result of c-transformation of the system* $\mathbf{T}$*.*

Now, we present the c-transformation algorithm for the temporal information system (see Algorithm 6.1). Performance of this algorithm is based on constructing a new information system which apart from attributes a_{id} and a_t has all the attributes based on the previously defined concepts.

Algorithm 6.1. C-transformation

Input:

1. temporal information system $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ such that $U = \{u_1, ..., u_n\}$,
2. family of concepts $C_1, ..., C_k$ defined in the set U,
3. family of classifiers $\mu_1, ..., \mu_k$ approximating concepts $C_1, ..., C_k$ on the basis of attributes from the set $A \setminus \{a_{id}, a_t\}$.

Output: The C-system $\mathbf{T}_c = (U, A_c, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ such that $A_c = \{a_{id}, a_t, c_1, ..., c_k\}$, where attributes $c_1, ..., c_k$ represent concepts $C_1, ..., C_k$

```
1  begin
2      Create an empty information system T_c which has attributes
           a_id, a_t, c_1, ..., c_k where attributes a_id and a_t are of the identical
           type as their counterparts in system T and attributes c_1, ..., c_k are
           binary attributes // T_c is without any objects for the
           time being
3      for i := 1 to n do
4          Create an empty list of values L.
5          Add a_id(u_i) to the list L.
6          Add a_t(u_i) to the list L.
7          for j = 1 to k do
8              Add μ_j(u_i) to the list L.
9          end
10         Add new object represented by values from L to the system T_c.
11     end
12     return T_c
13 end
```

With the assumption that each of the classifiers $\mu_1, ..., \mu_k$ can classify an object within the time of order $O(C)$, where C is a certain constant, then the time complexity of the above algorithm is of order $O(n \cdot k)$, where $n = card(U)$ and k is the number of concepts used for constructing the attributes.

Example 19. Let us take into consideration the temporal information system such as the one in Example 18. In such a system there may occur many continuous attributes like: *the speed of the vehicle*, *the location of the vehicle with regard to the crossroads*, *the location of the vehicle with regard to the left and right lane*, *visibility* and others. Therefore, this system, before being used in order to approximate temporal concepts, requires c-transformation which has to be executed on the basis of the established concepts. Also, classifiers for these concepts which were constructed earlier with the use of attributes from a given system are necessary. They may be, for example, the following concepts:

1. low (average, high) vehicle speed (approximation using the attribute: speed),
2. increasing (decreasing, maintaining) the vehicle speed in relation to the previous time point (approximation using attributes: speed and speed in the previous time point),
3. high (average, low) distance from the crossroads (approximation using the attribute: distance from the crossroads),
4. driving in the right (left) lane (approximation using the attribute: the location of the vehicle with regard to the left and right lane),
5. small movement of the vehicle towards the left (right) lane (approximation using attributes: the location of the vehicle with regard to the left and right lane and the location of the vehicle with regard to the left and right lane in the previous time point),
6. location of the vehicle at the crossroads (symbolic attribute moved from the initial system),
7. good (moderate, bad) visibility on the road,
8. high humidity (low humidity, lack of humidity) of the road.

After performing the c-transformation, the temporal information system from Example 18 is already a c-temporal information system, that is, apart from a_{id} and a_t attributes all of its attributes are binary ones representing concepts. □

Let us notice that the concepts applied during the c-transformation are usually constructed using discretization of chosen continuous attributes of a given information system performed manually by the expert. Obviously, this discretization may also be performed with the use of automatic methods (see Section 2.2).

6.4 Time Windows

The concepts concerning properties of unstructured complex objects at the current time point in relation to the previous time point are a way of representing very simple behaviors of the objects. However, the perception of more complex types of behavior requires the examination of behavior of complex objects over a longer period of time. This period is usually called the time window (see, *e.g.*, [173, 174, 175, 176, 316, 318]), which is to be understood as a sequence of objects of a given temporal information system registered for the established complex object starting from the established time point over the established period of time or as long as the expected number of time points are obtained. Therefore, learning to recognize complex types of behavior of complex objects with use of gathered data as well as the further use of learned classifiers to identify the types of behavior of complex objects, requires working out of the mechanisms of extraction of time windows from the data and their properties. That is, why we need the language of extraction of time windows from the c-system which we are about to define.

Definition 22 (*A language for extracting time windows*). *Let* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *be a c-temporal information system and let* $\mathbb{Z}_2$ *be the set of integer numbers equal or greater than 2. A language for extracting time windows from*

system $\mathbf{T}$ *(denoted by* $ETW(\mathbf{T})$ *or* ETW*-language, when* $\mathbf{T}$ *is fixed) is defined in the following way:*

- *the set* $AL_{ETW}(\mathbf{T}) = V_{a_{id}} \cup V_{a_t} \cup \mathbb{Z}_2 \cup \{",\"\}$ *is called an alphabet of the language* $ETW(\mathbf{T})$*,*
- *the set of atomic formulas of the language* $ETW(\mathbf{T})$ *is defined as a set of three-element tuples in the following form:* (i, b, s)*, where* $i \in V_{a_{id}}$*,* $b \in V_{a_t}$ *and* $s \in \mathbb{Z}_2$*.*

Now, we determine the semantics of the language $ETW(\mathbf{T})$. The language $ETW(\mathbf{T})$ formulas may be treated as the descriptions of object sequences occurring one after another in system $\mathbf{T}$.

Definition 23. *Let* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *be a c-temporal information system. A satisfiability of an atomic formula* $\phi = (i, b, s) \in ETW(\mathbf{T})$ *by an object* $u \in U$ *from* $\mathbf{T}$ *(denoted by* $u \models_{ETW(\mathbf{T})} \phi$*), is defined in the following way:*

$$u \models_{ETW(\mathbf{T})} (i, b, s) \Leftrightarrow$$

$$a_{id}(u) = i \ \wedge \ card(\{x \in U : x \text{ precedes } u \wedge b \leq_{a_t} a_t(x)\}) < s.$$

Let us notice that an object $u \in U$ satisfies a formula $\phi = (i, b, s) \in ETW(\mathbf{T})$ iff the following two conditions are satisfied:

1. the identifier of the object u is equal i,
2. the number of objects registered since b to $a_t(u)$ is less than s.

Formulas of the language ETW describe sets of objects which we call *time windows*.

Definition 24 (*A time window***).** *Let* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *be a c-temporal information system.*

1. *A time window in the c-temporal information system* $\mathbf{T}$ *is a set* $|\phi|_{ETW(\mathbf{T})}$*, where* $\phi \in ETW(\mathbf{T})$*.*
2. *The family of all time windows from the c-temporal information system* $\mathbf{T}$ *is denoted by* $TW(\mathbf{T})$*.*
3. *If* $W \in TW(\mathbf{T})$ *then the number* $card(W)$ *is called a length of time window* W *and is denoted by* $Length(W)$*.*
4. *The family of all time windows from the c-temporal information system* $\mathbf{T}$ *with length equals to* s *is denoted by* $TW(\mathbf{T}, s)$*.*

Because according to the definition of semantics of the language $ETW(\mathbf{T})$ the elements of each time window $W \in TW(\mathbf{T}, s)$ are linearly ordered by relation $\leq_{a_t}$, then each time window may be treated as an ordered sequence $W = (u_1, ..., u_s)$ of objects from set U. Additionally each i-th object of time window W we mark with $W[i]$, where $i \in \{1, ..., s\}$.

Here is an example of extraction of the time window from the c-temporal information system.

Example 20. Let us consider c-system $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ whose objects represent the states of vehicles at different time points. The attributes from set A describe concepts representing sensor properties of vehicle parameters at individual points (*e.g.*, high velocity, small acceleration, etc.). The distinguished attribute a_{id} is a unique identifier of each vehicle and attribute a_t represents the observation time registered in a given object of system $\mathbf{T}$. To simplify matters let us assume that the values of attributes a_t and a_{id} are natural numbers. Let us consider the vehicle with identifier 5 for which one hundred time points have been registered in the system from the time point with identifier 11 to the time point with identifier 109. For this vehicle we could, for example, isolate a time window defined by formula $(5, 20, 31)$ which represents the behavior of the vehicle from the time point marked 20 to the time point marked 50. □

6.5 Temporal Concepts

More complex types of behavior of complex objects may be defined using time widows over complex concepts which we call temporal concepts. We assume that temporal concepts are specified by a human expert. Temporal concepts are usually used in queries about the status of some objects in a particular time window. Answers to such queries can be of the form Yes, No or *Does not concern*. For example, in the case of road traffic one can define complex concepts such as *Is a vehicle accelerating in the right lane?*, *Is a vehicle speed stable while changing lanes?*, or *Is the speed of a vehicle in the left lane stable?*.

Intuitively, each temporal concept (defined on the time window) depends on object properties observed at some time points. At the same time we mean both spatial properties, that is, properties registering the spatial value of the complex object parameter observed at the time point, *e.g.*, *the left driving lane of the vehicle*, *high speed of the vehicle*, *low the patient's body temperature*) as well as the properties describing elementary changes of complex object parameters in relation to the previous observation of this object (*e.g.*, *increasing or decreasing the speed of the vehicle*, *small move of the vehicle towards the right lane*, *the increasing the patient's body temperature*). Such simple concepts we call *elementary concepts*. Usually it is possible to provide the ontology which shows a dependence between a temporal concept and some elementary concepts. For example, the temporal concept *accelerating and changing lanes from right to left* depends on such elementary concepts as *high speed*, *low speed*, *increasing speed*, *decreasing speed*, *small move of the vehicle towards the left lane*.

6.6 Temporal Patterns

It would seem that temporal concepts as concepts on the higher hierarchical level of ontology may be approximated using elementary concepts which are on the lower ontology level (see Section 6.5). It is sufficient to build a concept approximation table for the approximated concept. However, during the construction of such a table we encounter a serious problem resulting from the difference in meaning (semantical difference) of objects being examples and counterexamples

of concepts on both ontology levels. Therefore, concepts on the lower ontology level are defined for time points, while temporal concepts on the higher ontology level are determined on time windows. In other words the observations of objects at time points are examples and counterexamples for concepts on the lower ontology level. However, the objects which are examples and counterexamples for temporal concepts are the observations of complex objects registered over time windows, that is, sequences of object observations from time points. For this reason we cannot apply here the construction method of conditional attributes from Section 5.1 which is based on the language PEC, since that method required for the concepts existing on both ontology levels to concern the same type of objects.

That is why to define the attributes which approximate temporal concepts we need to introduce a different language which can make it possible to transfer the spatial properties of complex objects registered at time points onto the property level of complex objects over the time window. In this paper, for this purpose we propose *a language for definnig features of time windows.*

Definition 25 (*A language for definnig features of time windows***).** *Let* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *be a c-temporal information system. A language for definnig features of time windows of c-temporal information system* $\mathbf{T}$ *(denoted by* $FTW(\mathbf{T})$ *or FTW-language, when* $\mathbf{T}$ *is fixed) is defined in the following way:*

- *the set* $AL_{FTW}(\mathbf{T}) = (A \setminus \{a_{id}, a_t\}) \cup \{ExistsPoint, EachPoint, MajorityPoints, MinorityPoints, FirstPoint, LastPoint, OrderPoints\} \cup \{\neg, \vee, \wedge\}$ *is an alphabet of the language* $FTW(\mathbf{T})$,
- *for any* $a, b \in A \setminus \{a_{id}, a_t\}$ *expressions of the form* $ExistsPoint(a)$, $EachPoint(a)$, $MajorityPoints(a)$, $MinorityPoints(a)$, $FirstPoint(a)$, $LastPoint(a)$, $OrderPoints(a, b)$ *are atomic formulas of the language* $FTW(\mathbf{T})$.

Now, we determine the semantics of the language $FTW(\mathbf{T})$. The formulas of the language $FTW(\mathbf{T})$ may be treated as the descriptions of time windows in system $\mathbf{T}$. For example, the formula $ExistsPoint(a)$ is interpreted as the description of all those time windows of system $\mathbf{T}$ in which such an object u has been observed that $a(u) = 1$. Thus, we observed an object belonging to the concept represented by attribute a. Time windows may be described by different formulas, however, for formula ϕ to have sense in system $\mathbf{T}$, that is, to be semantically correct in the language $FTW(\mathbf{T})$, there has to exist at least one time window which is described by formula ϕ. For such a window we say that it *satisfies* formula ϕ.

Definition 26. *Let* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *be a c-information system and let* s *be a length of time windows. The satisfiability of an atomic formula* $\phi \in FTW(\mathbf{T})$ *by a time window* $W \in TW(\mathbf{T}, s)$ *(denoted by* $W \models_{FTW(\mathbf{T})} \phi$*), is defined in the following way:*

1. $W \models_{FTW(\mathbf{T})} ExistsPoint(a) \Leftrightarrow$ *exists* $u \in W$ *such that* $a(u) = 1$,
2. $W \models_{FTW(\mathbf{T})} EachPoint(a) \Leftrightarrow$ *for any* $u \in W$ *is satisfied* $a(u) = 1$,

3. $W \models_{FTW(\mathbf{T})} MajorityPoints(a) \Leftrightarrow$

$$card(\{u \in W : a(u) = 1\}) > \lfloor (Length(W) - 1)/2 \rfloor,$$

4. $W \models_{FTW(\mathbf{T})} MinorityPoints(a) \Leftrightarrow$

$$card(\{u \in W : a(u) = 1\}) < \lceil (Length(W) - 1)/2 \rceil,$$

5. $W \models_{FTW(\mathbf{T})} FirstPoint(a) \Leftrightarrow a(W[1]) = 1$,
6. $W \models_{FTW(\mathbf{T})} LastPoint(a) \Leftrightarrow a(W[s]) = 1$,
7. $W \models_{FTW(\mathbf{T})} OrderPoints(a, b) \Leftrightarrow$ *exist* $i, j \in \{1, ..., s\}$ *such that:*

$$i < j \ \wedge \ a(W[i]) = 1 \ \wedge \ b(W[j]) = 1.$$

It is worth noticing that the formulas of the language $FTW(\mathbf{T})$ may not only be satisfied by time windows from the set $TW(\mathbf{T})$ but also by time windows of the set $TW(\mathbf{T}')$ where $\mathbf{T}'$ is a temporal information system with an extended set of objects in relation to system $\mathbf{T}$.

Below we present several examples of formulas of the language FTW.

- If attribute a stores information about membership to the concept of *low speed*, then formula $EachPoint(a)$ describes a time window in which the vehicle's speed is low all the time.
- If attribute a stores information about membership to the concept of *accelerating*, then formula $ExistsPoint(a)$ describes time windows in which the vehicle happened to accelerate.
- If attribute a_1 stores information about membership to the concept of *accelerating* and attribute a_2 stores information about membership to the concept of *driving in the right lane*, then formula $ExistsPoint(a_1) \wedge EachPoint(a_2)$ describes the time window in which the vehicle happened to accelerate and the whole time drive in the right lane.

It is worthwhile mentioning that the language FTW defined above should be treated as an exemplary language for defining features of time windows, which has been used in experiments related to this paper. Obviously, it is possible to define many other languages of this type.

The FTW language formulas can be used to define patterns describing the properties of time windows, therefore, we call them *temporal patterns*.

Definition 27 (*A temporal pattern*)**.** *Let* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *be a c-temporal information system. Any formula of the language* $FTW(\mathbf{T})$ *is called a temporal pattern of the system* $\mathbf{T}$.

Temporal patterns are often used in queries with binary answers such as *Yes* or *No*. For example, in the case of road traffic we have exemplary temporal patterns such as *Did vehicle speed increase in the time window?*, *Was the speed stable in the time window?*, *Did the speed increase before a move to the left lane occurred?* or *Did the speed increase before a speed decrease occurred?*.

We assume that any temporal pattern ought to be defined by a human expert using domain knowledge accumulated for the given complex dynamical system.

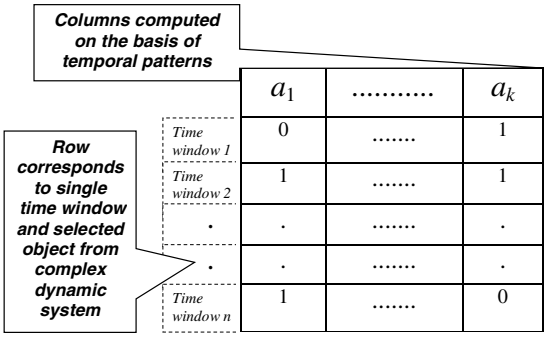


Fig. 26. The scheme of an information system of time windows

6.7 Information System of Time Windows

The properties of the accessible time windows could be represented in a special information system which is called an information system of time windows (see also Fig. 26).

Definition 28 (*An information system of time windows*)**.** *Let us assume that:*

- $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *is a c-temporal information system,*
- s *is a fixed length of time windows such that* $1 < s \leq card(U)$*,*
- $\Phi = \{\phi_1, ..., \phi_k\} \subseteq FTW(\mathbf{T})$ *is a family of temporal patterns defined by experts (sub-language of the language* $FTW(\mathbf{T})$*),*
- $\mathcal{P}_{FTW} = (\overline{U}, \Phi, \models_{FTW(\mathbf{T})})$ *is a property system, where* $\overline{U} = TW(\mathbf{T}, s)$*.*

The information system $\overline{\mathbf{T}} = (\overline{U}, \overline{A})$ *defined by the property system* $\mathcal{P}_{FTW}$ *is called an information system of time windows (TW-information system).*

Apparently, construction system $\overline{\mathbf{T}}$ requires generating the family of all time windows of established duration. Therefore, below we present the algorithm for generating all time windows of established duration (length) from a given c-system (see Algorithm 6.2).

On account of sorting objects in system $\mathbf{T}$, pessimistic time complexity of Algorithm 6.2 is of order $O(n \cdot log\ n)$, where n is the number of objects in the system $\mathbf{T}$.

Example 21. For the c-temporal information system from Example 19 an information system of time windows may be constructed. In order to do this, we may use for example temporal patterns chosen from the following collection of patterns:

1. low (moderate, high) vehicle speed at the first point (at the last point, at any point, at all points, at the minority of points, at the majority of points) of the time window,
2. low (moderate, high) maximal (minimal, average) speed of vehicle at the time window,

Algorithm 6.2. Generating all time windows

Input:
- temporal information system $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ such that $U = \{u_1, ..., u_n\}$,
- fixed length s of time windows such that $1 < s \leq n$.
- relation of linear order $\leq_{\{a_{id},a_t\}}$ defined on $U \times U$ in the following way:

$$\forall (u_1, u_2) \in U \times U : u_1 \leq_{\{a_{id},a_t\}} u_2 \Leftrightarrow u_1 \leq_{a_{id}} u_2 \wedge u_1 \leq_{a_t} u_2.$$

Output: The set of time windows $TW(\mathbf{T}, s)$

```
1  begin
2      Create empty list TW of windows
3      Sort set U using relation ≤_{a_id,a_t}
4      Create empty list window of objects from the set U
5      currentID := a_id(u_1)
6      Insert u_1 to the list window
7      for i := 2 to n do
8          if (a_id(u_i) = currentID) then
9              if (Length(window) < s) then
10                 Add u_i to the end of the list window
11             else
12                 Remove the first object from the list window
13                 Add u_i to the end of the list window
14                 Add window to family TW
15             end
16         else
17             Clear the list window
18             currentID := a_id(u_i)
19             Insert u_i to the list window
20         end
21     end
22     return TW
23 end
```

3. increasing (decreasing, maintaining) the speed of the vehicle (at the last point, at any point, at all points, at the minority of points, at the majority of points) of the time window,
4. low speed (moderate speed, high speed, increasing speed, decreasing speed, maintaining speed) in the time window before high speed (moderate speed, low speed, increasing speed, decreasing speed, maintaining speed) in the further part of the time window,
5. low (moderate, high) distance of the vehicle from the crossroads at the first point (at the last point, at any point, at all points, at the minority of points, at the majority of points) of the time window,

6. driving in the right (left) lane at the first point (at the last point, at any point, at all points, at the minority of points, at the majority of points) of the time window,
7. a slight turn towards the left (right) lane at the first point (at the last point, at any point, at all points, at the minority of points, at the majority of points) of the time window,
8. the location of the vehicle at the crossroads at the first point (at the last point, at any point, at all points, at the minority of points, at the majority of points) of the time window,
9. good (moderate, bad) visibility at the first point (at the last point, at any point, at all points, at the minority of points, at the majority of points) of the time window,
10. high humidity (low humidity, lack of humidity) of the road at the first point (at the last point, at any point, at all points, at the minority of points, at the majority of points) of the temporal.

It is easy to notice that each of the above patterns may be expressed in language FTW. □

The choice of specific patterns for the construction of the information system of time windows should depend on the concept which is to be approximated with the help of this system, obviously after performing the grouping (clustering) of objects (see Section 6.9).

6.8 Clustering Time Windows

The properties of time windows expressed by temporal patterns could be used for approximating temporal concepts which express more complex properties of time windows. However, it often happens that the objects of the information system of time windows (that is, system $\overline{\mathbf{T}}$) are not yet sufficient to use their properties for approximating temporal concepts. It is so due to the fact that there are too many of those objects and the descriptions of time windows which they represent are too detailed. Hence, if they are used for approximating temporal concepts, then the extension of the created classifier would be too small, which means that the classifier could classify too small number of tested objects.

Therefore, in this paper we use clustering (grouping) such objects which leads to obtaining a family of object clusters (clusters of time windows). From the general view point the grouping objects is always done using the language chosen by an expert and it is based on the fact that the clusters of objects are represented using formulas (patterns) defined in the language of grouping. Thanks to those patterns not only the objects of a given system are grouped but it is also possible to examine membership of other (tested) objects to individual clusters. Namely, we may say about the tested object that it belongs to the cluster when it satisfies the pattern describing this cluster.

In this paper, we propose the language $NL(\overline{\mathbf{T}})$ (neighborhood language) (see Definition 8) for grouping objects of system $\overline{\mathbf{T}} = (\overline{U}, \overline{A})$. The application of this language for object grouping requires defining the two following elements:

1. the dissimilarity function $DISM_{\overline{\mathbf{T}}}$ of object pairs in the information system $\overline{\mathbf{T}}$ (see Definition 7),
2. the family of formulas included in the set $NL(\overline{\mathbf{T}})$ which defines clusters of objects in system $\overline{\mathbf{T}}$ (see Definition 8).

We define the dissimilarity function using a dissimilarity classifier $\mu_{DISM_{\overline{\mathbf{T}}}}$ approximating it which is constructed for the dissimilarity table carefully specified by the expert.

However, defining the family of formulas included in $NL(\overline{\mathbf{T}})$, which defines clusters of objects in system $\overline{\mathbf{T}}$, requires the introduction of a subset of objects of system $\overline{\mathbf{T}}$ which are centers (generators) of clusters being created and the sequence of radiuses corresponding to them and limiting the clusters. Each atomic formula of the language $NL(\overline{\mathbf{T}})$ is, therefore, expression (u, ε) (where $u \in \overline{U}$ and $\varepsilon \in (0, 1]$), and at the same time such a formula encompasses all the objects for which the value of the dissimilarity function $DISM_{\overline{\mathbf{T}}}$ in relation to object u does not exceed value ε. Therefore, meanings of such formulas are in a sense neighborhoods of the objects membership to $\overline{U}$.

Algorithm 6.3. Clustering objects from an information system of time windows (version 1)

Input:
- information system of time windows $\overline{\mathbf{T}}=(\overline{U}, \overline{A})$ such that $\overline{U}=\{u_1, \ldots, u_n\}$,
- dissimilarity function $DISM_{\overline{\mathbf{T}}}$ of object pairs in the system $\overline{\mathbf{T}}$,
- deviation from standards ε.

Output: The family of formulas $F \subseteq NL(\overline{\mathbf{T}})$ defining clusters in the system $\overline{\mathbf{T}}$

```
1  begin
2      F := ∅
3      for i := 1 to n do
4          Compute a neighborhood N(u_i) = {u ∈ Ū : DISM_T̄(u_i, u) ≤ ε}
5      end
6      Sort the set Ū in descending order // By sizes of neighborhoods
           computed in the previous step
7      repeat
8          Take object u ∈ Ū such that its neighborhood is maximal
9          F := F ∪ (u, ε)
10         Ū := Ū \ N(u)
11     until Ū ≠ ∅
12     return F
13 end
```

The choice of the centers of those neighborhoods in order to construct formulas defining clusters should not be made at random. Objects being the centers of neighborhoods are called *standards* whereas the radiuses of neighborhoods *deviations* from the standards (see, *e.g.*, [89]). Both the standards and the deviations from them could be provided by experts. However, if this is for some reason difficult, there could be applied methods of determining standards and the deviations from them known from literature. It should be noted that in contemporary literature methods of this kind are often called *granulation methods* and the neighborhoods are called *granules* (see, *e.g.*, [89, 90, 91, 317, 319, 320, 321, 322, 323]).

In this paper, we propose the following clustering algorithm for automatic generation of object clusters, which is very similar to the algorithm presented in paper [165]. This algorithm is a greedy algorithm which initially chooses the object which has the biggest neighborhood and removes from the set being covered objects belonging to this neighborhood, thus choosing another neighborhood until it covers the whole set of objects (see Algorithm 6.3).

On the account of calculation of the neighborhoods for all the objects from set $\overline{U}$, the computational time complexity of Algorithm 6.3 is of order $O(n^2)$. In the case of bigger tables it may hinder or even make it impossible to effectively use this algorithm. Therefore, we also present a random version of the above algorithm (see Algorithm 6.4 and [165]).

As it can be observed, Algorithm 6.4 is in practice significantly faster in relation to Algorithm 6.3 because it does not require determining neighborhoods for all the objects from set $\overline{U}$, but only for the objects chosen randomly from set $\overline{U}$.

Algorithm 6.4. Clustering objects from an information system of time windows (version 2)

Input:
- information system of time windows $\overline{\mathbf{T}} = (\overline{U}, \overline{A})$ such that $\overline{U} = \{u_1, ..., u_n\}$,
- dissimilarity function $DISM_{\overline{\mathbf{T}}}$ of object pairs in the system $\overline{\mathbf{T}}$,
- deviation from standards ε.

Output: The family of formulas $F \subseteq NL(\overline{\mathbf{T}})$ defining clusters in the system $\overline{\mathbf{T}}$

```
1  begin
2      F := ∅
3      repeat
4          Randomly select u ∈ Ū
5          Compute a neighborhood N(u) = {v ∈ Ū : DIST_T̄(u, v) ≤ ε}
6          F := F ∪ (u, ε)
7          Ū := Ū \ N(u)
8      until Ū ≠ ∅
9      return F
10 end
```

From the formal viewpoint, clusters of time windows are defined using the language $ECTW$.

Definition 29 (*A language for extracting clusters of time windows*)**.** *Let us assume that:*

- $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *is a c-temporal information system,*
- $\overline{\mathbf{T}} = (\overline{U}, \overline{A})$ *is information system of time windows for the system* $\mathbf{T}$.

Any neighborhood language $NL(\overline{\mathbf{T}})$ *is called a language for extracting clusters of time windows from system* $\mathbf{T}$ *(denoted by* $ECTW(\mathbf{T})$ *or* $ECTW$*-language, when* $\mathbf{T}$ *is fixed).*

The formulas of the language $ECTW$ describe clusters of time windows from the temporal information system.

Definition 30 (*A cluster of time windows*)**.** *Let* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *be a c-temporal information system. The meaning any formula* $\phi \in ECTW(\mathbf{T})$ *is called a cluster of time windows from the system* $\mathbf{T}$.

In the example below we illustrate how the dissimilarity function may be defined for the time windows grouping.

Example 22. For the information system of time windows obtained in Example 21, a dissimilarity function may be constructed on the basis of expert knowledge. In order to do this, the value of the dissimilarity function for a certain (the most representative set of object pairs of this system) should be obtained from the experts. To make it happen, the expert should know what temporal concept is approximated using clusters obtained with the help of currently defined dissimilarity function (see Section 6.9). Let us assume that it is the concept *accelerating in the right lane*. In terms of this concept the expert may immediately define several vehicle groups which behave very similarly, that is, vehicles being in one of such groups, in terms of the dissimilarity function being constructed, should not differ too much. For example, they may be the following vehicle groups:

A. vehicles which accelerate in a given time window but they do it with a smaller or bigger intensity,
B. vehicles which accelerate in a given time window, however, they do not do it in a continuous manner, sometimes maintaining stable speed but they never decrease it,
C. vehicles which in a given time window increase, decrease or at times maintain stable speed,
D. vehicles which decrease speed in a given time window but they do it with a smaller or bigger intensity.

In Table 4, values of chosen attributes (temporal patterns) are presented for objects u_1, u_2, u_3, u_4, and u_5 of a certain information system of time windows.

Table 4. Exemplary objects of some TW-information system

No	**Temporal patterns**	u_1	u_2	u_3	u_4	u_5
1.	High speed of the vehicle at the first point of the time window	0	0	0	0	1
2.	Moderate speed of the vehicle at the first point of the time window	0	0	0	1	0
3.	Low speed of the vehicle at the first point of the time window	1	1	1	0	0
4.	Increasing the speed of the vehicle at all points of the time window	1	1	0	0	0
5.	Increasing the speed of the vehicle at some point of the time window	1	1	1	1	0
6.	Maintaining stable speed of the vehicle at all points of the time window	0	0	0	0	0
7.	Maintaining stable speed of the vehicle at some time window point	0	0	1	1	0
8.	Decreasing speed of the vehicle at all points of the time window	0	0	0	0	1
9.	Decreasing speed of the vehicle at some point of the time window	0	0	0	1	1
10.	High speed of the vehicle at the last point of the time window	0	1	0	1	0
11.	Moderate speed of the vehicle at the last point of the time window	1	0	1	0	0
12.	Low speed of the vehicle at the last point of the time window	0	0	0	0	1

It is easy to notice that object u_1 belongs to the group of vehicles A and does not belong to the groups B, C and D. Similarly, object u_2, which is presented in the table, also belongs to group A. The reason for this is the fact that both objects increased the speed the whole time, but object u_2 at the end of the time window reached a high speed and object u_1 only moderate speed. That is why, the dissimilarity function value for these two objects is low, that is, 0.1, whereas object u_3 belongs to group B, for it maintained stable speed throughout a part of the window. Therefore, the dissimilarity function between objects u_2 and u_3 is bigger and equals 0.25. Object u_4 belongs to group C and therefore the difference between this object and object u_1 is 0.5. Finally, the dissimilarity function value between objects u_2 and u_5 (u_5 belongs to group D) is the biggest and is as big as 1.0, for in terms of the concept *accelerating in the right lane* these objects differ to the maximum because object u_1 is increasing speed and object u_5 is decreasing speed all the time. The dissimilarity function values proposed for all object pairs are compared in Table 5. Finally, we notice that objects u_1, u_3, u_4 and u_5 may be treated as standards for which the deviation is 0.1. □

Table 5. Values of dissimilarity function for pairs of time windows

	u_1	u_2	u_3	u_4	u_5
u_1	0	0.1	0.25	0.5	1.0
u_2	0.1	0	0.25	0.5	1.0
u_3	0.25	0.25	0	0.25	0.75
u_4	0.5	0.5	0.25	0	0.5
u_5	1.0	1.0	0.75	0.5	0

If there is a family of standards given along with their deviations, then on their basis we can construct a family of formulas of the language $NL(\overline{\mathbf{T}})$ which represent the established clusters in system $\overline{\mathbf{T}}$. Now, in order to construct a new information system which represents the clusters' properties we need to define the language defining the cluster features.

Definition 31 (*A language for defining features of clusters of time windows*)**.** *Let* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *be a c-temporal information system and* $\overline{\mathbf{T}} = (\overline{U}, \overline{A})$ *be an information system of time windows for the system* $\mathbf{T}$*. A language for defining features of clusters of time windows for the system* $\mathbf{T}$ *(denoted by* $FCTW(\overline{\mathbf{T}})$ *or* $FCTW$*-language, when* $\overline{\mathbf{T}}$ *is fixed) is defined in the following way:*

- *the set* $AL_{FCTW}(\mathbf{T}) = AL_{FTW}(\mathbf{T}) \cup \{ExistsWindow, EachWindow, MajorityWindows, MinorityWindows\}$ *is an alphabet of the language* $FCTW(\overline{\mathbf{T}})$*,*
- *for any* $\alpha, \beta \in FCTW(\overline{\mathbf{T}})$ *expressions of the form:* $ExistsWindow(\alpha)$*,* $EachWindow(\alpha)$*,* $MajorityWindows(\alpha)$*,* $MinorityWindows(\alpha)$ *are atomic formulas of the language* $FCTW(\overline{\mathbf{T}})$*.*

Now, we define the semantics of the language $FCTW(\overline{\mathbf{T}})$. The formulas of the language $FCTW(\overline{\mathbf{T}})$ may be treated as the descriptions of families of time windows in system $\mathbf{T}$. For example, formula $ExistsWindow(\alpha)$ is interpreted as the description of all those families of time windows of system $\mathbf{T}$ in which there exists at least one such window that satisfies formula α.

Definition 32. *Let us assume that:*

- $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *is a c-temporal information system,*
- $\overline{\mathbf{T}} = (\overline{U}, \overline{A})$ *is a information system of time windows for the system* $\mathbf{T}$*.*

The satisfiability of an atomic formula $\alpha \in FCTW(\overline{\mathbf{T}})$ *by a family of time windows* $\mathcal{W} = \{W_1, ..., W_k\} \subseteq TW(\mathbf{T})$ *(denoted by* $\mathcal{W} \models_{FCTW(\mathbf{T})} \alpha$*), is defined in the following way:*

1. $\mathcal{W} \models_{FCTW(\mathbf{T})} ExistsWindow(\alpha) \Leftrightarrow \exists_{W \in \mathcal{W}}\ W \models_{FTW(\mathbf{T})} \alpha$*,*
2. $\mathcal{W} \models_{FCTW(\mathbf{T})} EachWindow(\alpha)) \Leftrightarrow \forall_{W \in \mathcal{W}}\ W \models_{FTW(\mathbf{T})} \alpha$*,*
3. $\mathcal{W} \models_{FCTW(\mathbf{T})} MajorityWindows(\alpha)) \Leftrightarrow$

$$card(\{W \in \mathcal{W} : W \models_{FTW(\mathbf{T})} \alpha\}) > \lfloor (k-1)/2 \rfloor,$$

4. $\mathcal{W} \models_{FCTW(\mathbf{T})} MinorityWindows(\alpha)) \Leftrightarrow$

$$card(\{W \in \mathcal{W} : W \models_{FTW(\mathbf{T})} \alpha\}) < \lceil (k-1)/2 \rceil.$$

Below we present several examples of formulas of the language $FCTW$.

- If attribute a_1 of system $\mathbf{T}$ stores information about membership to the concept of *accelerating*, then formula $ExistsWindow(LastPoint(a_1))$ describes such clusters of time windows where there is a window in which acceleration occurred at the last point of this window.
- If attribute a_1 of system $\mathbf{T}$ stores information about membership to the concept of *accelerating*, then formula $MajorityWindows(EachPoint(a_1))$ describes such clusters of time windows that in the majority of them the speed is being increased all the time.
- If attributes a_1 and a_2 of system $\mathbf{T}$ store information about membership respectively to the concepts of *accelerating* and *decelerating*, then formula $\neg ExistsWindow(ExistsPoint(a_2)) \wedge EachWindow(ExistsPoint(a_1))$ describes such clusters of time windows in which there does not exist a single window in which deceleration occurred and in all the windows of this cluster the speed is increased.

The patterns of the language $FCTW$ could be utilized to define the properties of clusters of time windows. Due to this clusters of time windows are represented by information systems which we call an information system of clusters of time windows.

Definition 33 (*An information system of clusters of time windows*)**.** *Let us assume that:*

- $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *is a c-temporal information system,*
- $\overline{\mathbf{T}} = (\overline{U}, \overline{A})$ *is a information system of time windows for the system* $\mathbf{T}$*.*
- $\psi_1, ..., \psi_n \in ECTW(\mathbf{T})$ *is a defined by experts family of temporal patterns corresponding to clusters of time windows* $CL_1, ..., CL_n$*,*
- $\Phi = \{\phi_1, ..., \phi_m\} \in FCTW(\mathbf{T})$ *is a defined by experts family of features of clusters of time windows,*
- $\mathcal{P}_{FCTW} = (\overline{\overline{U}}, \Phi, \models_{FCTW(\mathbf{T})})$ *is a property system, where* $\overline{U} = \{CL_1, ..., CL_n\}$*.*

The information system $\overline{\overline{\mathbf{T}}} = (\overline{\overline{U}}, \overline{\overline{A}})$ *defined be the property system* $\mathcal{P}_{FCTW}$ *is called an information system of clusters of time windows (CTW-information system).*

It is easy to see that construction of the system $\overline{\overline{\mathbf{T}}}$ requires generating clusters of time windows in such a way that it could be possible to check the ability to satisfy formulas of the language $FCTW(\overline{\mathbf{T}})$. Such clusters may be generated by the linear overview of set $\overline{U}$ and assigning the objects of this set to individual clusters. The complexity of such an algorithm would be of order $O(l \cdot n)$, where n is the number of clusters and $l = card(\overline{U})$.

Example 23. Using the dissimilarity function from Example 22, the objects of the information system of time windows from Example 21 may be clustered. However, in order to construct an information system for the determined family of clusters, patterns should be established. They may be the following:

1. in each time window of a cluster there occurs speed increase the whole time (the pattern describes the properties of vehicles from group A from Example 22),
2. in each time window of a cluster there occurs speed increase (the pattern describes the properties of vehicles from groups A, B, C from Example 22),
3. in the majority of time windows of the cluster there does not occur speed increase,
4. in the cluster there exists a time window in which there occurs speed decrease (the pattern describes the properties of vehicles from groups C and D from Example 22),
5. in each time window of the cluster at the first time point of the window the speed is low and at the last point of the window it is high or moderate,
6. in most time windows of the cluster the vehicles drive in the right lane the whole time. □

The choice of specific patterns to construct the information system of clusters of time windows depends on the temporal concept which is approximated with the help of this system (see Section 6.9).

6.9 Temporal Concept Table

The properties of clusters of time windows expressed by formulas of the language *FCTW* may be used for constructing decision tables which enable the identification of temporal concepts. For this purpose, it is necessary to add to system $\overline{\overline{\mathbf{T}}}$ a decision attribute which characterizes the cluster's membership to the established temporal concept. In this way, we obtain a decision table which is called the temporal concept table (see Fig. 27).

Definition 34 (*A temporal concept table*)**.** *Let us assume that:*

- $\overline{\overline{\mathbf{T}}} = (\overline{\overline{U}}, \overline{\overline{A}})$ *is a information system of clusters of time windows,*
- C *is concept defined in the set* $\overline{\overline{U}}$*.*

A temporal concept table for the concept C *is a decision table* $\overline{\overline{\mathbf{T}}}_C = (\overline{\overline{U}}, \overline{\overline{A}} \cup \{d_C\})$*, where attribute* d_C *is a membership function of the concept* C*.*

A question arises, how it is possible that an expert can propose the value of decision attribute d_C. To make a decision concerning the value of this attribute an expert has at his disposal the values of attributes from set $\overline{\overline{A}}$. These are the attributes proposed earlier as the characteristic features of clusters of time windows. The expert could therefore propose them in such a way that he or she is now able to use them successively in order to determine the membership of

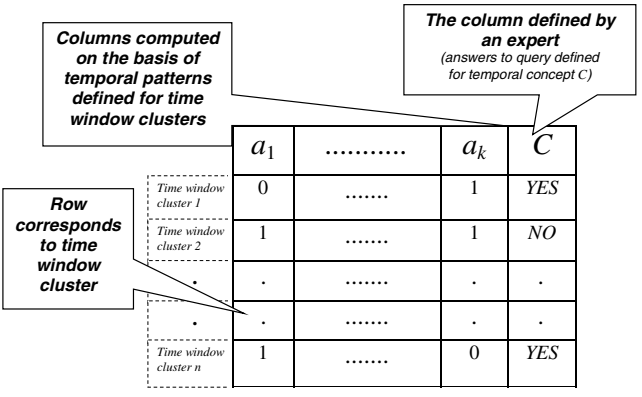


Fig. 27. The scheme of a temporal concept table

the whole clusters to concept C. In other words the properties of clusters have to be defined in such a way that they could serve to determine the time window clusters' membership to temporal concept C.

Example 24. The information system obtained on the basis of patterns such as in Example 23 may be enriched by a decision attribute which describes the membership of individual clusters of time windows to the established temporal concept (*e.g.*, the concept of *accelerating in the right lane*). In this way we obtain a decision table which may serve to the approximation of this concept. □

If there is given set H of linearly ordered layer labels of concept C, then for table $\overline{\overline{\mathbf{T}}}_C$ there could be constructed a stratifying classifier μ_C^H which can indicate, for each time window cluster, the layer of concept C to which this cluster belongs.

6.10 Classification of Time Windows

In practical applications we encounter a question whether the behavior of the complex objects observed in the time window belongs to a given temporal concept defined for clusters of time windows. It means the classification of time windows to the concept determined on the set of clusters of time windows. Meanwhile, the classifier constructed for table $\overline{\overline{\mathbf{T}}}_C$ can classify window clusters and not single time windows. Therefore, before we use such a classifier, it should be checked to which cluster of time windows the tested time window belongs. That is how the algorithm of time window classification works (see Algorithm 6.5). However, we assume that during execution of the Algorithm 6.5 the following elements are available (established or computed earlier):

- a training c-temporal information system $\mathbf{T}$,
- a fixed length l of time windows,
- a family of temporal patterns $\phi_1, ..., \phi_m \in FTW(\mathbf{T})$,
- a system $\overline{\mathbf{T}} = (\overline{U}, \overline{A})$ such that attributes from the set $\overline{A}$ correspond to formulas $\phi_1, ..., \phi_m$,

- a dissimilarity function $DISM_{\overline{\mathbf{T}}}$ of object pairs from the system $\overline{\mathbf{T}}$, approximated using the stratifying classifier $\mu_{DISM_{\overline{\mathbf{T}}}}$,
- a family of temporal patterns $\psi_1, ..., \psi_n \in ECTW(\mathbf{T})$ such that $\psi_i = (std_i, \varepsilon_i)$, for $i = 1, ..., n$,
- a system $\overline{\overline{\mathbf{T}}} = (\overline{U}, \overline{A})$ such that objects from the set $\overline{U}$ correspond to clusters defined by formulas $\psi_1, ..., \psi_n$,
- a concept C defined in the set $\overline{U}$,
- a decision table $\overline{\overline{\mathbf{T}}}_C = (\overline{\overline{U}}, \overline{A} \cup \{d_C\})$ with decision attribute representing membership of objects from the set $\overline{\overline{U}}$ to the concept C,
- a linearly ordered set $(H, \leq_H)$ such that H is a set of layer labels of the concept C and $\leq_H$ is a relation of linear order on the set H,
- a stratifying classifier μ_C^H constructed for the concept C on the basis the table $\overline{\overline{\mathbf{T}}}_C$,
- a test c-temporal information system $\mathbf{T}_{TS}$.

Because the Algorithm 6.5 classifies time windows to one layer of concept C, it is the classifier stratifying temporal concept C.

Assuming that all operations of the examination of formulas satisfiability, the computation of the dissimilarity function values and application of the classifier μ_C^H (occurring in the above algorithm) are executed at the constant time, then

Algorithm 6.5. Time window classification ($ClassifyWindow$)

Input: A test time window $W \in TW(\mathbf{T}_{TS}, l)$
Output: The layer of concept C

```
1  Procedure ClassifyWindow(W)
2  begin
3      Create empty list row
4      for φ_1, ..., φ_m do
5          if W ⊨_{FTW(T)} φ_i then
6              Add '1' to the end of the list row
7          else
8              Add '0' to the end of the list row
9          end
10     end
11     Create new object u_row of the system T̄ on the basis values of
       attributes from the list row.
12     Select a formula ψ = (std, ε) ∈ {ψ_1, ..., ψ_n) such that:
13       1. u_row ⊨_{ECTW(T)} ψ and
14       2. DISM_T̄(u_row, std) = min_{i∈{1,...,n}}{DISM_T̄(u_row, std_i)}
15     Select object u_ψ ∈ Ū corresponding to the formula ψ
16     return μ_C^H(u_ψ)
17 end
```

the time complexity of the above algorithm is of order $O(m+n)$, where m is the number of temporal patterns for time windows used and n is the number of time window clusters.

6.11 Behavioral Graphs

Temporal concepts defined for objects from a complex dynamical system can be treated as nodes of a graph called a *behavioral graph*, where connections between nodes represent temporal dependencies. Such connections between nodes can be defined by an expert or read from a data set that has been accumulated for a given complex dynamical system.

Definition 35 (*A behavioral graph*)**.**

1. *A behavioral graph G is an ordered pair (V, E), where V is a nonempty and finite set of nodes (temporal concepts) and E is a set of directed edges $E \subset V \times V$ (connections represent temporal dependencies between temporal concepts).*
2. *A temporal path in the behavioral graph $G = (V, E)$ is a sequence of nodes $v_1, ..., v_l$ such that for any $i \in \{1, ..., l-1\}$ an edge $(v_i, v_{i+1}) \in E$. A number l is called a length of temporal path $v_1, ..., v_l$.*
3. *The family of all temporal paths with the length l ($l > 0$) in the behavioral graph G is denoted by $PATH(G, l)$.*

Bellow, we present an example of behavioral graph.

Example 25. Fig. 28 presents an example of behavioral graph for a single object-vehicle exhibiting a behavioral pattern of vehicle while driving on a road. In this behavioral graph, for example, connections between node *Acceleration on the right lane* and node *Acceleration and changing lanes lanes from right to left* indicates that after an acceleration in the right lane, a vehicle can change to the left lane (maintaining its acceleration during both time windows). □

In addition, a behavioral graph can be constructed for different kinds of objects such as single vehicles or groups of vehicles and defined for behaviors such as driving on the strength road, driving through crossroads, overtaking, and passing. Therefore, we consider any behavioral graph as a model for behavioral patterns (see Section 6.23).

6.12 Representing Spatial Properties of Structured Objects Using Concepts

If we wish to observe the behavior of structured objects changing over time, then it turns out that observing the behaviors of individual parts of these objects separately is not sufficient. It happens that way because if we observe the behavior of a certain structured object we have to consider the issue what relations there are between the types of behaviors of individual parts of this object and how these relations coexist and change over time. For example, if we observe the

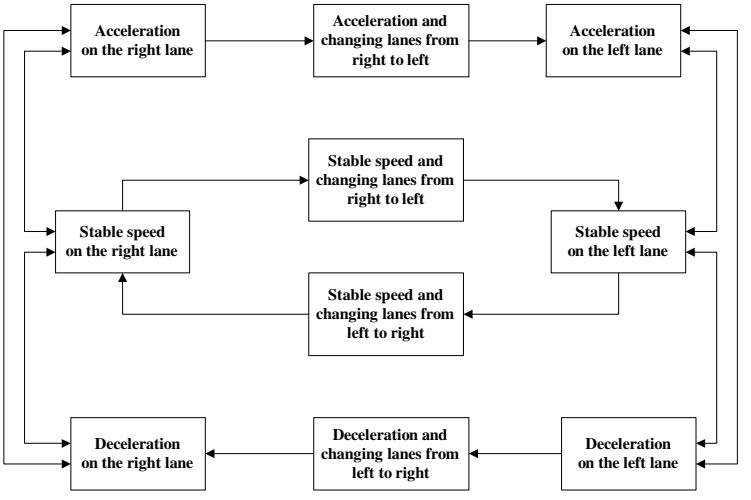


Fig. 28. A behavioral graph for a single object-vehicle

overtaking maneuver made by the structured object, which is a pair of vehicles (the overtaking and overtaken vehicles), then we are interested in the relations between the behavior of the overtaking vehicle and the behavior of the overtaken vehicle. Another example may concern the observation of the courses of illnesses, which being in the interaction with one another, develop in a given patient.

The spatial properties of such bounds may be represented using concepts which concern the sets of structured object parts. Such concepts represent the partition of all structured objects into those which belong to the concept and those structured objects which do not belong to the concept being concerned. Examples of such concepts may be concepts concerning the distance between two chosen vehicles belonging to the group of vehicles being examined (*e.g.*, short distance between two vehicles, long distance between two vehicles, driving on the same lane).

Similarly to the concepts representing spatial properties of unstructured objects, the concepts representing spatial properties of structured objects also may be approximated. The approximation may take place on the basis of appropriately constructed for this purpose decision table. Each object of such a table corresponds to a certain structured object. Conditional attributes are the arrangement of attributes from a given temporal information system registering the parameters of individual parts of a given structured object. However, the decision attribute of this table describes the membership of objects of the table under construction to the approximated concept concerning the spatial property of the whole structured object and its values are suggested by the expert on the basis of domain knowledge. The classifier constructed for such a table allows testing any structured object for the membership to the approximated concept. Due to the fact that the approximated concept is spatial (it does not require following changes over time with the exception of parameter changes since the

last observation of the structured object), we can apply here classical classifiers, stratifying classifiers as well as classifiers based on AR-schemes. However, during the construction of the decision table for the purpose of approximation of spatial concepts for structured objects we encounter a serious problem concerning extraction of relevant context for the parts of structured objects. One of the solutions to this problem could be application of *a sweeping algorithm around the complex object* which is characterized in the following subsection.

6.13 Sweeping Algorithm around the Complex Object

In the paper, each structured object occurring in a complex dynamical system is understood as an object consisting of parts, that is, objects of lesser complexity which are linked with one another by relations describing the context in which individual parts of a structured object occur. Therefore, both learning concepts concerning structured objects and testing structured objects for membership to such concepts requires a method of isolating structured objects under examination. Unfortunately, the elementary approach to isolation of structured objects which consists in analyzing all the subsets (of established number) from the existing set of potentially parts of structured objects cannot be used because of the high computational complexity of algorithms generating and examining such subsets. For example, if we examine a system which has 20 complex objects (*e.g.*, 20 vehicles on the road) and we are interested in structured objects defined as groups of 6 objects (6 vehicles as in the case of examining the dangerous overtaking maneuver) (see Fig. 55) then the general number of groups which require observation is equal to the number of 6-element combinations from the 20-element set, that is, $\binom{20}{6} = 38760$. Additionally, if the observation is carried out only over 100 time units, then the temporal information system describing the properties of all such groups of vehicles would have to have almost 4 million rows!

Another possibility is the application of methods which use the context in which the objects being parts of structured objects occur. This type of methods isolates structured objects not by direct indication of the set of parts of the searched structured object but by establishing one part of the searched structured object and attaching to it other parts, being in the context to the established part. Unfortunately, also here, the elementary approach to determining the context of the part of the structured object, consisting in examining all possible subsets (of established number) of the set of potential structured objects to which the established part of the structured object belongs, cannot be applied because of a great number of such subsets. For example, in order to identify a group of vehicles which are involved in a dangerous maneuver and to which the established vehicle belongs (to which we pay attention), it would be necessary to follow (in real time) the behavior of all possible groups of vehicles of the established number (*e.g.*, six vehicles) to which the established vehicle belongs, which is, with a relatively small number of visible vehicles, still too difficult.

Hence, we need special methods of determining the context of the established part of the structured object based on domain knowledge, which allows to limit the number of analyzed sets of parts of structured objects. Therefore, in this

paper we propose a special method which we call *the sweeping method*. The method works in the following way: at the stage of learning the behavior for structured objects only those structured objects are taken into account that came into being through the so-called sweeping around of all the complex objects which may be part of a structured object, and at the same time for each such object there is only one structured object constructed. The activity of sweeping is carried out by the special sweeping algorithm which returns to the established complex object an ordered list of these objects which are significant from the point of view of the type of examined complex behavior of structured objects. The organization of this list is important due to the fact that in the obtained structured object represented by the list, each object (a part of a structured object) can have its role in the examined complex behavior of structured objects.

Definition 36 (*A sweeping algorithm around the complex object*)**.** *Let* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *be a temporal information system such that objects from the set* U *are unstructured objects of a fixed type* T*. Each algorithm* SA *which at the input receives a chosen object* $u \in U$*, and at the output returns the subset of* U *such that the object* u *belongs to this subset represented in the form of organized* k*-element list* $SA(u)$ *(where* $k > 1$*) is called a sweeping algorithm around the complex object.*

The sweeping algorithms must be constructed individually on the basis of domain knowledge for each complex behavior which is to be identified. By this we mean such complex types of behavior as: overtaking vehicles on the road, driving of a group of vehicles in a traffic jam, chasing one car after another, persistence of respiratory insufficiency in patients. Also, the parameter k should be fixed individually for a given sweeping algorithm.

As a result of applying the sweeping algorithm we only obtain as many structured objects as many complex objects constituting potential parts of structured objects there are, for each of the established structured objects gets attached to one unstructured object (one of its parts) which plays a specific role in the structured object. For example, if we examine the behavior of a group of vehicles connected with the overtaking maneuver, then the vehicle distinguished during the sweeping algorithm's activity may be the vehicle which overtakes. In this group also other vehicles may be distinguished (the overtaken vehicle, the oncoming vehicle, the vehicle driving in the back, etc.) which takes place when we apply other sweeping algorithms (constructed for this type of objects).

We present an example of the sweeping algorithm for the purpose of recognizing the overtaking maneuver, where the distinguished overtaking vehicle is the complex object (see Algorithm 6.6). This algorithm regards the situation presented in the Fig. 29. The group of vehicles which are returned by Algorithm 6.6 which is the so called *sweeping zone* comprises 6 vehicles which are the most important from the point of view of planning and performing the overtaking maneuver by a given vehicle (see Appendix A).

Let us notice that in the Algorithm 6.6 a number of auxiliary concepts are used. They are for example such concepts as: *going close to* u, *going in the right lane*, *going in front of vehicle* u, *going behind vehicle* u, *going in the same*

Algorithm 6.6. The sweeping algorithm for the purpose of recognizing the overtaking maneuver

Input:
- temporal information system $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ such that any object $u \in U$ is an object of the fixed type T (a single vehicle),
- object $u \in U$

Output: The sweeping zone around u

begin
- Create empty list L.
- Insert object (vehicle) u to the list L.
- If there is a vehicle in the right lane close behind to vehicle u and going in the same direction, add it to list L as vehicle BR (see Fig. 29).
- If there is a vehicle in the left lane close behind to vehicle u, going in the same direction, add it to list L as vehicle BL (see Fig. 29).
- If there is a vehicle close in front of vehicle u in the right lane, going in the same direction, add it to list L as vehicle FR1 (see Fig. 29).
- If during the previous stage you added a vehicle to list L as vehicle $FR1$, then if in front of this vehicle in the right lane there is another vehicle going close in the same direction as vehicle u, then add it to list L as vehicle FR2 (see Fig. 29).
- If there is an oncoming vehicle in front of vehicle u in the left lane, then add it to list L as vehicle FL (see Fig. 29).
- **return** L as $SA(u)$.

end

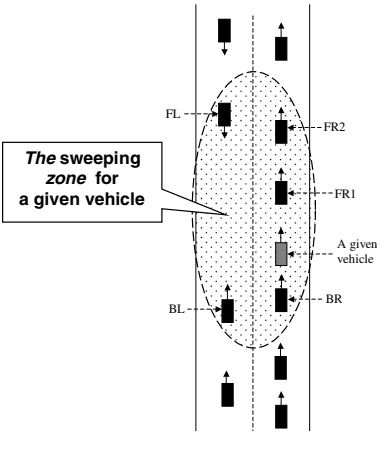


Fig. 29. A given vehicle and its sweeping zone

direction as u, *oncoming vehicle*. All these concepts require approximation, but they are spatial concepts and therefore they are much easier to be approximated than the spatio-temporal concepts. It also means that in the above algorithm the application of classifiers approximating these concepts is necessary.

One should, however, be aware of the fact that each sweeping algorithm works on the basis of the sweeping heuristics defined by an expert. Therefore, such an algorithm isolates only structured objects suggested by this heuristics. That is why, it is only a very selective perception and its application requires a great support from the experts who have to provide all crucial sweeping heuristics, in terms of perception of the whole complex dynamical system.

During experiments with the road simulator (see Section 6.25) the sweeping algorithm already worked at the stage of generating data using the simulator. Therefore, in the data set there is already available information about structured objects consisting of 6 vehicles connected with the identification of the overtaking maneuver. Whereas, in experiments with medical data (see Section 6.26), where a group of illnesses is the structured object, the performance of the sweeping algorithm is based on constructing only such groups of illnesses which occurred in particular patients from the data at the same time.

6.14 C-temporal Information System for Structured Objects

The sweeping algorithm, defined in Section 6.13, efficiently extracts structured objects. For objects extracted in such a way we may define concepts which may be approximated using attributes of a given temporal information system. Thanks to that c-temporal information systems may be constructed whose attributes describe spatial relation properties between parts of structured objects. However, because the attributes of such c-system are to concern relation properties between structured objects extracted with the sweeping algorithm, it comes into being in a slightly different way than the standard c-system, that is, a special algorithm is needed which constructs c-system on the basis of the available temporal information system with the use of the sweeping algorithm. Such an algorithm we call a *cr-transformation*.

Definition 37. *Let us assume that:*

- $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *is a c-temporal information system such that any object* $u \in U$ *is an object of the fixed type* T,
- $C_1, ..., C_k$ *is a family of spatial concepts determined for structured objects (parts of such objects must be registered at the same time),*
- $\mu_1, ..., \mu_k$ *is a family of classifiers which approximate concepts* $C_1, ..., C_k$ *on the basis of the parameters of the structured object parts established on the basis of attributes from set* $A \setminus \{a_{id}, a_t\}$,
- SA *is a sweeping algorithm around objects from the system* $\mathbf{T}$.

1. *An operation of changing the system* $\mathbf{T}$ *to the c-system*

$$\mathbf{T}_r = (U_r, A_r, c_{id}, \leq_{c_{id}}, c_t, \leq_{c_t})$$

is called a cr-transformation of the system $\mathbf{T}$ *iff*

- $U_r = \{g_1, ..., g_n\}$, *where* $g_i = SA(u_i)$ *is the result returned by the sweeping algorithm* SA *for* u_i, *for* $i = 1, ..., n$,
- $A_r = \{c_{id}, c_t, c_1, ..., c_k\}$ *where attributes from the set* A_r *are defined in the following way:*
  - $\forall g_i \in U_r : c_{id}(g_i) = a_{id}(u_i) \ \wedge \ c_t(g_i) = a_t(u_i)$,
  - *attributes* $c_1, ..., c_k$ *represent concepts* $C_1, ..., C_k$ *where:*

$$\forall_{g \in U_r} \ c_i(g) = \mu_i(g) \text{ for } i = 1, ..., k.$$

2. *The c-system* $\mathbf{T}_r$ *is called a result of cr-transformation of the system* $\mathbf{T}$.

We present the algorithm of the cr-transformation for the temporal information system (see Algorithm 6.7).

Algorithm 6.7. Cr-transformation

Input:

1. $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ is a c-temporal information system such that any $u \in U$ is a concept object of the fixed type T,
2. $C_1, ..., C_k$ is a family of spatial concepts determined for structured objects (parts of such objects must be registered at the same time),
3. $\mu_1, ..., \mu_k$ is a family of classifiers which approximate concepts C_1, $..., C_k$ on the basis of the parameters of the structured object parts established on the basis of attributes from set $A \setminus \{a_{id}, a_t\}$,
4. SA is a sweeping algorithm around objects from the system $\mathbf{T}$.

Output: The c-information temporal system
$\mathbf{T}_r = (U_r, A_r, c_{id}, \leq_{c_{id}}, c_t, \leq_{c_t})$

```
1  begin
2  |  Create an empty information system T_r which has attributes
   |     c_id, c_t, c_1, ..., c_k where attributes c_id and c_t are of the identical type
   |     as their counterparts in system T and attributes c_1, ..., c_k are
   |     binary attributes // T_r is without any objects for the
   |     time being
3  |  for i := 1 to card(U) do
4  |  |  Compute a structured object g := SA(u_i).
5  |  |  Create an empty list of values L.
6  |  |  Add a_id(u_i) to the list L.
7  |  |  Add a_t(u_i) to the list L.
8  |  |  for j = 1 to k do
9  |  |  |  Add μ_j(g) to the list L.
10 |  |  end
11 |  |  Add new object represented by values from L to the system T_r.
12 |  end
13 |  return T_r
14 end
```

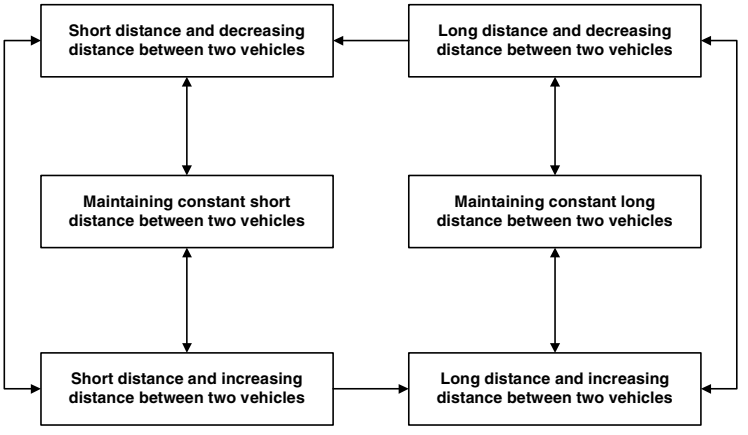


Fig. 30. A behavioral graph for relations between two vehicles exhibiting a changes of distance between vehicles while driving on a road

Assuming that each classifier $\mu_1, ..., \mu_k$ and algorithm SA work over the time of order $O(C)$, where C is a certain constant, then the time complexity of the Algorithm 6.7 is of order $O(n \cdot k)$, where $n = card(U)$ and k is the number of concepts used to construct attributes.

The result of the performance of the algorithm of the cr-transformation is the c-system and therefore it may be used to approximate temporal concepts describing the relation between the structured object's parts. Mechanisms of performing such approximation are the same as in the case of temporal concepts concerning unstructured complex objects. Finally, it leads to the behavioral graph describing complex objects being parts of structured objects with connection to the types of behavior of other parts of these structured objects.

Fig. 30 presents an example of behavioral graph for relations between two objects (vehicles) exhibiting a changes of distance between vehicles while driving on a road.

6.15 Sequences of Time Windows

Complex types of behavior of complex objects, treated as unstructured objects may be described using behavioral graphs of complex objects. As a result of temporal concept approximation classifiers may be obtained for all concepts occurring in behavioral graphs which enable the examination of objects' membership to the temporal concepts. Hence, behavioral graphs and classifiers for temporal concepts allow to follow the types of behavior of the complex objects over a longer period of time than the time window. This longer period we call *a sequence of time windows*. Therefore, the learning of the perception of the complex behavior of structured objects with the use of the gathered data and behavioral graphs constructed for all parts of structured objects of a given type, as well as the further use of the learned classifiers to identify the types of

behavior of structured objects, requires working out the mechanisms of the extraction of sequences of time windows. Therefore, we need a language for extraction of sequences of time windows.

Definition 38 (*A language for extracting sequences of time windows*)**.** *Let* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *be a c-temporal information system and let* $\mathbb{Z}_1$ *be the set of integer numbers equal or greater than 1. A language for extracting sequences of time windows from system* $\mathbf{T}$ *(denoted by* $ESTW(\mathbf{T})$ *or* $ESTW$*-language, when* $\mathbf{T}$ *is fixed) is defined in the following way:*

- *the set* $AL_{ESTW}(\mathbf{T}) = V_{a_{id}} \cup V_{a_t} \cup \mathbb{Z}_1 \cup \{\text{“,”}\}$ *is called an alphabet of language* $ESTW(\mathbf{T})$*,*
- *the set of atomic formulas of the language* $ESTW(\mathbf{T})$ *is defined as a set of four-element tuples in the following form:* (i, b, l, s)*, where* $i \in V_{a_{id}}$*,* $b \in V_{a_t}$ *and* l, s *are integer numbers such that* $l > 1$ *and* $s > 0$*.*

Now, we define a semantics of the language $ESTW(\mathbf{T})$. The formulas of the language $ESTW(\mathbf{T})$ are treated as descriptions of sequences of time windows occurring one after another in system $\mathbf{T}$, and at the same time each two neighboring windows of such a sequence overlap at the connecting points of these windows.

Definition 39. *Let* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *be a c-temporal information system and* l *is a length of time windows. The satisfiability of formula* $\phi = (i, b, l, s) \in ESTW(\mathbf{T})$ *(where* $i \in V_{a_{id}}$*,* $b \in V_{a_t}$ *and* l, s *are integer numbers such that* $l > 1$ *and* $s > 0$*) by a time window* $W = (u_1, ..., u_n) \in TW(\mathbf{T})$ *(denoted by* $W \models_{ESTW(\mathbf{T})} \phi$*), is defined in the following way:*

$W \models_{ESTW(\mathbf{T})} (i, b, l, s) \Leftrightarrow$

$$\forall_{j \in \{1,...,n\}}\ a_{id}(u_j) = i\ \wedge\ l = n\ \wedge p\ mod\ (l-1) = 0\ \wedge \frac{p}{l-1} < s$$

where $p = card(\{x \in U : x\ precedes\ u_1\ \wedge\ b \leq_{a_t} a_t(x)\})$.

Let us notice that a time window $W = (u_1, ..., u_n) \in TW(\mathbf{T})$ satisfies a formula $\phi = (i, b, l, s) \in ESTW(\mathbf{T})$ iff the following four conditions are satisfied:

1. the time window W describes parameters of the object with the identifier i,
2. the size of the time window W is equal l,
3. the integer number of time windows with the length l has been registered since b to $a_t(u_1)$, i.e., the number of time windows with the first point registered not earlier than b and preceding time point u_1 (in the sense of Definition 19) is divisible by $l - 1$,
4. the number of time windows with the length l and with the first time point registered since b to $a_t(u_1)$ is less than s.

The formulas of the language $ESTW$ describe sets of time windows which we call sequences of time windows.

Definition 40 (*A sequence of time windows*). *Let* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *be a c-temporal information system.*

1. *Each set of time windows* $|\phi|_{ESTW(\mathbf{T})}$ *which is a meaning of a certain formula* $\phi \in ESTW(\mathbf{T})$ *we call a sequence of time windows in c-temporal information system* $\mathbf{T}$.
2. *A family of all sequences of time windows of a given c-temporal information system* $\mathbf{T}$ *we denote* $SEQ(\mathbf{T})$.
3. *If* $S \in SEQ(\mathbf{T})$ *then a number* $card(S)$ *we call a length of the sequence of time windows* S *and it is denoted by* $Length(S)$.
4. *A family of all sequences of time windows of the length* s *of a given c-temporal information system* $\mathbf{T}$ *is denoted by* $SEQ(\mathbf{T}, s)$.
5. *A family of all sequences of time windows of the length* s *of a given c-temporal information system* $\mathbf{T}$ *that they are windows of the length* l *we denote* $SEQ(\mathbf{T}, l, s)$.
6. *Due to the definition of the language* $ESTW(\mathbf{T})$ *the elements of each sequence of time windows* $S \in SEQ(\mathbf{T}, l, s)$ *are linearly ordered by relation* $\leq_{a_t}$, *then each sequence of time windows sequence may be treated as an ordered time sequence* $S = (W_1, ..., W_s)$ *of time windows from the set* $TW(\mathbf{T}, l)$. *Additionally, each* i*-th time window of sequence* S *we denote by* $S[i]$, *where* $i \in \{1, ..., s\}$.
7. *Any sequence of time windows* $S' = (W_i, ..., W_j) \in SEQ(\mathbf{T}, j - i + 1)$ *created by removing from the sequence* $p = (W_1, ..., W_k) \in SEQ(\mathbf{T}, k)$ *time windows* $W_1, ..., W_{i-1}$ *and* $W_{j+1}, ..., W_k$, *where* $i, j \in \{1, ..., k\}$ *and* $i < j$, *is called a sub-sequence of the sequence* S *and is denoted by* $Subsequence(S, i, j)$.

Here is an example of extracting a sequence of time windows from the c-temporal information system.

Example 26. Let us consider system $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ whose objects represent states of vehicles at different time points. Attributes from set A describe sensor parameters of the vehicle at individual points (*e.g.*, velocity, acceleration, location, lane). The distinguished attribute a_{id} is a unique identifier of each vehicle and attribute a_t represents the observation time registered in a given object in system $\mathbf{T}$. Let us assume, for the sake of simplification, that attribute values a_t and a_{id} are natural numbers. Let us take as an example a vehicle marked with the identifier 5 for which a hundred time points are registered in system $\mathbf{T}$ from the time point marked with the identifier 1 to the time point marked with the identifier 100. For such a vehicle a sequence of time windows may be extracted and defined by formula $(5, 1, 4, 3)$ which represents three time windows: W_1, W_2, W_3 defined by formulas $(5, 1, 4)$, $(5, 4, 4)$, $(5, 7, 4)$ (see Fig. 31). □

In practical applications the partition of a time window is an important notion concerning the sequence of time windows. It concerns the situation in which we extract the sequence of time windows from a given time window.

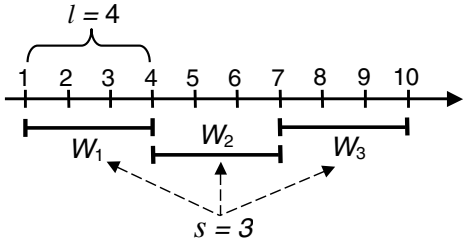


Fig. 31. A sequence of time windows

Definition 41 (*A partition of a time window*). *Let* $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *be a c-temporal information system and* $W \in TW(\mathbf{T}, m)$. *A family of time windows* $Partition(W, k) = \{W_1, ..., W_k\}$ *is called k-partition of time windows* W, *if the following conditions are satisfied:*

1. $\{W_1, ..., W_k\} \in SEQ(\mathbf{T}, l, k)$, *where* $k \cdot (l-1) + 1 = m$,
2. $W = \bigcup_{i \in \{1,...,k\}} W_i$.

According to the above definition for a given time window of duration m we can determine k-partition only when $k \cdot (l-1) + 1 = m$ that is $\frac{m-1}{k} = l - 1$, where l is the length of each of time windows of the obtained partition. Therefore, in order for us to be able to determine k-partition for a given time window of the length m, number m reduced by l has to be divisible by k, and the result of this division is the length of the window in the partition reduced by l. Going back to Example 26 let us notice that it is possible to determine k-partition for $k = 3$ for window $(5, 1, 10)$ (where $m = 10$), and this is the partition into 3 windows, each of length $l = 4$. They are windows: $(5, 1, 4)$, $(5, 4, 4)$ and $(5, 7, 4)$. On the other hand, for the same window $(5, 1, 10)$ we cannot determine k-partition for $k = 4$ because number $m - l = 9$ is not divisible by 4.

Let us notice that it is easy to construct an algorithm which for a given time window $W \in TW(\mathbf{T}, m)$ determines the partition $Partition(W, k)$ through linear overview of window W in order to create k-time windows of this partition.

6.16 Algorithm of Replacing a Sequence of Time Windows with the Sequence of Nodes of a Behavioral Graph

Each time window could be classified into a given temporal concept with the use of stratifying classifier. Each such concept corresponds to one node of the behavioral graph of complex object $\mathbf{G}$. Hence, each sequence of time windows may be replaced with the sequence of nodes of behavioral graph $\mathbf{G}$. The problem of classification conflict of a given time window consisting in the fact that a time window may be classified into many concepts, could be solved by choosing a concept for which classification certainty is the highest. In other words during classification the concept whose result layer of classification is possibly the

Algorithm 6.8. Replacing a sequence of time windows with the sequence of nodes of a behavioral graph ($MakeTimePath$)

Input: A sequence of time windows $S = \{W_1, ..., W_k\} \in SEQ(\mathbf{T}, l, k)$
Output: The path $P \in PATH(\mathbf{G})$ corresponding to the sequence S

```
1 Procedure MakeTimePath(S)
2 begin
3 |   Create empty list P
4 |   for i := 1 to k do
5 |   |   Select v_max ∈ V such that ∀v ∈ V : μ_v^H(W_i) ≤_H μ_{v_max}^H(W_i)
6 |   |   Add v_max to the end of the list P.
7 |   end
8 |   return P
9 end
```

highest is chosen. We present an algorithm performing such an operation (see Algorithm 6.8). However, we assume that during execution of the Algorithm 6.8 the following data structures and algorithms are available:

- $\mathbf{T}$ is a temporal information system,
- $\mathbf{G} = (V, E)$ is a behavioral graph such that $V = \{v_1, ..., v_m\}$,
- $(H, \leq_H)$ is a linearly ordered set such that H is a set of labels of concepts layers and $\leq_H$ is a relation of linear order on the set H,
- $\mu^H_{v_1}, ..., \mu^H_{v_m}$ is a family of stratifying classifiers, which are constructed for temporal concepts corresponding to nodes $v_1, ..., v_m \in V$ (see Algorithm 6.5).

Assuming that all operations of classifiers μ^H_v (for $v \in V$) applications occurring in the above algorithm are executed at the constant time, then the temporal computational complexity of the above algorithm is of order $O(k \cdot m)$, where k is the length of the sequence of time windows which is replaced and $m = card(V)$.

6.17 Temporal Concept for Structured Objects

Complex behaviors of structured objects could be defined on sequences of time windows with the use of complex concepts which are called temporal concepts for structured objects.

We assume that temporal concepts for structured objects are specified by a human expert and are usually used in queries about the status of some structured objects in a particular sequence of time windows. Answers to such queries can be of the form Yes, No or *Does not concern*.

Intuitively each such temporal concept (defined on a sequence of time windows) depends on whether there occurred behaviors defined by temporal concepts for unstructured objects in the observed time windows, with those objects being parts of structured objects. It is usually possible to provide an ontology which shows such a dependence.

For example, temporal concept for structured objects which is a group of two objects (vehicles) *unhurried driving vehicle A after vehicle B in the right lane*, depends on such temporal concepts as: *driving at stable speed*, *changing lanes at constant speed*, *maintaining constant distance between vehicles A and B*, and others.

Temporal concepts for structured objects cannot be straightforwardly approximated by temporal concepts for single time windows because they concern a sequence of time windows, that is, a longer period of time than temporal concepts for time windows.

Therefore, in order to define attributes approximating temporal concepts for structured objects we need to introduce another language which enables us to transfer properties of time windows onto the level of sequences of time windows.

Due to the fact that each sequence of time windows can be replaced with a temporal path in the behavioral graph (see Section 6.16), while describing the properties of time windows we can use temporal paths corresponding to them. In this way, the language FTP below is constructed.

Definition 42 (*A language for defining features of temporal paths*)**.** *Let* $\mathbf{G} = (V, E)$ *be a behavioral graph of complex objects of the fixed type* T*. A language for defining features of temporal paths of behavioral graph* $\mathbf{G}$ *(denoted by* $FTP(\mathbf{G})$ *or* FTP*-language, when* $\mathbf{G}$ *is fixed) is defined in the following way:*

- *the set* $AL_{FTP}(\mathbf{G}) = V \cup \{$ *ExistsNode, EachNode, MajorityNodes, MinorityNodes, FirstNode, LastNode, OrderNodes* $\} \cup \{\neg, \vee, \wedge\}$ *is an alphabet of the language* $FTP(\mathbf{G})$*,*
- *for any* $v, v' \in V$ *expressions of the form:* $ExistsNode(v)$*,* $EachNode(v)$*,* $MajorityNodes(v)$*,* $MinorityNodes(v)$*,* $FirstNode(v)$*,* $LastNode(v)$*,* $OrderNodes(v, v')$ *are atomic formulas of the language* $FTP(\mathbf{G})$*.*

Presently, we define the semantics of the language $FTP(\mathbf{G})$. The formulas of the language $FTP(\mathbf{G})$ express properties of paths in the behavioral graph $\mathbf{G}$. For example, formula $ExistsNode(v)$ is interpreted as description of all those paths in the behavioral graph in which there exists node v.

Definition 43. *Let* $\mathbf{G} = (V, E)$ *be a behavioral graph of complex objects of the fixed type* T*. The satisfiability of an atomic formula* $\phi \in FTP(\mathbf{G})$ *by a temporal path* $P = (v_1, \ldots, v_k) \in PATH(\mathbf{G})$ *(denoted by* $P \models_{FTP(\mathbf{G})} \phi$*), is defined in the following way:*

1. $P \models_{FTP(\mathbf{G})} ExistsNode(v) \Leftrightarrow \exists_{i \in \{1,\ldots,k\}}\ v_i = v$,
2. $P \models_{FTP(\mathbf{G})} EachNode(v) \Leftrightarrow \forall_{i \in \{1,\ldots,k\}}\ v_i = v$,
3. $P \models_{FTP(\mathbf{G})} MajorityNodes(v) \Leftrightarrow$

$$card(\{i \in \{1, ..., k\} : v_i = v\}) > \lfloor (k-1)/2 \rfloor,$$

4. $P \models_{FTP(\mathbf{G})} MinorityNodes(v) \Leftrightarrow$

$$card(\{i \in \{1, ..., k\} : v_i = v\}) < \lceil (k-1)/2 \rceil,$$

5. $P \models_{FTP(\mathbf{G})} FirstNode(v) \Leftrightarrow v_1 = v$,
6. $P \models_{FTP(\mathbf{G})} LastNode(v) \Leftrightarrow v_k = v$,
7. $P \models_{FTP(\mathbf{G})} OrderNodes(v, v') \Leftrightarrow \exists_{i,j \in \{1,...,k\}}\ i < j\ \wedge\ v_i = v\ \wedge\ v_j = v'$.

Below, we provide a few examples of the FTP-language formulas.

- If node v of a certain behavioral graph of a single vehicle corresponds to the concept of *stable speed in the right lane*, then formula $EachNode(v)$ describes a temporal path in which the vehicle drives the whole time at the stable speed in the right lane.
- If node v of a certain behavioral graph of a single vehicle corresponds to the concept of *accelerating in the right lane*, then formula $ExistsNode(v)$ describes a temporal path in which the vehicle for a certain amount of time accelerates.
- If nodes v_1 and v_2 of a certain behavioral graph of a single vehicle correspond respectively to the concepts of *stable velocity in the right lane* and *accelerating in the right lane*, then formula $MajorityNodes(v_1) \wedge ExistsNode(v_2)$ describes a temporal path in which for most of the time the vehicle drive at the stable speed but for some time the vehicle increases speed.

Formulas of the FTP language can be used for defining temporal path features in such a way that to each formula a temporal pattern for temporal paths can be attached.

Definition 44 (*A temporal pattern for temporal paths*)**.** *Let $\mathbf{G} = (V, E)$ be a behavioral graph of complex objects of the fixed type T. Each formula of the language $FTP(\mathbf{G})$ is called a temporal pattern for temporal paths in the behavioral graph $\mathbf{G}$.*

We assume that any temporal pattern ought to be defined by a human expert using domain knowledge accumulated for the given complex dynamical system.

6.18 Information System of Temporal Paths

The properties of accessible temporal paths may be represented in special information system which is called *an information system of temporal paths.*

Definition 45 (*An information system of temporal paths*)**.** *Let us assume that:*

- *$\mathbf{G} = (V, E)$ is a behavioral graph of complex objects of the fixed type,*
- *s is a fixed length of temporal paths ($s > 1$),*
- *$\Phi = \{\phi_1, ..., \phi_k\} \subseteq FTP(\mathbf{G})$ is a defined by experts family of temporal patterns for temporal paths (sub-language of the language $FTP(\mathbf{G})$),*
- *$\mathcal{P}_{FTP} = (\overline{U}, \Phi, \models_{FTP(\mathbf{G})})$ is a property system, where $\overline{U} \subseteq PATH(\mathbf{G}, s)$.*

The information system $\overline{\mathbf{G}} = (\overline{U}, \overline{A})$ defined be the property system $\mathcal{P}_{FTP}$ is called an information system of temporal paths (TP-information system).

Evidently, construction of system $\overline{\mathbf{G}}$ requires generating a family of temporal paths of a fixed length. It is possible to construct an algorithm generating all temporal paths of the established duration s from a given behavioral graph $\mathbf{G} = (V, E)$ which would work in such a way that for each node $v \in V$ there would be generated all paths of the established duration s which start in the node. If the number of edges coming out of a given node of graph $\mathbf{G}$ equaled no more than r, then the number of paths generated using such an algorithm would be pessimistically equal $card(V) \cdot r^{s-1}$, which in practice may be a great number of paths. For example, if $card(V) = 10$, $r = 4$ and $s = 10$ then the number of paths to generate equals over 2.6 million! Therefore, in practice, during system $\overline{\mathbf{G}}$ construction only chosen paths from the set $PATH(\mathbf{G}, s)$ are generated.

Although the set $PATH(\mathbf{G}, s)$ may be sampled, it is reasonable to assume that in system $\overline{\mathbf{G}}$ only those paths are taken into account which occur in the training data. This means that in system $\overline{\mathbf{G}}$ only such a path exists that is registered in the data for the specific complex object. Obviously, the node sequence is not directly registered in the data, but time point sequence is registered which may be grouped into time windows and then into sequences of time windows. These sequences, thanks to classifiers constructed for individual temporal concepts, may be changed into the node sequence from set V informing about the membership of time windows to individual temporal concepts. The set of all temporal paths observed in data of duration s we denote as $DPATH(\mathbf{G}, s)$. For small behavioral graphs the size of the set $DPATH(\mathbf{G}, s)$ is comparable with the size of the set $PATH(\mathbf{G}, s)$. However, for nontrivially small behavioral graphs the size of the set $DPATH(\mathbf{G}, s)$ is much smaller than the size of the set $PATH(\mathbf{G}, s)$. We present the algorithm of generating the set $DPATH(\mathbf{G}, s)$ (see Algorithm 6.9).

Algorithm 6.9. Generating temporal paths observed in data

Input:
- temporal information system $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ such that $U = \{u_1, ..., u_n\}$,
- behavioral graph $\mathbf{G} = (V, E)$ such that $V = \{v_1, ..., v_m\}$,
- fixed length of temporal paths s (where $s > 1$),
- fixed length of time windows l (where $l > 1$).

Output: The set $DPATH(\mathbf{G}, s)$

```
1 begin
2 |  Generate set TW := TW(T, s · (l − 1) + 1).
3 |  Create empty list of paths DPATH.
4 |  for all W ∈ TW do
5 |  |  path := MakeTimePath(Partition(W, s))
6 |  |  Add path to the list DPATH.
7 |  end
8 |  return DPATH
9 end
```

In regard to determining the set $TW(\mathbf{T}, s \cdot (l-1)+1)$ and executions of procedures $MakeTimePath$ and $Partition$, the pessimistic time complexity of the Algorithm 6.9 is of order $O(n \cdot log\ n + \ n \cdot s \cdot l + n \cdot s \cdot m)$, where n is the number of objects in the set U, s is the length of temporal paths, l is the length of time windows and $m = card(V)$.

Example 27. For the behavior graph from Fig. 28 an information system of temporal paths may be constructed. In order to do this one may use temporal patterns for temporal paths chosen from the following collection of patterns:

1. the vehicle accelerates (decelerates, maintains stable speed) in the right lane at the first node (at the last node, at some node, at all nodes, at the minority of nodes, at the majority of nodes) of the temporal path,
2. the vehicle accelerates and at the same time changes the lane from the right to the left one at the first node (at the last node, at some node, at all nodes, at the minority of nodes, at the majority of nodes) of the temporal path,
3. the vehicle decelerates (maintains stable speed) and at the same time changes lanes from the left to the right one at the first node (at the last node, at some node, at all nodes, at the minority of nodes, at the majority of nodes) of the temporal path,
4. the vehicle accelerates (decelerates , maintains stable speed) in the right lane at some node of the temporal path, and after that the vehicle accelerates (decelerates , maintains stable speed) in the right lane at one of the following nodes,
5. the vehicle accelerates (maintains stable speed) and at the same time changes the lane from the right to the left one at some node of the temporal path, and after that at one of the following nodes the vehicle decelerates (maintains stable speed) in the right lane,
6. the vehicle decelerates (maintains stable speed) and at the same time changes the lane from the left to the right one at some node of the temporal path, and after that at one of the following nodes the vehicle decelerates (maintains stable speed) in the right lane.

It is easy to notice that each of the above patterns may be expressed in language FTP. □

In approximating temporal concepts for structured objects (see Section 6.20), it is necessary to use information systems of temporal paths which are constructed on the basis of behavioral graphs illustrating relation changes between parts of structured objects. The example below shows how such an information system for behavior graph from Fig. 30 may be constructed.

Example 28. The behavioral graph from Fig. 30 describes distance changes between parts of the structured object which is a pair of vehicles driving on the road. For this graph the information system of temporal paths may be constructed using temporal patterns for temporal paths chosen from the following collection of patterns:

1. the distance between both vehicles is short and is decreasing (is increasing, is stable) at the first node (at the last node, at some node, at all nodes, at the minority of nodes, at the majority of nodes) of the temporal path,
2. the distance between both vehicles is long and is decreasing (is increasing, is stable) at the first node (at the last node, at some node, at all nodes, at the minority of nodes, at the majority of nodes) of the temporal path. □

The choice of specific temporal patterns to construct the information system of temporal paths depends on the temporal concept which is to be approximated using this system, obviously after performing the grouping (clustering) of temporal paths (see Section 6.20).

6.19 Clustering Temporal Paths

The properties of temporal paths expressed using temporal patterns may be used to approximate temporal concepts for structured objects. Frequently, however, objects of system $\overline{\mathbf{G}}$ are not yet suitable for their properties to describe temporal concepts for structured objects. It happens that way because there are too many of these objects and the descriptions of temporal paths, which they represent, are too detailed. Hence, if applied for temporal concept approximation for structured objects, the extension of the created classifier would be too small, that is, the classifier could classify too small number of tested paths.

Therefore, similarly to the case of temporal concepts for time windows the object clustering is applied which leads to obtaining the family of clusters of temporal paths. The grouping tools are the same as in the case of time window grouping. Therefore, to define path clusters we use the language $ECTP$ which similarly to the language $ECTW$ is based on the language NL.

Definition 46 (*A language for extracting clusters of temporal paths*)**.** *Let us assume that:*

- $\mathbf{G} = (V, E)$ *is a behavioral graph of complex objects of the fixed type,*
- $\overline{\mathbf{G}} = (\overline{U}, \overline{A})$ *is an information system of temporal paths of behavioral graph* $\mathbf{G}$.

Any neighborhood language $NL(\overline{\mathbf{G}})$ *we call a language for extracting clusters of temporal paths from behavioral graph* $\mathbf{G}$ *(*$ECTP$*-language) and we denote it by* $ECTP(\mathbf{G})$.

Formulas of the language $ECTP$ enable to define clusters of temporal paths from the behavioral graph.

Definition 47 (*A cluster of temporal paths*)**.** *Let* $\mathbf{G} = (V, E)$ *be a behavioral graph of complex objects of the fixed type. We call the meaning of each formula* $\phi \in ECTP(\mathbf{G})$ *a cluster of temporal paths of behavioral graph* $\mathbf{G}$.

In the example below we illustrate how the dissimilarity function may be defined for grouping temporal paths.

Example 29. For the information system of temporal paths obtained with the use of temporal patterns from Example 27, the dissimilarity function may be constructed on the basis of expert knowledge. In order to do this, the dissimilarity function value should be obtained from experts for a certain, i.e., the most representative set of object pairs of this system. It should be also determined what temporal concept is approximated using clusters obtained with the help of the dissimilarity function being defined (see Section 6.20). It is possible to approximate concepts defined on the set of unstructured objects as well as on the set of structured objects. Let us assume that this is the concept concerning the overtaking maneuver of vehicle B by vehicle A and we mean this fragment of the overtaking maneuver when vehicle B drives in the right lane and vehicle A while driving behind vehicle B changes the lane from the right to the left one. In terms of this concept and knowledge about typical behaviors of vehicle A, the expert may instantly define several groups of vehicles which, if put in place of vehicle A, behave very similarly, that is, vehicles which are in one of such groups in terms of the dissimilarity function which is being constructed, should not differ significantly. For example, they may be the following groups of vehicles:

I. vehicles which at the beginning of the temporal path drive in the right lane and then drive off the right lane into the left one,
II. vehicles which along the whole temporal path drive in the right lane,
III. vehicles which along the temporal path drive in the left lane,
IV. vehicles which at the beginning of the temporal path drive in the left lane and then drive off the left lane into the right one.

Let us now consider five specific vehicles which behave in the following way:

1. vehicle u_1 accelerates in the right lane at the beginning of the temporal path, and then still accelerates and commences changing lanes from the right to the left one, but at the end of the temporal path is still not in the left lane,
2. vehicle u_2 drives at a stable speed in the right lane at the beginning of the temporal path, and then commences changing lanes from the right to the left one at a stable speed, but at the end of the temporal path it is still not in the left lane,
3. vehicle u_3 drives at a stable speed in the right lane along the whole temporal path,
4. vehicle u_4 drives at a stable speed in the left lane at the beginning of the temporal path, and then decelerates in the left lane,
5. vehicle u_5 drives at a stable speed in the left lane at the beginning of the temporal path, and then at a stable speed commences changing lanes from the left to the right one, but at the end of the temporal path it is still not in the right lane.

In Table 6 values of chosen temporal patterns for objects u_1, u_2, u_3, u_4 and u_5 are presented.

It is evident that object u_1 belongs to the group of vehicles I and it does not belong to groups II, III and IV. Similarly, object u_2 also belongs to group I. The

Table 6. An exemplary objects of some *TP*-information system

No	**Temporal patterns for temporal paths**	u_1	u_2	u_3	u_4	u_5
1.	The vehicle accelerates driving in the right lane at the first node of the temporal path	1	0	0	0	0
2.	The vehicle drives at a stable speed in the right lane at the first node of the temporal pattern	0	1	1	0	0
3.	The vehicle accelerates and changes lanes from the right to the left one at some node of the temporal path	1	0	0	0	0
4.	The vehicle drives at a stable speed and changes lanes from right to the left one at some node of the temporal path	0	1	0	0	0
5.	The vehicle drives at a stable speed in the left lane at the first node of the temporal path	0	0	0	1	1
6.	The vehicle decelerates in the left lane at some node of the temporal path	0	0	0	1	0
7.	The vehicle drives at a stable speed and changes lanes from the left to the right one at some node of the temporal path	0	0	0	0	0
8.	The vehicle decelerates and changes lanes from the left to the right one at some node of the temporal path	0	0	0	0	1
9.	The vehicle drives at a stable speed in the right lane at the last node of the temporal path	0	0	1	0	0
10.	The vehicle drives at a stable speed in the left lane at the last node of the temporal path	0	0	0	0	0

reason for this is the fact that both objects drive in the right lane first and then drive off to the left lane, but object u_2 drives at a stable speed the whole time and object u_1 accelerates. Therefore, the dissimilarity function value for these two objects is low, that is, 0.1, whereas object u_3 belongs to group II because it drives in the right lane the whole time. That is why its behavior differs significantly from the behavior of object u_2, for object u_3 only drives in the right lane while object u_2 changes lanes from the right to the left one. Hence, the dissimilarity function value for objects u_2 and u_3 is larger and equals 0.3. Object u_4 belongs to group III (drives only in the left lane) and therefore its behavior has even less in common with the behavior of object u_2. Therefore, the dissimilarity function value for the pair of objects u_2 and u_4 is 0.4. Finally, object u_5 belongs to group IV and its behavior differs the most from the behavior of object u_1. The reason for this is the fact that object u_5 not only commences driving in the left lane (unlike u_1, which commences driving in the right lane), but at the end of the considered temporal path u_5 changes lanes from the left to the right one, unlike u_1, which changes lanes from the right to the left one. Moreover, the object u_5

Table 7. Values of dissimilarity function for pairs of temporal paths

	u_1	u_2	u_3	u_4	u_5
u_1	0	0.1	0.4	0.4	1.0
u_2	0.1	0	0.3	0.3	0.9
u_3	0.4	0.3	0	0.6	0.4
u_4	0.4	0.4	0.6	0	0.4
u_5	1.0	0.9	0.4	0.4	0

decelerates, while the object u_1 accelerates. Therefore, the dissimilarity function value for the pair of objects u_1 and u_5 is the highest possible and equals 1.0. Let us notice that the dissimilarity function value for the pair of objects u_2 and u_5 is bit smaller (0.9) because the object u_5 decelerates, while the object u_2 maintains stable speed. The dissimilarity function values for all pairs of objects proposed by the expert are put together in Table 7. Finally, let us notice that objects u_1, u_3, u_4 and u_5 may be treated as standards for which the deviation is 0.1. □

Now, to construct a new information system which represents the features of clusters of temporal paths we need to define a new language.

Definition 48 (*A language for definnig features of clusters of temporal paths***).** *Let us assume that:*

- $\mathbf{G} = (V, E)$ *is a behavioral graph of complex objects of the fixed type,*
- $\overline{\mathbf{G}} = (\overline{U}, \overline{A})$ *is a information system of temporal paths for behavioral graph* $\mathbf{G}$.

A language for definnig features of clusters of temporal paths for behavioral graph $\mathbf{G}$ *(denoted by* $FCTP(\mathbf{G})$ *or* $FCTP$*-language, when* $\mathbf{G}$ *is fixed) is defined in the following way:*

- *the set* $AL_{FCTP}(\mathbf{G}) = AL_{FTP}(\mathbf{G}) \cup \{ExistsPath, EachPath, MajorityPaths, MinorityPaths\}$ *is an alphabet of the language* $FCTP(\mathbf{G})$,
- *for any* $\alpha, \beta \in FTP(\mathbf{G})$ *expressions of the form:* $ExistsPath(\alpha)$, $EachPath(\alpha)$, $MajorityPaths(\alpha)$, $MinorityPaths(\alpha)$ *are atomic formulas of the language* $FCTP(\mathbf{G})$.

Now, we determine semantics of the language $FCTP(\mathbf{G})$. The formulas of the language $FCTP(\mathbf{G})$ we interpret as a description of clusters of temporal paths of graph G. For example, formula $ExistsPath(\alpha)$ is interpreted as the description of all those clusters of temporal paths in which there exists at least one path satisfying formula α.

Definition 49. *Let* $\mathbf{G} = (V, E)$ *be a behavioral graph of complex objects of the fixed type* T. *The satisfiability of an atomic formula* $\phi \in FCTP(\mathbf{G})$ *by a family of temporal paths* $\mathcal{P} = \{P_1, ..., P_k\} \subseteq PATH(\mathbf{G})$ *(denoted by* $\mathcal{P} \models_{FCTP(\mathbf{G})} \phi$*) is defined in the following way:*

1. $\mathcal{P} \models_{FCTP(\mathbf{G})} ExistsPath(\alpha) \Leftrightarrow \exists_{P \in \mathcal{P}}\ P \models_{FTP(\mathbf{G})} \alpha$,
2. $\mathcal{P} \models_{FCTP(\mathbf{G})} EachPath(\alpha) \Leftrightarrow \forall_{P \in \mathcal{P}}\ P \models_{FTP(\mathbf{G})} \alpha$,
3. $\mathcal{P} \models_{FCTP(\mathbf{G})} MajorityPaths(\alpha) \Leftrightarrow$

$$card(\{P \in \mathcal{P} : P \models_{FTP(\mathbf{G})} \alpha\}) > \lfloor (k-1)/2 \rfloor,$$

4. $\mathcal{P} \models_{FCTP(\mathbf{G})} MinorityPaths(\alpha) \Leftrightarrow$

$$card(\{P \in \mathcal{P} : P \models_{FTP(\mathbf{G})} \alpha\}) < \lceil (k-1)/2 \rceil.$$

The patterns of the language $FCTP$ may be applied to define the features of clusters of temporal paths.

The example below illustrates how one may define properties of clusters of temporal paths which are constructed using the dissimilarity function from Example 29.

Example 30. Using the dissimilarity function from Example 29, the objects from the information system of temporal paths from Example 27 may be grouped. However, in order to construct an information system for the determined family of clusters, patterns should be established which serve computing features of clusters of temporal paths. For example, they may be the following patterns:

1. in each temporal path of the cluster vehicles first drive in the right lane and then change lanes from the right to the left one (the pattern describes the property of vehicles from group I from Example 29),
2. in each temporal path of the cluster vehicles drive only in the right lane (the pattern describes the property of vehicles from group II from Example 29),
3. in each temporal path of the cluster vehicles first drive in the left lane and then change lanes from the left to the right one (the end of the overtaking maneuver),
4. in each temporal path of the cluster there exists a node at which vehicles drive in the left lane,
5. in the majority of temporal paths of the cluster there occurs no deceleration,
6. in the majority of temporal paths of the cluster vehicles drive at a stable speed the whole time.

It is easy to notice that each of the above patterns may be expressed in language $FCTP$. □

The choice of specific patterns to construct the information system of the temporal path clusters depends on the concept which is to be approximated with the help of this system (see Section 6.20).

Clusters of temporal paths are represented by the information systems which we call *an information system of clusters of temporal paths.*

Definition 50 (*An information system of clusters of temporal paths*)**.** *Let us assume that:*

- $\mathbf{G} = (V, E)$ *is a behavioral graph of complex objects of the fixed type,*
- $\psi_1, ..., \psi_n \in ECTP(\mathbf{G})$ *is a defined by experts family of temporal patterns corresponding to clusters of temporal paths* $CL_1, ..., CL_n$,

- $\Phi = \{\phi_1, ..., \phi_m\} \subseteq FCTP(\mathbf{G})$ *is a defined by experts family of features of clusters of temporal paths,*
- $\mathcal{P}_{FCTP} = (\overline{\overline{U}}, \Phi, \models_{FCTP(\mathbf{G})})$ *is a property system, where* $\overline{\overline{U}} = \{CL_1, ..., CL_n\}$.

The information system $\overline{\overline{\mathbf{G}}} = (\overline{\overline{U}}, \overline{\overline{A}})$ *defined be the property system* $\mathcal{P}_{FCTP}$ *is called an information system of clusters of temporal paths (CTP-information system).*

Evidently, constructing system $\overline{\overline{\mathbf{G}}}$ requires generating clusters of temporal paths in such a way that it would be possible to check the ability to satisfy the formulas of the language $FCTP(\mathbf{G})$. Such clusters may be generated by the linear search in the table $\overline{\mathbf{G}}$ and assigning objects of this table to individual clusters.

Algorithm 6.10 presented bellow determines for a given temporal path the list of feature values of the cluster to which the tested temporal path belongs. Therefore, this algorithm is important for testing temporal paths. We assume

Algorithm 6.10. Computation features of cluster to which the tested temporal path belongs

Input: temporal path $P = (v_1, \ldots, v_s) \in PATH(\mathbf{G})$.
Output: The list of feature values of the cluster to which the tested temporal path P belongs

```
1  Procedure GetClusterRow(P)
2  begin
3      Create empty list row
4      for φ1, ..., φm do
5          if P ⊨FTP(G) φi then
6              Add '1' to the end of the list row
7          else
8              Add '0' to the end of the list row
9          end
10     end
11     Create new object u_row of the system G on the basis values of
       attributes from the list row.
12     Select a formula ψ = (std, ε) ∈ {ψ1, ..., ψn) such that:
13       1. u_row ⊨FCTP(G) ψ and
14       2. DISM_G(u_row, std) = min_{i∈{1,...,n}}{DISM_G(u_row, std_i)}
15     Select object u_ψ ∈ U corresponding to the formula ψ
16     Create empty list clusterRow
17     for i = 1 to k do
18         clusterRow := clusterRow + a_i(u_ψ)
19     end
20     return clusterRow
21 end
```

that during execution of the Algorithm 6.10 the following data structures are available:

- a behavioral graph $\mathbf{G} = (V, E)$ of complex objects of a fixed type,
- a fixed length s of temporal paths in the behavioral graph $\mathbf{G}$,
- a family of temporal patterns $\phi_1, ..., \phi_m \in FTP(\mathbf{G}))$ and a system $\overline{\mathbf{G}} = (\overline{U}, \overline{A})$ such that attributes from the set $\overline{A}$ correspond to formulas $\phi_1, ..., \phi_m$,
- a dissimilarity function $DISM_{\overline{\mathbf{G}}}$ of object pairs from the system $\overline{\mathbf{G}}$, approximated using the stratifying classifier $\mu_{DISM_{\overline{\mathbf{G}}}}$,
- a family of temporal patterns $\psi_1, ..., \psi_n \in ECTP(\mathbf{G})$ such that $\psi_i = (std_i, \varepsilon_i)$ (for $i = 1, ..., n$) and a system $\overline{\overline{\mathbf{G}}} = (\overline{\overline{U}}, \overline{\overline{A}})$ such that objects from the set $\overline{\overline{U}}$ correspond to clusters described by formulas $\psi_1, ..., \psi_n$ and $\overline{\overline{A}} = \{\overline{\overline{a}}_1, ..., \overline{\overline{a}}_k\}$.

Assuming that all operations of checking the satisfiability of formulas for a given path and computing values of dissimilarity function (occurring in the above algorithm) are carried out at the constant time, then the temporal computational complexity of the above algorithm is of order $O(m+n+k)$ where m is the number of used temporal path features, n is the number of clusters of temporal paths, and k is the number of features of clusters of temporal paths.

6.20 Temporal Concept Table for Structured Objects

CTP-information systems constructed for different unstructured objects may be joined in order to obtain information systems describing features of structured objects. We obtain objects of such a system by arranging (joining) all possible objects of information systems being joined. From the mathematical point of view such an arrangement is the Cartesian product of sets of objects of joined information systems (see [78, 84, 186, 187]). In this way we obtain an information system that may be used to approximate concepts for structured objects (see Definition 52). However, from the point of view of domain knowledge not all mentioned above object arrangements are possible and sensible. For example, if we approximate the concept of overtaking performed by two vehicles (overtaking and overtaken ones), then it is sensible to link the three following path clusters:

1. the first path cluster (coming from the behavior graph of the overtaking vehicle) describes a sequence of vehicle behaviors which behave in the same way as the overtaking vehicle (*e.g.*, accelerate and change lanes from the right to the left one),
2. the second path cluster (coming from the behavior graph of the overtaken vehicle) describes a sequence of vehicle behaviors which behave in the same way as the overtaken vehicle (*e.g.*, drive with a stable speed in the right lane),
3. the third path cluster (coming from the behavior graph describing relation changes between the overtaking and overtaken vehicle) describes a sequence of behaviors of such vehicles pairs that the relations determined between them change in a way connected with overtaking (*e.g.*, the distance between vehicles decreases rather quickly).

For the above reason, that is, to eliminate object arrangements which are unreal or are not sensible, a relation of constraints is formulated which informs which objects may be arranged in order to obtain positive or negative example of approximated concept (for structured objects) and which cannot be arranged. The relation of constraints we define in the form of the formula of the language GDL.

Definition 51 (*An information system of clusters of temporal paths for structured objects*)**.** *Let us assume that:*

- $\mathbf{G}_i = (V_i, E_i)$ *is a behavioral graph of complex objects of the fixed type* T_i*, for* $i = 1, ..., k$*,*
- $\overline{\overline{\mathbf{G}}}_i = (\overline{\overline{U}}_i, \overline{\overline{A}}_i)$ *is a CTP-information system for behavioral graph* $\mathbf{G}_i$*, for* $i = 1, ..., k$*,*
- $\mathbf{G}_R = (V_R, E_R)$ *a behavioral graph representing changes of relations between parts of structured objects, where such parts are objects of types* $T_1, ..., T_k$*,*
- $\overline{\overline{\mathbf{G}}}_R = (\overline{\overline{U}}_R, \overline{\overline{A}}_R)$ *is a CTP-information system for behavioral graph* $\mathbf{G}_R$*,*
- $\mathbf{G}_\otimes = (U_\otimes, A_\otimes)$ *is an information system, where:*
  - $U_\otimes = \overline{\overline{U}}_1 \times \ldots \times \overline{\overline{U}}_k \times \overline{\overline{U}}_R$*,*
  - $A_\otimes = A_1 \cup \ldots \cup A_k \cup A_R$*, where attributes from sets* $A_1, ..., A_k, A_R$ *are natural extension of attributes from sets* $\overline{\overline{A}}_1, ..., \overline{\overline{A}}_k, \overline{\overline{A}}_R$ *on the set* $U_\otimes$*,*
- $\Phi \in GDL(\mathbf{G}_\otimes)$ *is a formula of constraints.*

An information system of clusters of temporal paths for structured objects (SCTP-information system) is an information system $\mathbf{G}_\bowtie = (U_\bowtie, A_\bowtie)$*, where:*

- $U_\bowtie = \{u \in U_\otimes : u \models_{GDL(\mathbf{G}_\otimes)} \Phi\}$*,*
- $A_\bowtie = A_\otimes$*, i.e., the set* $A_\bowtie$ *contains all attributes from the set* $A_\otimes$*, that are restricted to* $U_\bowtie$*.*

The notion of information system $\mathbf{G}_\bowtie$ may be used to approximate concepts for structured objects. In order to do this it is sufficient to add a decision attribute to this system which describes the approximated concept. We assume that for each arrangement of objects accepted by constraints (satisfying the formula of constraints), the expert adds the decision value informing about whether a given arrangement belongs to the approximated concept of the higher level or not.

Now, the definition of a temporal concept table for structured objects may be presented.

Definition 52 (*A temporal concept table for structured objects*)**.** *Let us assume that:*

- $\mathbf{G}_\bowtie = (U_\bowtie, A_\bowtie)$ *is an information system of clusters of temporal paths for structured objects,*
- $C \subseteq U_\bowtie$ *is a temporal concept for structured objects.*

A temporal concept table for structured objects for the concept C *is a decision table* $\mathbf{G}^C_\bowtie = (U_\bowtie, A_\bowtie, d_C)$*, where the decision attribute* d_C *is representing membership of objects from the set* $U_\bowtie$ *to the concept* C*.*

As we already mentioned, the relation of constraints is defined as a formula in the language GDL (see Definition 5) on the basis of attributes of the table $\mathbf{G}_{\otimes}$. However, the relation of constraints may also be approximated with the help of classifiers. It is required, in such a case, to give examples of objects belonging and not belonging to this relation, that is, satisfying and not satisfying the formula Φ corresponding to this relation (see [78]).

Example 31. In Fig. 32 there is presented a construction scheme of temporal concept table for structured objects consisting of two vehicles, i.e., an overtaking vehicle (vehicle A) and an overtaken vehicle (vehicle B). Any object of this table is represented by a triple of clusters $(P_i^{(A)}, P_i^{(B)}, P_i^{(R)})$ for $i \in \{1, ..., card(U_{\bowtie})\}$, where:

- $P_i^{(A)}$ denotes the cluster of overtaking vehicles,
- $P_i^{(B)}$ denotes the cluster of overtaken vehicles,
- $P_i^{(R)}$ denotes the cluster of pairs of both vehicles.

Attributes describing the clusters of overtaking and overtaken vehicles, *e.g.*, clusters $P_i^{(A)}$ and $P_i^{(B)}$ are constructed as presented in Example 30. They de-

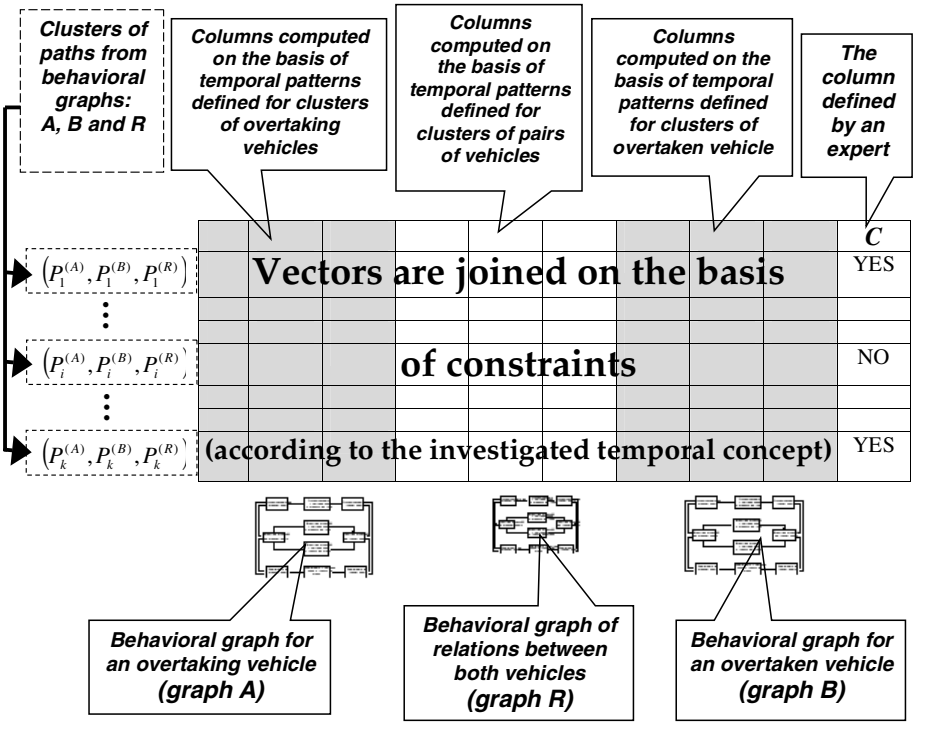


Fig. 32. The scheme of the temporal concept table of structured objects (two vehicles during the overtaking maneuver)

scribe rather general behaviors of individual vehicles observed during the sequence of time windows. Whereas, the attributes describing cluster $P_i^{(R)}$ describe the properties of both vehicles (A and B) in terms of relation changes between these vehicles. During observing vehicles during the overtaking maneuver the most important thing is to observe the distance changes between these vehicles (see Example 28). Therefore, to describe the properties of $P_i^{(R)}$ clusters the following patterns may be applied:

1. in each temporal path of the cluster, at the majority of nodes the distance between A and B is short and is still decreasing,
2. in each temporal path of the cluster there exists a node at which the distance between A and B is long,
3. in each temporal path of the cluster at the majority of nodes the distance between A and B is short and is increasing,
4. in the majority of temporal paths of the cluster there exists a node at which the distance between A and B is long and it is decreasing, and after it there exists a node at which the distance between A and B is short and is decreasing,
5. in each temporal path of the cluster and at all the nodes of these paths the distance between A and B is short.

The information system obtained in such a way may be enriched with the decision attribute which describes the membership of individual clusters of time windows to the established temporal concept for the group of both vehicles (*e.g.*, it may be the concept: *vehicle B drives in the right lane and vehicle A while driving behind vehicle B changes the lane from the right to the left one*). In this way we obtain a decision table which may serve the approximation of this concept.

It is worth noticing that the attribute created on the basis of the last of the above mentioned patterns may be used to construct the constraint formula. If we build the constraint formula based on this attribute, then the clusters joints representing vehicles which are at a long distance are eliminated from the system $\mathbf{G}_{\otimes}$. Such cluster joints are not useful for differentiating pairs of vehicles which drive close to each other and are involved in the overtaking maneuver from those pairs of vehicles which also drive close to each other, but they are not involved in the overtaking maneuver. □

A question arises, how it is possible that the expert may propose the decision value related to the membership to system $\mathbf{G}_{\bowtie}$. To make this decision the expert has at his or her disposal attribute values from the set $A_{\bowtie}$. They are attributes proposed earlier by the expert as features of clusters of temporal paths. The expert is able to propose them in such a way that he/she may now use them successively to determine the membership of the whole clusters to concept C. In other words cluster features must be defined in such a way that they could serve to determine the membership of clusters of time paths to temporal concept C.

If $E = \{e_1, ..., e_l\}$ is a family of layer labels of concept C, then for table $\mathbf{G}_{\bowtie}^{C}$ stratifying classifier μ_C^E may be constructed. This classifier enables classifying structured objects to concept C.

6.21 Classification of Structured Objects

In this section, we present an algorithm which allows to answer the question whether the behavior of structured objects observed in a sequence of time windows belongs to a given temporal concept defined for structured objects.

Stratifying classifier μ_C^E constructed for table $\mathbf{G}_{\bowtie}^C$ enables the classification of structured objects to concept C. However, it may be applied for the tested object which is the arrangement of clusters of temporal paths of the behavioral graph for complex objects being parts of the examined structured object and clusters of temporal paths of the behavioral graph describing relation changes between the parts of the examined structured object. That is why it is necessary to construct earlier a suitable object (like in the table $\mathbf{G}_{\bowtie}^C$) in order to test of structured object. We present the algorithm of structured object classification. However, we assume that during execution of the Algorithm 6.5 the following elements are available:

- a test c-temporal information system $\mathbf{T}_{TS-P}$,
- a sweeping algorithm SA around objects from systems $\mathbf{T}_{TS-P}$,
- a test c-temporal information system $\mathbf{T}_{TS-R}$ representing features of relations between parts of structured objects determined by the algorithm SA,
- a formula of constraints $\Phi \in GDL(\mathbf{G}_{\otimes})$,
- a concept C defined by experts in the set $U_{\bowtie}$, representing some feature of structured objects of a fixed type T, where any object of the type T consists of parts, that are objects of types $T_1, ..., T_k$,
- a temporal concept table $\mathbf{G}_{\bowtie}^C$ for the concept C, constructed using the formula of constraints Φ,
- a linearly ordered set $(H, \leq_H)$ such that H is a set of labels of concepts layers and $\leq_H$ is a relation of linear order on the set H,
- a stratifying classifier μ_C^H constructed for the concept C on the basis the table $\mathbf{G}_{\bowtie}^C$.

The Algorithm 6.11 works as follows. On the input of the algorithm there is a description given of the behavior of a certain structured object, in the form of sequences of time windows: $S_1, ..., S_k, S_{k+1}$ which describe the behavior of a part of this object (sequences: $S_1, ..., S_k$) and the relation changes between the parts of this object (the sequence S_{k+1}). Next, on the basis of the sequences of time windows obtained on the input and with the usage of algorithms $MakeTimePath$ and $GetClusterRow$, there is created a new object u_{row} which has the structure of objects from the $\mathbf{G}_{\otimes}$ table. Next, it is checked whether object u_{row} satisfies the constraints expressed by formula Φ. If it happens this way, then object u_{row} is classified by the stratifying classifier μ_C^H, otherwise the algorithm returns the information that it cannot classify the behavior of the complex object whose description is given on the input of the algorithm.

Assuming that the operation of examining the satisfiability of formula Φ and the operation of classification with the help of classifier μ_C^H may be performed in constant time and considering the complexity of procedures $MakeTimePath$ and $GetClusterRow$, the pessimistic time complexity of the Algorithm 6.11 is of

Algorithm 6.11. Structured object classification to the temporal concept

Input: Family of sequences of time windows $S_1, ..., S_k, S_{k+1}$ describing behavior of a given structured object, where $S_i \in SEQ(\mathbf{T}_{TS-P})$, for $i = 1, ..., k$ and $S_{k+1} \in SEQ(\mathbf{T}_{TS-R})$

Output: Information about the membership of the observed structured object to the concept C

```
1  begin
2  |  Create empty list row
3  |  for i := 1 to k + 1 do
4  |  |  path := MakeTimePath(S_i) (see Algorithm 6.8)
5  |  |  pRow := GetClusterRow(path) (see Algorithm 6.10)
6  |  |  Add pRow to the end of the list row
7  |  end
8  |  Create new object u_row of the system G_⊗ on the basis values of
   |  attributes from the list row.
9  |  if u_row ⊭_{GDL(G_⊗)} Φ then
10 |  |  return "Has nothing to do with"
11 |  else
12 |  |  return μ_C^H(u_row)
13 |  end
14 end
```

order $O(k \cdot s \cdot m + k \cdot n_{pf} + k \cdot n_c + k \cdot n_{cf})$, where k is the number of parts of which the examined structured objects consist, s is the length of each $S_1, ..., S_k, S_{k+1}$ sequences, m is the maximum number of nodes from the behavior graphs of individual parts of structured objects, n_{pf} is the maximum number of features used to define path properties in behavior graphs, n_c is the maximum number of path clusters in behavior graphs and n_{cf} is the maximum number of the applied features of such clusters.

6.22 Behavioral Graphs for Structured Objects

Analogously to the case of temporal concepts for unstructured complex objects, temporal concepts defined for structured objects may also be treated as nodes of a certain directed graph that we call *a behavioral graph for a structured object*. One can say, that the behavioral graph for a structured object expresses temporal dependencies on a higher level of generalization than the behavioral graph on lower level, i.e., the behavioral graph for unstructured objects.

Definition 53 (*A behavioral graph for a structured object*)**.**

1. *A behavioral graph for a structured object $\mathcal{G}$ is an ordered pair $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a nonempty and finite set of nodes (temporal concepts for structured objects) and $\mathcal{E}$ is a set of directed edges $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ (connections represent temporal dependencies between temporal concepts for structured objects).*

2. *A temporal path in the behavioral graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *is a sequence of nodes* $v_1, ..., v_l$ *such that for any* $i \in \{1, ..., l-1\}$ *an edge* $(v_i, v_{i+1}) \in \mathcal{E}$. *A number* l *is called a length of temporal path* $v_1, ..., v_l$.
3. *A family of all temporal paths with length* l $(l > 0)$ *in the behavioral graph* $\mathcal{G}$ *is denoted by* $PATH(\mathcal{G}, l)$.

Bellow we present an example of behavioral graph for a structured object.

Example 32. In Fig. 33, we present exemplary behavioral graph for a structured object, that is a group of two vehicles: vehicle A and vehicle B, related to the standard overtaking maneuver. There are 6 nodes in this graph representing the following temporal concepts: *vehicle A is driving behind B on the right lane*, *vehicle A is changing lanes from right to left*, *vehicle A is moving back to the right lane*, *vehicle A is passing B when A is on the left lane and B is on the right lane*, *vehicle A is changing lanes from left to right and vehicle A is before B on the right lane*. There are 7 connections representing spatio-temporal dependencies between temporal concepts from nodes. For example, after the node *Vehicle A is driving behind B on the right lane* the behavior of these two vehicles matches to the node (temporal concept) *Vehicle A is changing lanes from right to left and B is driving on the right lane.* □

6.23 Behavioral Patterns

Both the behavioral graph for an unstructured object and the behavioral graph for a structured object may be used as a complex classifier enabling the identification of the behavioral pattern described by this graph. It is possible based on the observation of the behavior of a unstructured object or a structured object over a longer period of time and testing if the behavior matches the chosen path of the behavioral graph. If this is the case, it is stated that the behavior matches

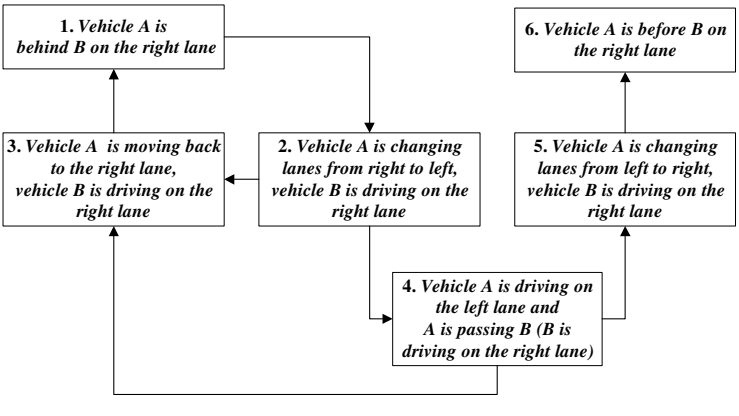


Fig. 33. A behavioral graph for the maneuver of overtaking

Algorithm 6.12. Behavioral pattern identification

Input: Family of sequences of time windows $S_1, ..., S_k, S_{k+1}$ describing behavior of a given structured object, where:
- $S_i \in SEQ(\mathbf{T}_{TS-P})$, for $i = 1, ..., k$ and $S_{k+1} \in SEQ(\mathbf{T}_{TS-R})$,
- $Length(S_i) = z \cdot s$, where z is a natural number and s is a length of sequences of time windows used for identification of temporal concepts for structured objects, for $i = 1, ..., k+1$.

Output: The binary information (*YES* or *NO*) about matching the observed structured object to the behavioral pattern represented by the graph $\mathcal{G}$

```
1  begin
2      Create empty list path
3      for i := 0 to z − 1 do
4          SS_1 := Subsequence(S_1, i · s + 1, (i + 1) · s)
5          ...
6          SS_k := Subsequence(S_k, i · s + 1, (i + 1) · s)
7          SS_{k+1} := Subsequence(S_{k+1}, i · s + 1, (i + 1) · s)
8          v_max := v_1
9          layer_max := μ^H_{v_1}(SS_1, ..., SS_k, SS_{k+1})
10         for j := 2 to m do
11             layer := μ^H_{v_j}(SS_1, ..., SS_k, SS_{k+1})
12             if layer_max ≤_H layer then
13                 layer_max := layer
14             end
15         end
16         Add v_max to the end of the list path.
17     end
18     if path ∈ PATH(G, z) then
19         return YES
20     else
21         return NO
22     end
23 end
```

behavioral pattern represented by this graph which enables detecting particular behaviors in a complex dynamical system.

In result, we can use a behavioral graph as a complex classifier for perception of the complex behavior of unstructured or structured objects. For this reason, a behavioral graph constructed for some complex behavior is called a *behavioral pattern*.

We present a detailed algorithm of behavioral pattern identification (see Algorithm 6.12). However, we assume that during execution of the Algorithm 6.12 the following elements are available:

- a test c-temporal information system $\mathbf{T}_{TS-P}$,
- a sweeping algorithm SA around objects from system $\mathbf{T}_{TS-P}$,
- a test c-temporal information system $\mathbf{T}_{TS-R}$ representing features of relations between parts of structured objects determined by the algorithm SA,
- a behavioral graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of structured objects of a fixed type T determined by the algorithm SA such that $\mathcal{V} = \{v_1, ..., v_m\}$,
- a linearly ordered set $(H, \leq_H)$ such that H is a set of labels of concepts layers and $\leq_H$ is a relation of linear order on the set H,
- a family of stratifying classifiers $\mu^H_{v_1}, ..., \mu^H_{v_m}$ constructed for concepts corresponding to nodes $v_1, ..., v_m \in \mathcal{V}$.

The way of working Algorithm 6.12 is the following. On the input there is a description given of the behavior of a certain structured object, in the form of sequences of time windows: $S_1, ..., S_k, S_{k+1}$ which describe the behavior of a part of this object (sequences: $S_1, ..., S_k$) and the relation changes between the parts of this object (sequence S_{k+1}). The length of these sequences is established and equals $z \cdot s$, where z is a fixed natural number and s is the length of sequences of time windows needed to identify a single temporal concept for structured objects. Further, for the all subsequence families of the length s isolated from the input sequences, stratifying classifiers are applied constructed for concepts corresponding to all nodes of behavior graph $\mathcal{G}$. Thus, it is possible to chose such a node for each subsequence family that the classifier corresponding to it classifies the subsequence family to the possibly lowest layer (if there are more such nodes the choice among them is nondeterministic). The chosen node is put at the end of the path represented using the *path* list. After choosing the node of graph $\mathcal{G}$ for all subsequence families and what follows completing the *path* list, the return of decision value occurs which tells us whether the examined structured object matches the behavioral pattern represented by graph $\mathcal{G}$ or not. Namely, the YES decision is returned when the list *path* is a path in graph $\mathcal{G}$, otherwise the NO value is returned.

The above algorithm is able to classify a given structured object only when all subsequence families isolated from the input family of the sequence of time windows may be classified. However, this is possible only when each subsequence family is classified at least by one of the stratifying classifiers available for nodes from set $\mathcal{V}$.

Let us notice that in terms of computational complexity the time cost of executing the Algorithm 6.12 is equal, with an accuracy of constant, the sum of costs of $(z \cdot m)$-times execution of Algorithm 6.11, where z is the number of the subsequences of the length s into which the input sequences are divided and $m = card(\mathcal{V})$.

Example 33. Let us study the behavioral graph presented in Fig. 33 for a group of two objects-vehicles (vehicle A and vehicle B) related to the standard overtaking maneuver. We can see that the path of temporal concepts with indexes "1, 2, 3, 1, 2, 4" matches a path from this behavioral graph, while the path with indexes: "6, 5, 4" doesn't match any path from this behavioral graph (this path can match some other behavioral patterns). □

A path of temporal patterns (that makes it possible to identify behavioral patterns) should have a suitable length. In the case where the length is too short, it may be impossible to discern one behavioral pattern from another pattern. For example, we can make a mistake between an overtaking and a passing by a vehicle in traffic.

6.24 Discovering Rules for Fast Elimination of Behavioral Patterns

In this section, we present how the method of behavioral patterns identification presented in Section 6.23 can be speeded up using special decision rules. Let us assume that we have a family of behavioral patterns $BP = \{b_1, ..., b_n\}$ defined for structured objects (or parts of a given structured object). For any pattern b_i from the family BP one can construct a complex classifier based on a suitable behavioral graph (see Section 6.23) that makes it possible to answer the question: "Does the behavior of the investigated structured object (or the part of a structured object) match the pattern b_i?". The identification of behavioral patterns of any structured object is performed by investigation of a sequence of time windows registered for this object during some period (sometimes quite long). This registration of time windows is necessary if we want to avoid mistakes in identification of the investigated structured object. However, in many applications, we are forced to make a faster (often in real-time) decision if some structured object matches the given behavioral pattern. In other words, we would like to check the investigated structured object at once, that is, using the first or second time window of our observation only. This is very important from the computational complexity point of view, because if we investigate complex dynamical systems, we usually have to investigate many structured objects. Hence, faster verification of structured objects can help us to optimize the process of searching among structured objects matching the given behavioral pattern.

The verification of complex dynamical systems consisting of some structured objects can be speeded up by using some special decision rules, that are computed by an Algorithm 6.13 (see also Fig. 34).

Any decision rule computed by the Algorithm 6.13 expresses a dependency between a temporal concept and the set of behavioral patterns that are not matching this temporal concept. Such rules can make it possible to exclude very quickly many parts of a given complex object as irrelevant for identification of a given behavioral pattern. This is possible because these rules can be often applied at once, that is after only one time window of our observation. Let us consider a very simple illustrative example. Assume we are interested in the recognition of overtaking that can be understood as a behavioral pattern, defined for the group of two vehicles. Using the methodology presented above, we can obtain the following decision rule:

- `If the vehicle A is overtaking B then the vehicle B is driving on the right lane.`

Algorithm 6.13. Discovering decision rules for fast elimination of behavioral patterns

Input:
- c-temporal information system $\mathbf{T}$,
- a family of behavioral patterns $BP = \{b_1, ..., b_n\}$ defined for structured objects or parts of structured objects.

Output: Decision rules for fast elimination of behavioral patterns from the family BP

1. Define a family of temporal concepts $TC = \{t_1, ..., t_m\}$ that have influence on matching investigated structured objects to behavioral patterns from family BP (defined on the basis of information from time windows from the set $TW(\mathbf{T})$).
2. Construct classifiers for all temporal concepts from TC using the method from Section 6.9.
3. For any temporal concept t_i from the family TC create a decision table DT_i, that has the following structure:
   (a) any row of the table DT_i is constructed on the basis of information registered during a period that is typical for the given temporal concept t_i,
   (b) the condition attribute b of table DT_i registers the index of behavioral pattern from the family BP (the index computation is based on observation that any complex classifier from BP can check for the investigated structured objects if there is a sequence of time windows matching the given behavioral pattern and starting from a given time window),
   (c) the decision attribute of the table DT_i is computed on the basis of values returned by classifier constructed for t_i in previous step.
4. Compute decision rules for DT_i using methods of discretization by attribute values grouping (see Section 2.2).

After applying the transposition law, we obtain the following rule:

- `If the vehicle B is not driving on the right lane then the vehicle A is not overtaking B.`

The last rule (see also Fig. 35) allowing fast verification whether the investigated structured object (two vehicles: A and B) is matching the behavioral pattern of overtaking.

Of course, in case of the considered complex dynamical system, there are many other rules that can help us in the fast verification of structured objects related to the overtaking behavioral pattern. Besides, there are many other behavioral patterns in this complex dynamical system and we have to calculate rules for them using the methodology presented above.

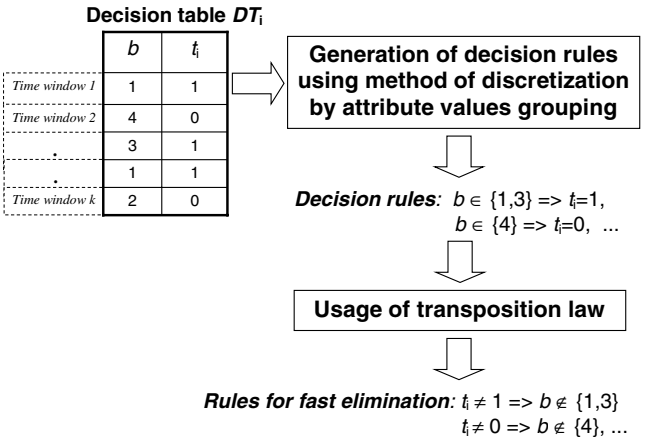


Fig. 34. The scheme of rules extraction for the fast elimination of behavioral patterns from data tables

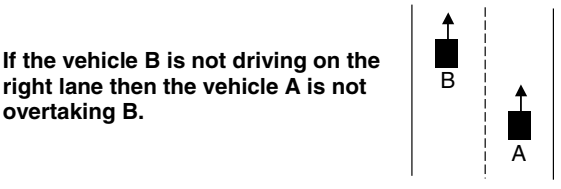


Fig. 35. The illustration of the decision rule for fast elimination of behavioral pattern

The presented method, that we call *the method for on-line elimination of non-relevant for a given behavioral pattern parts of complex object* (ENP method [177, 178]), is not a method for behavioral pattern identification. However, this method allows to eliminate some paths of a given complex object behavior that are not relevant for checking if this object matches a given behavioral pattern. After such elimination the complex classifiers based on a suitable behavioral graphs should be applied to the remaining complex objects.

6.25 Experiments with Road Simulator Data

To verify the effectiveness of classifiers based on behavioral patterns, we have implemented the algorithms from the *Behavioral Patterns* library (BP-lib), which is an extension of the RSES-lib 2.1 library forming the computational kernel of the RSES system (see [15]). Our experiments have been performed on the data sets obtained from the road simulator (see Appendix A) and on the medical data sets. In this section we report results of experiments preformed on the data sets obtained from the road simulator. Results obtained for the medical data sets are presented in Section 6.26.

In the case of experiments on the data sets obtained from the road simulator, we have applied the *train-and-test* method. A training set consisted of about 17 thousands objects generated by the road simulator during one thousand of simulation steps. Whereas, a testing set consisted also of about 17 thousands objects collected during another (completely different) session with the road simulator.

In our experiments, we compared the quality of three classifiers: *rough set classifier with decomposition* (RS-D), *Behavioral Pattern classifier* (BP) and *Behavioral Pattern classifier with the fast elimination of behavioral patterns* (BP-ENP).

For induction of RS-D, we employed RSES system generating the set of minimal decision rules by algorithm LEM2 (see Section 2.4) that are next used for classification of situations from the testing data. However, we had to use the method of generating decision rules joined with a standard decomposition algorithm from the RSES system (see Section 2.7). This was necessary because the size of the training table was too large for the direct generation of decision rules. The classifiers BP is based on behavioral patterns (see Section 6.23), whilst BP-ENP are based on behavioral patterns too but with application of fast elimination of behavioral patterns related to the investigated group of objects (see Section 6.24).

In application of BP and BP-ENP methods the distance between time points was constant, that is time points were recorded after each stage of the simulation. The prediction of temporal concepts for individual vehicles was performed on the basis of time windows whose duration equals 3 time points, whereas the prediction of temporal concepts for pairs of vehicles was performed on the basis of sequence of time windows whose duration was equal 2. Finally, the behavioral pattern was recognized on the basis of vehicle observation over 2 sequences of time windows, that is on the basis of vehicle observation over 4 time windows. The tested object for the analyzed behavioral pattern was, thus, the sequence of 9 successive time points. Therefore, classifier RS-D used the table whose objects were sequences of 9 successive time points, whereas the attributes gave the values of sensor attributes for all these points. The value of decision attribute was given by the expert in the same manner for all three classification methods.

We compared RS-D, BP, and BP-ENP classifiers using the accuracy, the coverage, the real accuracy, the accuracy for positive examples (the sensitivity or the true positive rate), the coverage for positive examples, the real accuracy for positive examples, the accuracy for negative examples (the specificity or the true negative rate), the coverage for negative examples and the real accuracy for negative examples (see Section 2.9).

In order to determine the standard deviation of the obtained results each experiment was repeated for 10 pairs of tables (training table + testing table). Therefore, 20 tables in total were applied (collected during 20 completely different sessions with the road simulator).

Table 8 shows the results of applying these classification algorithms for the concept related to the *overtaking* behavioral pattern.

Table 8. Results of experiments for the overtaking pattern

Decision class	Method	Accuracy	Coverage	Real accuracy
Yes	RS-D	0.875 ± 0.050	0.760 ± 0.069	0.665 ± 0.038
(overtaking)	BP	0.954 ± 0.033	1.000 ± 0.000	0.954 ± 0.033
	BP-ENP	0.947 ± 0.031	1.000 ± 0.000	0.947 ± 0.031
No	RS-D	0.996 ± 0.001	0.989 ± 0.003	0.985 ± 0.001
(no overtaking)	BP	0.990 ± 0.004	1.000 ± 0.000	0.990 ± 0.004
	BP-ENP	0.990 ± 0.004	1.000 ± 0.000	0.990 ± 0.004
All classes	RS-D	0.993 ± 0.002	0.980 ± 0.004	0.973 ± 0.002
(Yes + No)	BP	0.988 ± 0.003	1.000 ± 0.000	0.988 ± 0.003
	BP-ENP	0.988 ± 0.003	1.000 ± 0.000	0.988 ± 0.003

One can see that in the case of perception of the overtaking maneuver (decision class Yes) the accuracy and the real accuracy of algorithm BP are higher than the accuracy and the real accuracy of algorithm RS-D for the analyzed data set. Besides, we see that the accuracy of algorithm BP-ENP (for decision class YES) is only 0.7 percent lower than the accuracy of algorithm BP. Whereas, the algorithm BP-ENP allows us to reduce the time of perception, because during perception we can usually identify the lack of overtaking earlier than in the algorithm BP. This means that it is not necessary to collect and investigate the whole sequence of time windows (that is required in the BP method) but only some first part of this sequence. In our experiments with the classifier BP-ENP it was enough to use on average $59.7\% \pm 1.5\%$ percent of the whole time window sequence for objects from the decision class No (the lack of overtaking in the sequence of time windows). However, it should be stressed that this result concerns only identification of vehicle groups which were preliminarily selected by the sweeping algorithm (see Section 6.13), whereas in comparison with the number of time windows needed to analyze all possible two-element vehicle groups, in using the sweeping algorithm and BP-ENP method the number of analyzed time windows constitutes only $2.3\% \pm 0.1\%$ of the number of time windows needed to analyze all 2-element vehicle groups.

6.26 Risk Pattern Identification in Medical Data: Case Study

An identification of some behavioral patterns can be very important for identification or prediction of behavior of a complex dynamical system, especially when behavioral patterns describe some dangerous situations. In this case, we call such behavioral patterns *risk patterns* and we need some tools for their identification. If in the current situation some risk patterns are identified, then the control object (a driver of the vehicle, a medical doctor, a pilot of the aircraft, etc.) can use this information to adjust selected parameters to obtain the desirable behavior of the complex dynamical system. This can make it possible to overcome inconvenient or unsafe situations. For example, a very important element of the treatment of the infants with respiratory failure is the appropriate assessment of

the risk of death. The appropriate assessment of this risk leads to the decision of particular method and level of treatment. Therefore, if some complex behavior of an infant that causes a danger of death is identified, we can try to change her/his behavior by using some other methods of treatment (may be more radical) in order to avoid the infant's death. In this section we describe how the presented approach in previous sections can be applied to identification of the infants' death risk caused by respiratory failure (see Appendix B). In this approach, a given patient is treated as an investigated complex dynamical system, whilst diseases of this patient (RDS, PDA, sepsis, Ureaplasma and respiratory failure) are treated as complex objects changing and interacting over time.

It is also worthwhile mentioning that the research reported in this section is a continuation, in some sense, of the previous research on the survival analysis (see [324, 325, 326]).

Medical Temporal Patterns. As we wrote before (see, e.g, Section 6.4), the concepts concerning the properties of complex objects at the current time point in connection with the previous time point are a way of representing very simple behaviors of the complex objects. These concepts, that we call *elementary concepts*, usually characterize a status of sensor's values. In the case of our medical example (the treatment of the infants with respiratory failure), we can distinguish the following elementary concepts such as *low value of FiO2* (the percent concentration of oxygen in the gas entering the lungs), *increase in FiO2* , *decrease in PaO2* (the arterial oxygen tension), *decrease in PaO2/FiO2*, *low creatinine serum (blood) level*. However, a perception of more complex behaviors requires identification of elementary concepts over a longer period called a *time window* (see Section 6.6). Therefore, if we want to predict such more complex behaviors or discover a behavioral pattern, we have to investigate elementary concepts registered in the current time window. Such investigation can be expressed using temporal patterns. For example, in the case of the medical example one can consider patters expressed by following questions: "Did PaO2/FiO2 increase in the first point of the time window?", "Was PaO2/FiO2 stable in the time window?", "Did the PaO2/FiO2 increase before the closing of PDA?" or "Did the PaO2/FiO2 increase before a PaO2/FiO2 decrease occurred?". Notice that all such patterns ought to be defined by a human, medical expert using domain knowledge accumulated for the respiratory failure disease.

Behavioral Graph for a Disease. The temporal patterns can be treated as new features that can be used to approximate temporal concepts (see Section 6.6). In the case of the treatment of infants with respiratory failure one can define temporal concepts such as "Is the infant suffering from the RDS on level 1?", "Was an multi-organ failure detected?", or "Is the progress in multi-organ failure on level 4?".

Temporal concepts defined for objects from a complex dynamical system and approximated by classifiers, can be treated as nodes of a graph called a *behavioral graph* (see Section 6.11), where connections between nodes represent temporal dependencies. Fig. 36 presents a behavioral graph for a single patient exhibiting

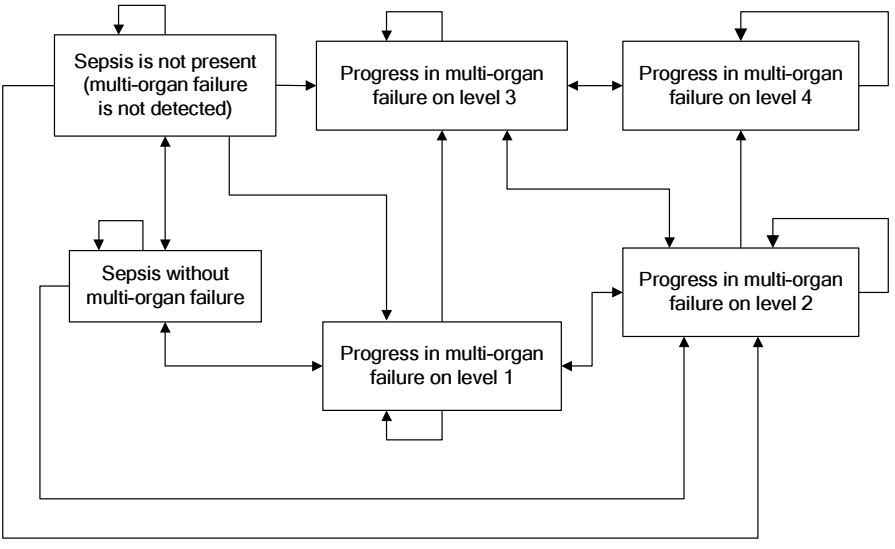


Fig. 36. A behavioral graph of sepsis by analyzing the multi-organ failure

a behavioral pattern of patient by analysis of the organ failure caused by sepsis. This graph has been created on the basis of observation of medical data sets (see Appendix B) and the SOFA scale (Sepsis-related Organ Failure Assessment) (see [327, 328] for more details).

In this behavioral graph, for example, connections between node "Progress in multi-organ failure on level 1" and node "Progress in multi-organ failure on level 3" indicates that after some period of progress in organ failure on level 1 (rather low progressing), a patient can change his behavior to the period, when progress in organ failure is high. In addition, a behavioral graph can be constructed for different kinds of diseases (like RDS, PDA, Ureaplasma) (see Appendix B) or groups of diseases represented for example by the respiratory failure (see Fig. 37).

Medical Risk Pattern. The temporal concepts defined for structured objects and approximated by classifiers, are nodes of a new graph, that we call *a behavioral graph for a structured object* (see Section 6.22). In Fig. 37, we present an exemplary behavioral graph for group of four diseases: sepsis, Ureaplasma, RDS and PDA, related to the behavior of the infant during high death risk period due to respiratory failure. This graph has been created on the basis of observation of medical data sets (see next subsection) and with support of medical experts. There are 16 nodes in this graph and 21 connections represented spatio-temporal dependencies between temporal concepts from nodes. For example, after the node "Stabile and mild respiratory failure in sepsis" the behavior of patient can match the node "Exacerbation of respiratory failure from mild to moderate in sepsis".

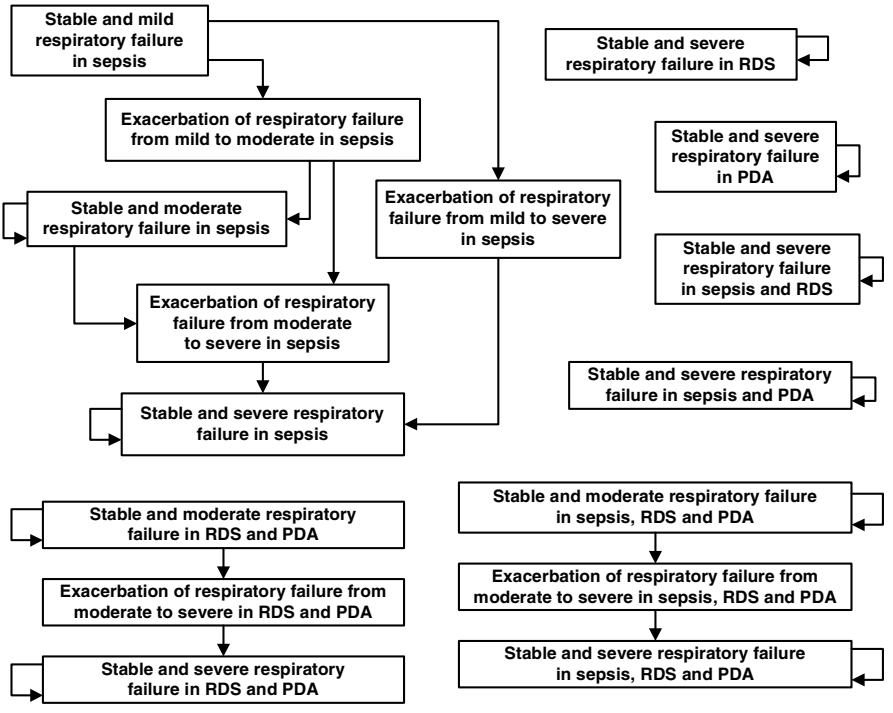


Fig. 37. A behavioral graph of the infant during high death risk period due to respiratory failure

This behavioral graph is an example of risk pattern. We can see that the path of temporal patterns: ("Stable and mild respiratory failure in sepsis", "Exacerbation of respiratory failure from mild to severe in sepsis", "Stable and severe respiratory failure in sepsis") matches a path from this behavioral graph, while the path: ("Stable and severe respiratory failure in sepsis", "Exacerbation of respiratory failure from moderate to severe in sepsis", "Stable and moderate respiratory failure in sepsis") doesn't match any path from this behavioral graph.

Experiments with Medical Data. In this section we present results of experiments performed for obtained from Neonatal Intensive Care Unit, First Department of Pediatrics, Polish-American Institute of Pediatrics, Collegium Medicum, Jagiellonian University, Krakow, Poland. The data were collected between 2002 and 2004 using computer database NIS, i.e, Neonatal Information System (see [329]). The detailed information about treatment of 340 newborns are available in the data set, such as perinatal history, birth weight, gestational age, lab tests results, imagine techniques results, detailed diagnoses during hospitalization, procedures and medication were recorded for the each patient. The study group included prematurely born infants with the birth weight $\leq 1500g$, admitted to

the hospital before end of the 2 day of life. Additionally, the children suffering from the respiratory failure but without diagnosis of RDS, PDA, sepsis or Ureaplasma infection during their entire clinical course were excluded from the study group (193 patients stayed after the reduction).

In our experiments, we used one data table extracted from the NIS system, that consists of 11099 objects. Each object of this table describes parameters of one patient in single time point.

The aim of conducted experiments was to check the effectiveness of the algorithms described in this paper in order to predict the behavioral pattern related to a high risk of death of infants. This pattern was defined by experts (see Fig. 37). It is worth adding that as many as 90.9% of infants whose behavior matched this pattern died shortly after (this fact results from a simple analysis of medical data set which were gathered).

As a measure of classification success (or failure) we use: the accuracy, the coverage, the real accuracy, the accuracy for positive examples (the high risk of death), the coverage for positive examples, the real accuracy for positive examples, the accuracy for negative examples (the low risk of death), the coverage for negative examples and the real accuracy for negative examples (see Section 2.9).

We have applied the *train-and-test* method. However, because of the specificity of the analyzed data the method of data division differed slightly from the standard method. Namely, in each experiment the whole patient set was randomly divided into two groups (training and testing one). The same number of patients belonged to each of these groups, at the same time patients who died and those who survived were divided separately. In other words, in each of two groups the number of dead patients and those who survived was the same. This division of data was necessary because the correlation between patient's death and the fact of matching patient's behavior the considered behavioral pattern is very strong. Obviously, the information about whether the patient died or survived the treatment was not available during learning and testing process of classifiers.

In the discussed experiments the distance between time points recorded for a specific patient was variable, that is, various time points of different frequencies were recorded in the data over different periods of time. For instance, if patient's condition was serious, then quite often (*e.g.*, every two hours) parameters representing his or her condition were registered and recorded for this patient, whereas if the patient's condition was good and stable, then the information about this patient was recorded rather rarely (*e.g.*, once a day). In relation to this, although the prediction of temporal concepts for individual disease (RDS, PDA, sepsis, Ureaplasma) was always performed on the basis of time windows having 2 time points, then in practice these windows had very different temporal durations. This way, the duration of time windows was in a certain way determined by experts. However, the prediction of temporal concepts for respiratory failure was performed on the basis of the sequence of time windows whose duration was equal 2. Finally, the pattern of high death risk was recognized on the basis of patient observation for 3 sequences of time windows that is on the basis

Table 9. Results of experiments for the risk pattern of death due to respiratory failure

Decision class	Accuracy	Coverage	Real accuracy
Yes (the high risk of death)	0.994 ± 0.008	1.0 ± 0.000	0.994 ± 0.008
No (the low risk of death)	0.938 ± 0.012	1.0 ± 0.000	0.938 ± 0.012
All classes (Yes + No)	0.958 ± 0.010	1.0 ± 0.000	0.958 ± 0.010

of observation over 6 time windows. A tested object for the analyzed behavioral pattern was, therefore, the sequence of 7 successive time points.

As a result of the above mentioned division of patients into training and testing ones, each of these parts made it possible to create approximately 6000 time windows having duration of 7 time points. Time windows created on the basis of training patients created a training table for a given experiment, while time windows created on the basis of tested patients created a test table for the experiment.

In order to determine the standard deviation of the obtained results each experiment was repeated for 10 random divisions of the whole data set.

Table 9 shows the results of applying this algorithm for the concept related to the risk pattern of death due to respiratory failure. Together with the results we present a standard deviation of the obtained results.

Notice, that the accuracy of decision class Yes in medical statistics (see [260] and Section 2.9) is called a *sensitivity* (the proportion of those cases having a true positive test result of all positive cases tested), whereas the accuracy of decision class No is called a *specificity* (the proportion of true negatives of all the negative samples tested). We see both main parameters of our classifier (i.e., sensitivity and specificity) are sufficiently high.

Experimental results showed that the suggested method of behavioral patterns identification gives good results, also in the opinion of medical experts (compatible enough with the medical experience) and may be applied in medical practice as a supporting tool for infants suffering from respiratory failure monitoring.

Some results of our experiments on medical data were surprising even for medical experts (*e.g.*, very low frequency of fatal cases in infants with Ureaplasma infection). Therefore, one can say that our tools were useful for development of new interesting observation and experience.

Finally, let us notice that the specific feature of the methods considered here is not only high accuracy (with low standard deviation) but also very high coverage (equal 1.0).

7 Automated Planning Based on Data Sets and Domain Knowledge

Behavioral patterns described in Section 6 may be very useful for effective complex dynamical systems monitoring, particularly when certain behavioral patterns are connected to undesirable behaviors of complex objects. If during the observation

of complex dynamical system such a pattern is identified then the control module may try to change, using appropriate actions, the behavior of the system in such a way as to get the system out of an uncomfortable or dangerous situation. However, these types of short-term interventions may not be sufficient for a permanent rescuing the system out of an undesirable situation. Therefore, very often the possibility of using some methods of automated planning is considered.

Automated planning is a branch of artificial intelligence that concerns the realization of strategies or action sequences (called as plans), typically for execution by intelligent agents, autonomous robots and unmanned vehicles, that can change their environment (see, *e.g.*, [70, 76, 273]). The essential inputs for planning are an initial world state, a repertoire of action for changing that world, and a set of goals. The purpose of plan is to achieve one or more goals. The form of the plan is commonly just a linear sequence of actions or acyclic directed graph of actions.

In the case of the control of complex dynamical systems, automated generated plans can be used to carry out a given complex object to more comfortable or safer situation (see, *e.g.*, [70, 72, 74, 76, 180, 181]). Such plans may be performed by (or on) unstructured objects or structured objects, i.e., by each part of any structured object separately with a presence of complete synchronization of actions performed by (or on) individual parts of the structured object).

7.1 Classical Algorithms of Automated Planning

Classical planning algorithms can be classified according to how they structure the search space. There are three very common such classes, like: *state space planners*, *plan space planners* and *planners encoding the planning problem as a problem of some other kind* (see, *e.g.*, [70, 71, 74, 330, 331]).

State Space Planners. In the first class of classical planning algorithm are very early planners, *e.g.*, STRIPS (see [332]), and some successful recent planners, like Graphplan (see [333]). These algorithms are based on searching in the state space, where such searching is most often done either by forward-chaining, i.e., searching from the initial state to the goal state, or by backward-chaining, i.e., searching from the goal state to the initial state (see, *e.g.*, [70, 71, 74] for more details).

Plan Space Search. In the second class of planning algorithms are causal link planners and constraint-based planners (see, *e.g.*, [74]). In this case, the plan space consists of incomplete plans, which, in contrast to the notion of plan in the state space view, do not have to be sequences or actions or sets of parallel actions. One alternative is to view the plan as a partially ordered set of actions. The potential advantage of this view is that one partially ordered set can represent many linear plans, and that the planner needs only to enforce the orderings that are absolutely necessary, whereas in a linear plan, many action orderings are quite arbitrary (see, *e.g.*, [70, 71, 74] for more details).

Encoding Planning as a Different Problem. A third class of classical planners encode the planning problem as a problem of some other kind and

solve this problem. This approach was first used in SATPLAN (see, e.g, [71, 334, 335, 336, 337]) which converts the planning problem instance into an instance of the Boolean satisfiability problem, which is then solved using a method for establishing satisfiability such as the DPLL algorithm or WalkSAT (see, *e.g.*, [338]). Other methods of planning in which the planning problem has been encoded as a problem of some other kind are methods based on a linear programming (see, *e.g.*, [339, 340]) or using constraint satisfaction problems (CSP) (see, *e.g.*, [341]).

7.2 Domain Dependent Automated Planning

Planning applications in practice may be large and involve complicated actions, but they commonly also have a great deal of structure, known to people experienced with the domain. There are many potential plans that are (to the human domain expert) obviously useless, and sometimes simple criteria can be found for sifting out the "promising" partial solutions. If this knowledge is encoded and given to the planner, it should help to speed up the process of finding a plan. This idea leads to what is called "domain-dependent planning".

There are many planning methods which use domain knowledge. At this section we briefly discuss a few most widely known exemplary approaches, that is *the planning with learning*, *the planning with time*, *the planning with incomplete information*, *the hierarchical task network planning* and *the domain-dependent search control*.

Learning in Planning. Generally speaking, machine learning techniques can be used to extract useful knowledge, from solutions to similar problem instances in the past or from previous failed attempts to solve the present problem instance. This has been used to improve planning efficiency and to improve plan quality (see, *e.g.*, [70, 71] for more details). For instance, two of the earliest systems to integrate machine learning and planning are SOAR (see, *e.g.*, [35, 302]), a general cognitive architecture for developing systems that exhibit intelligent behaviour, and PRODIGY, an architecture that integrates planning and learning in its several modules (see, *e.g.*, [303, 342]).

The approach to automated planning presented in this paper may also be included into the approaches integrating methods of machine learning with classical planning. However, there are significant differences between methods known from literature and methods presented in this paper (see Section 7.3 for more details).

Planning with Time. In classical planning actions are assumed to take "unit time". This assumption is not critically important, as long as there is no deadline to meet and one does not try to optimize the actual execution time. An early planner to deal with actions of different duration and goals with deadlines is Deviser (see [343]). It is based on the idea of partial-order planning, in which the simple partial order over the actions in the plan are replaced by more complex constraints on their starting and ending times. The idea has been picked up in

several later planners (see, *e.g.*, [70, 71, 74] for more details). Recently, temporal planning has become a very active area of research, and almost every classical planning approach has been adapted to deal with durative actions.

Planning with Incomplete Information. Another assumption made by most planners is that all relevant information is available in the problem description, and that the effects of actions are perfectly predictable. There have been several approaches to relaxing this assumption, by introducing probabilistic information. The probabilistic information has been used with state space planners, partial-order planners but most of all in the form of Markov Decision Problems (see, *e.g.*, [70, 71, 74] for more details).

Hierarchical Task Network Planning. Hierarchical Task Network (HTN) planning (see, *e.g.*, [70, 344, 345]) is like classical planning in that each state of world is represented by set of atoms, and each action corresponds to deterministic state transition. However, HTN planners differ from classical planners in what they plan for and how they plan for it. In an HTN planner, the objective is not to achieve a set of goals but instead to perform some set of tasks. The input to the planning system includes a set of operators similar to those of classical planning and also a set of methods. Each of which is a prescription for how to decompose some task into some set of subtasks (smaller tasks). Planning proceeds by decomposition non-primitive tasks recursively into smaller and smaller subtasks, until primitive tasks are reached that can be performed directly using the planning operators (see [70, 71] for more details).

Domain-Dependent Search Control. Many search-based planners allow the speciation of domain-dependent heuristics and rules for controlling and reducing search. For example, two recent planners, TLPlan (see [346]) and TALplanner (see [347, 348]) depend entirely on domain-specific search control, given in the form of logical formulas.

7.3 Automated Planning for Complex Objects

In the all aforementioned approaches to automated planning for complex objects, it is assumed that we know the current state of the complex object, which results from a simple analysis of current values of this object's available parameters. In other words the state of the complex object may be directly read from the values of its parameters or from a simple analysis of dependencies between these values. For example, if we consider a classic *blocks world problem* (see, *e.g.*, [74, 349]) which consists in planning the way of arranging available blocks on the table to make a determined construction, the state of the object is the information about the current placement of the blocks; and at the same time while planning the arrangement the answers to the three following questions are taken into account.

1. *Is a given block lying directly on the table?*
2. *Is another block lying on a given block?*
3. *Is another specific block lying on a given block?*

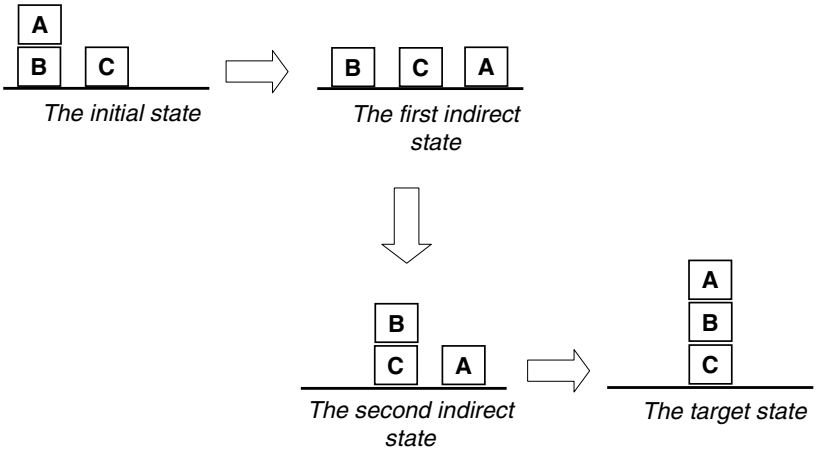


Fig. 38. Four states in the blocks world problem

For example, for the state in which there are three blocks available: A, B and C where blocks B and C are lying directly on the table and block A is lying on block B (initial state from Fig. 38), the description of such a state could be described in a natural language with the help of the five following facts:

1. *block A is lying directly on the table*,
2. *block B is lying directly on the table*,
3. *block C is lying on block A*,
4. *there is no block on block C*,
5. *there is no block on block B*.

Let us notice that in the above example concerning arranging a predetermined construction out of blocks, the description of the current state can be directly read from the information about the current values of their parameters, that is, from the information about the arrangement of blocks in relation to the table and other blocks. In the meantime, in complex dynamical system the state of the complex object is often expressed in a natural language using vague spatio-temporal conditions whose authenticity cannot be checked on the basis of a simple analysis of the available information on the object. For example, while planning treatment the condition of an infant who suffers from respiratory failure may be described by the following condition.

– *Patient with RDS type IV, persistent PDA and sepsis with mild internal organs involvement* (see Appendix B for mor medical details).

Stating the fact that a given patient is in the above condition requires the analysis of the examination result of this patient registered over a certain period

of time with a great support of domain knowledge deriving from experts (medical doctors). Conditions of this type can be represented by complex concepts and the identification of the condition is a check if the analyzed objects belong to this concept or not. However, the identification of such states requires approximation concepts representing them with the help of classifiers using data sets and domain knowledge. In a few next sections we describe the automated planning method for unstructured complex objects whose states are described using this type of complex concepts.

7.4 Planning Rules and Planning Graphs

The basic concept used in this paper for automated planning is *a planning rule*. It is a simple tool for modeling changes of the states of complex objects as a result of applying (performing) actions.

Definition 54 (*A planning rule*)**.** *Let S be the set of complex objects' states of the fixed type T and set A be the set of actions whose application causes the complex objects to change state from one to another. Each expression of the form: $(s_l, a) \rightarrow s_{r_1}|s_{r_2}\ldots|s_{r_k}$, where $s_l, s_{r_1}\ldots s_{r_k} \in S$ and $a \in A$ is called a planning rule of complex object of type T. Moreover, expression (s_l, a) is called a predecessor of the planning rule and expression $s_{r_1}|s_{r_2}\ldots|s_{r_k}$ is called a successor of the planning rule.*

Such rule can be used to change the state s_l of a complex object, using the action a to some state from the right hand side of a rule. But the result of applying such a rule is nondeterministic, because there are usually many states on the right hand side of a planning rule.

Example 34. Let us consider the planning rule from Fig. 39. This is the planning rule for treating RDS (respiratory distress syndrome) obtained from domain knowledge (see Appendix B). The rule may be applied when RDS with very severe hypoxemia is present. The application of the rule consists in performing a medical action utilizing the respirator in the MAP3 mode (see Example 36 for more medical details). As an effect of the application of this action at the following time point of observation (*e.g.*, the following morning) the patient's condition may remain unchanged or improve so as to reach the condition of RDS with severe hypoxemia. □

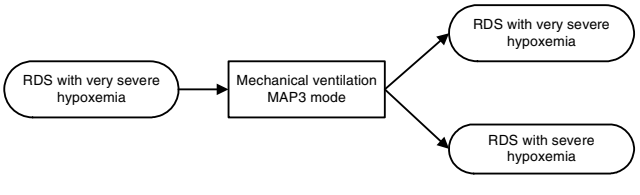


Fig. 39. The medical planning rule

Let us notice that there exists a certain similarity between the planning rules presented in the subsection and planning operators known from literature (see, *e.g.*, [70]). Similarly to the planning rule each operator describes an action which may be performed on a given complex object. However, each planning operator may have many initial conditions of its execution and many effects of its execution expressed with the help of a family of logical conditions which are to be satisfied after creating the operator. Whereas, in the approach described in this paper all initial conditions of executing a given planning rule are represented using one state which is a complex spatio-temporal concept which requires approximation. Similarly, the effects of planning rule performance are also represented using states which require approximation, which also distinguishes the presented approach from the methods known from literature.

A more complex tool, used to model changes of the states of complex objects as a result of applying action, is a planning graph whose paths describe such changes.

Definition 55 (*A planning graph*)**.**

1. *A planning graph for objects of a fixed type is an ordered triple* $\mathbf{PG} = (S, A, E)$*, where* S *is a nonempty and finite set of states,* A *is a nonempty and finite set of actions and* $E \subseteq (S \times A) \cup (A \times S)$ *is a set of directed edges.*
2. *If* $\mathbf{PG} = (S, A, E)$ *is a planning graph, then any* k*-element sequence* $(v_1, ..., v_k)$ *of elements from the set* $S \cup A$ *such that* $k > 1$ *and* $(v_i, v_{i+1}) \in E$ *for* $i \in \{1, ..., k-1\}$*, is called a path in the planning graph* $\mathbf{PG}$*.*
3. *A family of all paths with length* k *in the planning graph* $\mathbf{PG}$ *is denoted by* $PATH(\mathbf{PG}, k)$*, while a family of all paths in the planning graph* $\mathbf{PG}$ *is denoted by* $PATH(\mathbf{PG})$*.*
4. *Any path* $p' = (v_i, ..., v_j) \in PATH(\mathbf{PG}, j - i + 1)$ *created by removing from the path* $p = (v_1, ..., v_k) \in PATH(\mathbf{PG}, k)$ *elements* $v_1, ..., v_{i-1}$ *and* $v_{j+1}, ..., v_k$*, where* $i, j \in \{1, ..., k\}$ *and* $i < j$*, is called a sub-path of the path* p *and is denoted by* $Subpath(p, i, j)$*.*

Let us notice, that from the point of view of automata theory the planning graph is an nondeterministic finite automata in which the automata's states are states from the planning graph, the automata's alphabet is the set of actions from the planning graph and the transfer function is described by the edges of the planning graph.

Such paths in the planning graph are of a particular meaning for the process of automated planning. They tell us how it is possible to bring complex objects from the given state to another given state using actions. Therefore, these types of paths is called *plans*.

Definition 56 (*A plan in a planning graph*)**.** *Let* $\mathbf{PG} = (S, A, E)$ *be a planning graph.*

1. *Any path* $(v_1, ..., v_k) \in PATH(\mathbf{PG}, k)$ *is called a plan in the planning graph* $\mathbf{PG}$ *if and only if* $k > 2$ *and* $v_1, v_k \in S$*.*

2. *A family of all plans with length k in the planning graph* $\mathbf{PG}$ *is denoted by* $PLAN(\mathbf{PG}, k)$*, while a family of all plans in the planning graph* $\mathbf{PG}$ *is denoted by* $PLAN(\mathbf{PG})$*.*
3. *If* $p = (v_1, ..., v_k) \in PLAN(\mathbf{PG}, k)$*, then any sub-path* $Subpath(p, i, j)$ *such that* $v_i, v_j \in S$*, is called a sub-plan of the plan* p *and is denoted by* $Subplan(p, i, j)$*.*

Below, we present an example which illustrates such concepts as: a planning graph, a path in the planning graph and a plan in the planning graph.

Example 35. Let us consider planning graph $\mathbf{PG} = (S, A, E)$ such that $S = \{s_1, s_2, s_3, s_4\}$, $A = \{a_1, a_2, a_3\}$ and $E = \{(s_1, a_1), (s_1, a_2), (s_2, a_3), (s_3, a_3), (a_1, s_1), (a_1, s_3), (a_1, s_4), (a_2, s_1), (a_2, s_2), (a_3, s_4), (a_3, s_2), (a_3, s_3)\}$. This graph is presented in Fig. 40 where the states are represented using ovals, and actions are represented using rectangles. Each link between the nodes of this graph represents a time dependencies. For example, the link between state s_1 and action a_1 tells us that in state s_1 of the complex object action a_1 may be performed, whereas the link between action a_1 and state s_3 means that after performing action a_1 the state of the complex object may change to s_1. An example of a path in graph $\mathbf{PG}$ is sequence (a_2, s_2, a_3, s_4) whereas path $(s_1, a_2, s_2, a_3, s_3)$ is an exemplary plan in graph $\mathbf{PG}$. □

Having the concept of the planning graph defined, the so-called *planning problem* may be defined which works in the way that for a given initial state it should be proposed such a sequence of nodes from the planning graph that brings the initial state to the expected target state. Formally, the planning problem in the elementary version may be depicted in this way.

Problem. *The planning problem* **Input:**

- *planning graph* $\mathbf{PG} = (S, A, E)$*,*
- *initial state* s_i*,*
- *target state* s_t*.*

Output: *Plan* $p = (v_1, ..., v_k) \in PLAN(\mathbf{PG})$ *such that* $v_1 = s_i$ *and* $v_k = s_t$*.*

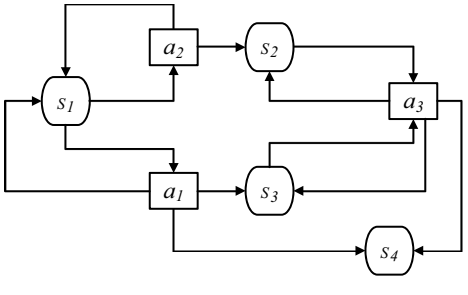


Fig. 40. An exemplary planning graph

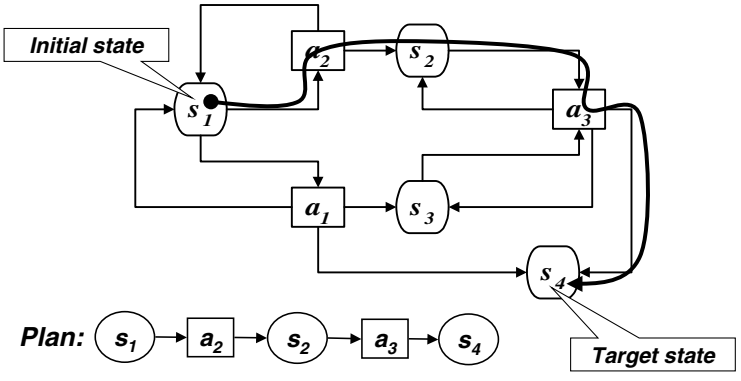


Fig. 41. The output for the planning problem

Fig. 41 presents a solution to the problem of finding a plan bringing state s_1 to state s_4 in the planning graph from Example 35.

The planning graph may be obtained through linking available planning rules, and in order to obtain a planning graph through linking planning rules belonging to a given family of planning rule F, the four following steps should be performed:

1. create a set of states S as a sum of all states which occur in the predecessors and successors of rules of family F,
2. create a set of actions A as a sum of all actions which occur in the rules of family F,
3. create a set of edges E as a sum of all pairs (s, a) for which there exists such a rule in family F that s is the predecessor of this rule and a is an action occurring in this rule,
4. add all the pairs (a, s) to set E for which there exists such a rule in family F that a is the action occurring in this rule and s occurs in the successor of this rule.

In Fig. 42 we present how the three following rules:

- $(s_1, a_1) \rightarrow s_1|s_2$,
- $(s_1, a_2) \rightarrow s_1|s_2$,
- $(s_2, a_1) \rightarrow s_1|s_2$,

may be linked to make a planning graph.

Let's notice that it exists an essential difference between the behavioral graph (see Definition 35 and Definition 53), and the planning graph (see Definition 55). There is one kind of nodes in the behavioral graph only, representing properties of behavior of complex objects during certain period (*e.g.*, time window). Whereas, there are the following two kinds of nodes in the planning graph, namely, states of complex objects (registered in a time point) and actions, that can be performed on complex objects during some period (*e.g.*, time window). Hence, the main

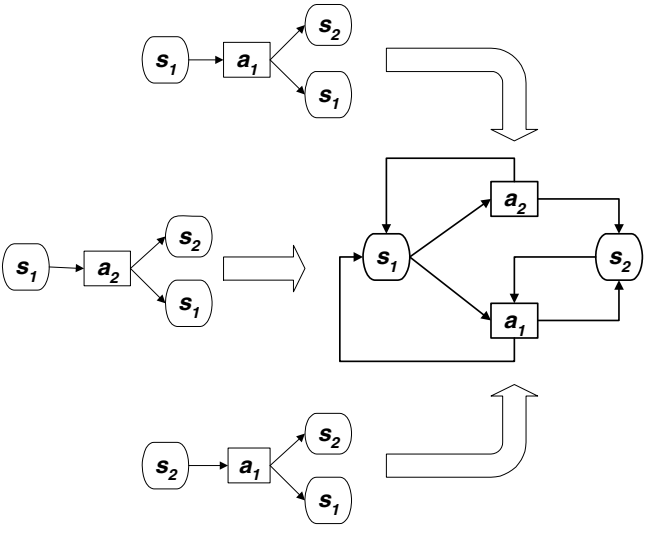


Fig. 42. From planning rules to a planning graph

application of behavioral graphs is to represent observed properties of complex objects, while the main application of planning graphs is to represent changes of object's parameters in the expected direction.

Similarly, there exists a certain similarity here between planning graphs known from literature and planning graphs defined in this paper. It works in the way that in both approaches there occur states, actions and links between them. However, the planning graph known from literature (see, *e.g.*, [70]) is constructed in order to plan the sequence of actions for the established initial state and its construction is aimed at this particular state. Moreover, the construction of this graph is layered and individual layers are connected with the next steps of the plan under construction. However, in this paper the planning graphs are constructed in order to depict the whole knowledge (all possible actions together with their results) concerning the behavior planning of complex objects. Apart from that, there is a difference in understanding states of complex objects (nodes of planning graphs). In approaches known from literature the state of a complex object may be read directly from the values of its parameters or from a simple analysis of dependencies between these values. However, in the approach presented in this paper the state of the complex object is described in a natural language with the help of complex concepts which require approximation (see the beginning of Subsection 7.3).

There also exists a similarity between the concept of the planning graph (see Definition 55) and C/E–systems well known from literature (see, *e.g.*, [350]). The similarity is that both graphs look very similar: the states from the planning graph correspond to the conditions from C/E–system and actions from the behavioral graphs correspond to the events from the C/E–system. However, it should be emphasized here that the interpretation of both graphs is significantly

different. In the case of C/E–systems the dynamics of the real system modeled by the net is based on the simulating an occurrence of an event but a given event might have taken place only when all the conditions from which the arches are led to a given event are satisfied. Apart from that, after the occurrence of a given event all conditions, to which the arches of a given event are transferred, are satisfied. It happens differently in the case of the planning graph. The action may be performed when the complex object is in one of the initial states of a given action, that is, in one of the states from which the arches are transferred to this action (complex object may not be simultaneously in more than one states). Similarly, after the performance of the action the complex object goes to one exit state of a given action, which also differentiates significantly the planning graph from C/E–systems. Besides, there is another important difference. In the case of C/E–systems, conditions have a local character, whilst in the case of the planning graph, states have a global character.

7.5 Identification of the Current State of Complex Objects

At the beginning of planning the behavior of the complex object based on a given planning graph, the initial state of this object should be determined. In other words, one of the states occurring in the planning graph should be located in which there is the complex object under examination. In this paper, each state of the planning graph is treated as a spatio-temporal complex concept and to recognize such a state we propose the two following approaches:

1. ask an expert from a given domain to indicate the appropriate state in which there is a given complex object,
2. treat the state as a temporal concept and use methods of temporal concept approximation described in Section 6.

The first of the above possibilities is very convenient, because it does not require the construction of any algorithms. It has, however, a very important drawback: the engagement of an expert in the functioning of the planning system may be too absorbing. However, in some cases the application of this method is possible and sensible. For example, in hospital conditions the current condition of a patient may be determined by an experienced doctor or negotiated by a group of experienced doctors at the established times of the day (*e.g.*, in the morning and in the evening), whereas through the remaining time of the day the treatment of the patient conducted by the doctor on duty may be supported by the planning system and its performance assumes the initial condition of the patient determined by a group of doctors and suggests further treatment.

The second possibility of recognition of the current state of the complex object is treating the state as a temporal concept and using methods of its approximation described in Section 6; and at the same time the interpretation of such a concept is slightly different from the one which appeared in Section 6. In Section 6, the temporal concept described the behavior of a complex object over a certain period of time (time window). Here, however, the temporal concept

represents the consequences of the complex object's behavior over a certain period of time, that is, the current state of a complex object (at a time point). We assume that to determine the state of a complex object at a time point, the observation of this object is necessary over a certain period of time (time window). For example, to determine the fact that the patient's condition is very serious, it is often not sufficient to determine what his current medical parameters are (*e.g.*, the results of a clinical interview or his/her laboratory results), but it is necessary to observe how the medical parameters of the patient have changed recently and how the patient's organism reacts to specific treatment methods. It may happen that the patient's medical parameters are very bad, but the application of the typical treatment method causes a sudden and permanent improvement. A crucial modification of methods of temporal concept approximation in the case of state approximation is the usage of information about actions performed on the complex object. This information may be easily introduced to these methods in the form of additional conditional attributes of the c-temporal information system.

Therefore, using methods from Section 6 a stratifying classifier may be constructed for each state, which for a given complex object, provides the degree to which this object belongs to a given state. Next, all these classifiers are linked in order to obtain a general aggregating classifier which recognizes the state of the complex object. Such an aggregating classifier is called *a state identifying classifier*. The performance of the state identifying classifier consists in its choosing such a state for the tested complex object that the stratifying classifier corresponding to this state provided the highest degree of membership for the tested complex object.

7.6 Language of Features of Paths of Planning Graphs

In the further part of the section, we construct information systems whose objects are the paths in the planning graphs and the attributes are the properties (features) of these paths. Therefore, currently we define the *FPPG*-language in which we express features of paths of planning graphs.

Definition 57 (*A language for defining features of paths in planning graphs*)**.** *Let* $\mathbf{PG} = (S, A, E)$ *be a planning graph and let* $\mathbb{N}$ *be a set of natural numbers. A language for defining features of paths in planning graphs (denoted by* $FPPG(\mathbf{PG})$ *or* $FPPG$*-language, when* $\mathbf{PG}$ *is fixed) is defined for the planning graph* $\mathbf{PG}$ *in the following way:*

- *the set* $AL_{FPPG}(\mathbf{PG}) = (2^S \setminus \emptyset) \cup (2^A \setminus \emptyset) \cup \mathbb{N} \cup (0,1] \cup \{ Exists, Each, Occurence, First, Last, Order \} \cup \{\neg, \vee, \wedge\}$ *is an alphabet of the language* $FPPG(\mathbf{PG})$*,*
- *atomic formulas of the language* $FPPG(\mathbf{PG})$ *are constructed in the following way:*
  1. *for any pair of non-empty sets* $X, Y \subseteq S$, $l, r \in \mathbb{N}$ *and* $t \in (0,1]$, *expressions of the form* $First(X,l,r)$, $Last(X,l,r)$, $Exists(X,l,r)$, $Each(X,l,r)$, $Occurence(X,l,r,t)$, $Order(X,Y,l,r)$ *are atomic formulas of the language* $FPPG(\mathbf{PG})$*,*

2. *for any pair of non-empty sets* $B, C \subseteq A$, $l, r \in \mathbb{N}$ *and* $t \in (0,1]$, *expressions of the form* $First(B,l,r)$, $Last(B,l,r)$, $Exists(B,l,r)$, $Each(B,l,r)$, $Occurence(B,l,r,t)$, $Order(B,C,l,r)$ *are atomic formulas of the language* $FPPG(\mathbf{PG})$,
3. *for any pair of non-empty sets* $X \subseteq S$ *and* $B \subseteq A$, $l, r \in \mathbb{N}$ *and* $t \in (0,1]$, *expressions of the form* $Order(X,B,l,r)$ *and* $Order(B,X,l,r)$ *are atomic formulas of the language* $FPPG(\mathbf{PG})$.

Currently, we determine the semantics of the language $FPPG(\mathbf{PG})$. Each formula of the language $FPPG(\mathbf{PG})$ is treated as the description of a set of paths belonging to the set $PATH(\mathbf{PG})$.

Definition 58. *Let* $\mathbf{PG} = (S, A, E)$ *be a planning graph. The semantics of the language FPPG is defined in the following way:*

1. *for any non-empty set* $X \subseteq S$, *numbers* $l, r \in \{1, ..., k\}$ *(where* $l < r$*) and* $t \in (0,1)$*:*
   - $|Exists(X,l,r)|_{FPPG(\mathbf{PG})} =$
   $$\{(v_1, ..., v_k) \in PATH(\mathbf{PG}) : \exists_{i \in \{l,...,r\}}\ v_i \in X\},$$
   - $|Each(X,l,r)|_{FPPG(\mathbf{PG})} =$
   $$\{(v_1, ..., v_k) \in PATH(\mathbf{PG}) : \forall_{i \in \{l,...,r\}}\ \text{if } v_i \in S \text{ then } v_i \in X\},$$
   - $|First(X,l,r)|_{FPPG(\mathbf{PG})} =$
   $$\{(v_1, ..., v_k) \in PATH(\mathbf{PG}) : v_l \in X\},$$
   - $|Last(X,l,r)|_{FPPG(\mathbf{PG})} =$
   $$\{(v_1, ..., v_k) \in PATH(\mathbf{PG}) : v_r \in X\},$$
   - $|Occurence(X,l,r,t)|_{FPPG(\mathbf{PG})} =$
   $$\{(v_1, ..., v_k) \in PATH(\mathbf{PG}) : \frac{card(\{i \in \{l, ..., r\} : v_i \in X\})}{card(\{i \in \{l, ..., r\} : v_i \in S\})} \geq t\},$$

2. *for any non-empty set* $B \subseteq A$, $l, r \in \{1, ..., k\}$ *(where* $l < r$*) and* $t \in (0,1)$*:*
   - $|Exists(X,l,r)|_{FPPG(\mathbf{PG})} =$
   $$\{(v_1, ..., v_k) \in PATH(\mathbf{PG}) : \exists_{i \in \{l,...,r\}}\ v_i \in B\},$$
   - $|Each(X,l,r)|_{FPPG(\mathbf{PG})} =$
   $$\{(v_1, ..., v_k) \in PATH(\mathbf{PG}) : \forall_{i \in \{l,...,r\}}\ \text{if } v_i \in A \text{ then } v_i \in B\},$$
   - $|First(X,l,r)|_{FPPG(\mathbf{PG})} =$
   $$\{(v_1, ..., v_k) \in PATH(\mathbf{PG}) : v_l \in B\},$$

– $|Last(X,l,r)|_{FPPG(\mathbf{PG})} =$

$$\{(v_1,...,v_k) \in PATH(\mathbf{PG}) : v_r \in B\},$$

– $|Occurence(X,l,r,t)|_{FPPG(\mathbf{PG})} =$

$$\{(v_1,...,v_k) \in PATH(\mathbf{PG}) : \frac{card(\{i \in \{l,...,r\} : v_i \in B\})}{card(\{i \in \{l,...,r\} : v_i \in A\})} \geq t\},$$

3. *for any sets* X, Y, B, C *(where* $X, Y \subseteq S$ *and* $B, C \subseteq A$*) and* $l, r \in \{1,...,k\}$ *(where* $l < r$*):*
   – $|Order(X,Y)|_{FPPG(\mathbf{PG})} =$

$$\{(v_1,...,v_k) \in PATH(\mathbf{PG}) : \exists_{i,j \in \{l,...,r\},\ i<j}\ v_i \in X\ \wedge\ v_j \in Y\},$$

   – $|Order(B,C)|_{FPPG(\mathbf{PG})} =$

$$\{(v_1,...,v_k) \in PATH(\mathbf{PG}) : \exists_{i,j \in \{l,...,r\},\ i<j}\ v_i \in B\ \wedge\ v_j \in C\},$$

   – $|Order(X,B)|_{FPPG(\mathbf{PG})} =$

$$\{(v_1,...,v_k) \in PATH(\mathbf{PG}) : \exists_{i,j \in \{l,...,r\},\ i<j}\ v_i \in X\ \wedge\ v_j \in B\},$$

   – $|Order(B,X)|_{FPPG(\mathbf{PG})} =$

$$\{(v_1,...,v_k) \in PATH(\mathbf{PG}) : \exists_{i,j \in \{l,...,r\},\ i<j}\ v_i \in B\ \wedge\ v_j \in X\}.$$

Below, we provide several examples of formulas of the language *FPPG* constructed for the planning graph from Example 35.

– Formula $First(\{s_2\},1,4)$ describes the path whose first state from the node number 1 to node number 4 is state s_2. This is for example path (s_2, a_3, s_3, a_3, s_3).
– Formula $Exists(\{s_2\},2,5)$ describes the path in which, from node number 2 to node number 5 there exists state s_2. This is for example path $(s_1,a_2,s_2,a_3,s_3,a_3)$.
– Formula $Exists(\{s_2,s_3\},2,7)$ describes the path in which, from node number 2 to node number 5 there exists state s_2 or state s_3 or both of them. There are for example paths: $(s_1,a_2,s_2,a_3,s_3,a_3,s_2,a_3)$, $(s_1,a_1,s_3,a_3,s_3,a_3,s_3,a_3,s_3)$ or $(s_1,a_2,s_1,a_2,s_2,a_3,s_2,a_3,s_2)$.
– Formula $Each(\{a_3\},1,5)$ describes the path, in which from node number 1 to node number 5 there is only action a_3. This is for example path (s_2,a_3,s_3,a_3,s_2).
– Formula $Occurence(\{s_2\},3,7,0.6)$ describes the path, in which from node number 3 to node number 7, at least 60% of all states constitute state s_2. This is for example path number $(s_1,a_2,s_2,a_3,s_3,a_3,s_2,a_3,s_4)$.
– Formula $Order(\{a_2\},\{a_3\},2,7)$ describes the path, in which from node number 2 to node number 7, firstly action a_2 is performed and then action a_3. This is for example path $(s_1,a_2,s_2,a_4,s_3,a_3,s_3)$.

– Formula $Order(\{a_2\}, \{s_3\}, 2, 7)$ describes the path, in which from node number 2 to node number 6, firstly action a_2 is performed and then state s_3 is observed. This is for example path $(s_1, a_2, s_1, a_4, s_2, a_3, s_3)$.

Patterns of the language $FPPG$ may be applied in defining the path properties in the planning graph. Owing to this each path in the planning graph may be represented using the values of its features. It enables approximation of the concepts determined in the set of paths with the help of classifiers (see Section 7.7).

7.7 Resolving Table

As we mentioned before, the output for the planing problem for a single complex object is a path in the planning graph from the initial node-state to the expected (target) node-state (see Fig. 41).

However, in the planning graphs there often appears a problem of nondeterministic choice of one of the actions possible to apply in a given state. For example, in the graph from Fig. 40 action a_1 or a_2 may be performed in state s_1. Apart from that, there also occurs the uncertainty concerning the choice of the state after applying the action. For example, in the graph from Fig. 40 in state s_1 after applying action a_2 the complex object may change to state s_2 or remain in state s_1. That is why there may be usually many solutions to a given planning problem consisting in going from the initial state to the target state on different paths in the graph. Assuming that we always treat all actions and states in the same way and the choice of actions in a given state and the choice of the state after applying the action is random or directed using a heuristic function onto the target state, then to solve the planning problem one may use planning methods known from literature such as: *forward search*, *backward search* or *heuristic for state-space search*, which in fact would consist in searching the planning graph (see, *e.g.*, [70, 76]).

However, in practice there often occurs such a situation that the automatically generated plan must be compatible with the plan suggested by an expert (*e.g.*, the treatment plan should be compatible with the plan suggested by human experts from a medical clinic). Therefore, it is strongly recommended that the method of the verification and evaluation of generated plans should be based on the similarity between the generated plan and the plan proposed by human experts (see Section 7.21). Apart from that we need tools which during the generating the automatic plan may be used to solve the conflicts occurring between actions which may be used at a given planning stage in such a way as to make this choice compatible with domain knowledge provided by the experts. Such tools should work on the basis of the current state of the tested complex object and on the basis of information about the previously observed states of the tested complex object as well as on the basis of information about actions performed earlier on this object. In other words, while choosing the action needed to perform in a given state of the complex object one has to use information about the sequence of states and actions which have led the object under examination

to the current state. In terms of the planning graph such information is simply a path of an established length k in this graph, which ends in the current state of the complex object under examination. In practice, for a given state from the planning graph there exist very many different paths which end exactly with that state. That is why in constructing tools allowing choosing actions on the basis of the path in the planning graph preceding the current state of the complex object, one should use the available data sets gathered during the observation of the complex dynamical systems. So far, we have used temporal information systems to represent such data sets. However, in the temporal information system the actions performed on the complex objects are not represented in an overt way although they may obviously be represented using the established attribute. Therefore, we define a certain particular type of a temporal information system which is called *a temporal information system with actions.*

Definition 59 (*A temporal information system with actions***).** *A temporal information system with actions is a seven-element tuple:*

$$\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t}, a_c),$$

where a tuple $(U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t})$ *is the temporal information system and* $a_c \in A$ *is a distinguished attribute in the set* A *different from attributes* a_{id} *and* a_t*, which is an attribute identifying the action performed at a time point represented by a given object of system* $\mathbf{T}$*.*

Thus, in the temporal information system with actions for each object of this system $u \in U$ (at a time point of this system) the action performed at this point is remembered and it is action $a_c(u)$.

Let us notice that we consider here one action performed on the complex object at a given time point. However, it seems that in practical applications, at a given time point a sequence of actions could be performed synchronically on the complex object. However, they are always actions chosen from the established set of single actions. Therefore, the action performed at a given time point may be treated as a subset of the established set of single actions. For example, if $M = \{m_1, m_2, m_3, m_4\}$ is a set of medicaments which may be used during the treatment of a certain illness, then an example of a specific action of the patient's treatment is action $\{m_1, m_3\}$ which consists in giving the patient medicine m_1 and m_2 simultaneously. Apparently, other actions are also possible. For example, action $\{m_1, m_2, m_3, m_4\}$ is the action of giving all possible medicaments. While, action $\{\ \}$ (empty set) is the action of not giving any medicament.

For temporal information system with actions, one can speak about states in which there are individual objects of this system. We mean here the states specified in a planning graph. However, the identification of the state of a given time point requires application of the classifier constructed specially for this purpose (see Section 7.5).

If it is possible to identify the current state of the examined object and the actions performed at individual time points are known, then it is possible to represent the history of the examined complex object arranged as a path from the planning graph observed in a given temporal information system.

Definition 60. *Let us assume that:*

- $\mathbf{T} = (U,\ A,\ a_{id},\ \leq_{a_{id}},\ a_t,\ \leq_{a_t},\ a_c)$ *is a temporal information system with actions,*
- $\mathbf{PG} = (S, A, E)$ *is a planning graph,*
- $s \in S$ *is a fixed state,*
- k *is a fixed plan length in the planning graph* $\mathbf{PG}$.

1. *Any plan* $p = (v_1, ..., v_k) \in PLAN(\mathbf{PG}, k)$ *is called a plan occurring in system* $\mathbf{T}$ *if exists such a time window* $W = (u_1, ..., u_l) \in TW(\mathbf{T}, l)$ *such that* $l = \frac{k-1}{2} + 1$*) and at the successive time points of this window there occur states* $v_1, v_3, v_5..., v_k$ *and at the time points from* $u_1, ..., u_{l-1}$ *actions* $v_2, v_4, ..., v_{k-1}$ *are performed respectively. For a given plan* p *such a time window is called a time window of this plan.*
2. *A set of all time windows of a given plan* p *in system* $\mathbf{T}$ *is denoted by* $TW(\mathbf{T}, p)$.
3. *A set of all plans, occurring in system* $\mathbf{T}$ *of the length* k *is denoted by* $DPLAN(\mathbf{T}, \mathbf{PG}, k)$.
4. *A set of all plans, occurring in system* $\mathbf{T}$ *and ending with state* s *and of the length* k*, is denoted by* $DPLAN(\mathbf{T}, \mathbf{PG}, s, k)$.

Let us notice that set $DPLAN$ may be determined in such a way that firstly a set of all time windows for a given temporal information system is determined (see Section 6.7) and then these windows are treated as potential time windows of plans from the set $DPLAN$.

Using the planning graph paths occurring in data, decision tables may be constructed and what follows there may also be constructed classifiers which allow solving conflicts between actions which may be performed in a given state for the complex object. Let us notice that the classifiers mentioned above also allow determining what the state of the complex object will be after performing the chosen action. The starting point for the construction of such classifiers is *a resolving table* (see Fig. 43).

Definition 61 (*A resolving table***).** *Let us assume that:*

- $\mathbf{T} = (U,\ A,\ a_{id},\ \leq_{a_{id}},\ a_t,\ \leq_{a_t},\ a_c)$ *is a temporal information system with actions,*
- $\mathbf{PG} = (S, A, E)$ *is a planning graph,*
- $s \in S$ *is a fixed state,*
- k *is a fixed length of path,*
- $\Phi = \{\phi_1, ..., \phi_m\} \subseteq FPPG(\mathbf{PG})$ *is a family of formulas defined by experts,*
- $\mathcal{P}_{FPPG} = (U, \Phi, \models_{FPPG})$ *is a property system, where*

$$U = DPLAN(\mathbf{T}, \mathbf{PG}, s, k)$$

and the satisfiability relation $\models_{FPPG} \subseteq U \times \Phi$ *is defined in the following way:* $\forall p = (v_1, ..., v_k) \in U$ *and* $\phi \in \Phi$:

$$u \models_{FPPG} \phi \quad \Leftrightarrow \quad Subpath(p, 1, k-2) \ \models_{FPPG(\mathbf{PG})} \ \phi.$$

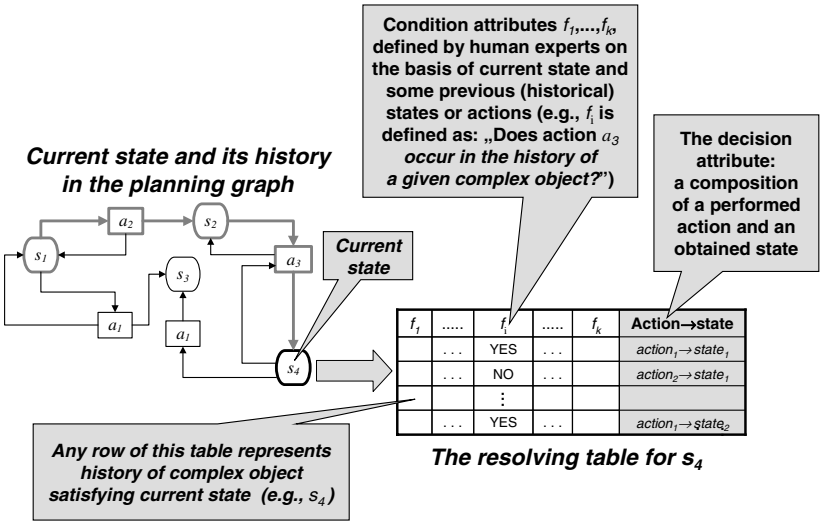


Fig. 43. The scheme of construction of the resolving table for a given state

A resolving table for the state s from the planning graph **PG** *constructed using the length of path k is a decision table* $\mathbf{RT}(s,k) = (U, A, d)$*, where:*

- (U, A) *is an information system defined be the property system* $\mathcal{P}_{FPPG}$*,*
- d *is a decision attribute, where values of the attribute d, being ordered pairs of the form* $(action, state)$*, are computed in the following way:*

$$\forall p = (v_1, ..., v_k) \in U : d_i(p) = (v_{k-1}, v_k).$$

The objects of this table are paths in the planning graph observed in data, starting and ending with a state. Thus, they are plans. Conditional attributes describe the properties of these paths excluding the last two nodes of each path and they are defined on the basis of the formulas of the language $FPPG$ provided by the expert, whereas the values of the decision attribute are arrangement of the action performed after the last but one state on the path and the last state on the path.

Example 36. In Fig. 44, the planning graph for the RDS treatment is shown. For each state occurring in the graph, with the use of available data sets concerning the treatment of respiratory failure, resolving tables may be constructed. Conditional attributes of these tables are created with the use of patterns defined by experts in language $FPPG$. Below, we present examples of typical patterns of this type:

1. the first (last) state in the plan is the *RDS excluded* (*RDS with mild hypoxemia*, *RDS with severe hypoxemia*, *RDS with very severe hypoxemia*, *RDS with mild or severe hypoxemia*, *RDS with severe or very severe hypoxemia*) state,

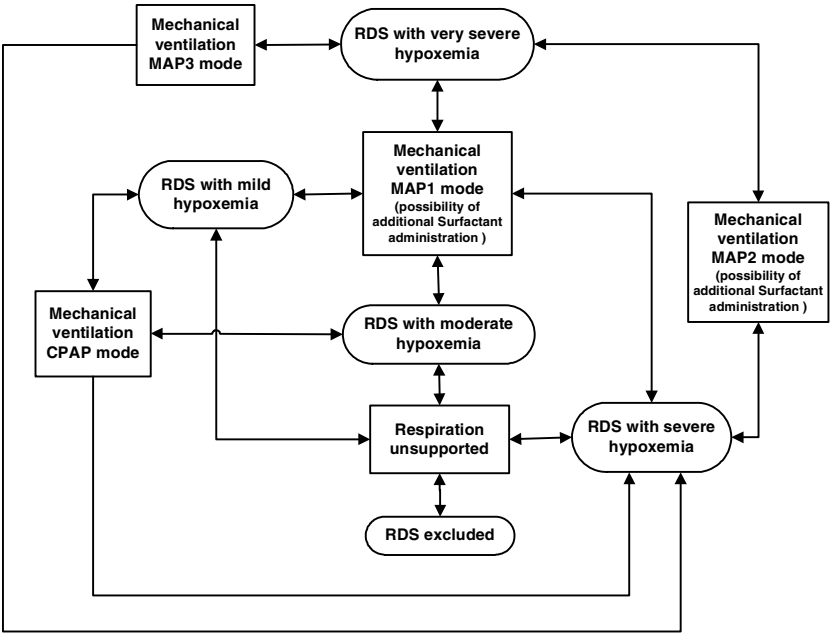


Fig. 44. A planning graph for the treatment of infants during the RDS

2. the first (last) action in the plan is the *Mechanical ventilation MAP1 mode*[3] (*Mechanical ventilation MAP2 mode*, *Mechanical ventilation MAP3 mode*, *Mechanical ventilation CPAP mode*[4], *Respiration unsupported*, *Mechanical ventilation MAP2 or MAP3 mode*) action,

[3] Invasive mechanical ventilation is a method to mechanically assist or replace spontaneous breathing when patients cannot do so on their own. It is administered after an invasive intubation, a procedure wherein an endotracheal or tracheostomy tube is inserted into the airway, through which air is directly delivered under pressure (see [328] for more details). It could be simplify, that *mean airway pressure* (MAP) delivered by mechanical device is proportional to severity of respiratory failure. For purpose of our experiments mechanical ventilation was divided into three following modes:

- *MAP1* - airway pressure lower than 10 cm H2O (low-intensity ventilation),
- *MAP2* - airway pressure 10-16 cm H2O (middling-intensity ventilation),
- *MAP3* - airway pressure higher than 16 cm H2O (high-intensity ventilation).

[4] *CPAP* (continuous positive airway pressure) - a method of non-invasive ventilation delivering a stream of compressed air via a hose to a nasal pillow, nose mask or full-face mask, splinting the airway (keeping it open under air pressure). This is a gentle type of respiratory ventilation, which can prevent the need for endotracheal intubation, or allow earlier extubation of critically ill patients (see [328] for more details).

3. in the plan there occurs the *RDS excluded* (*RDS with mild hypoxemia*, *RDS with severe hypoxemia*, *RDS with very severe hypoxemia*, *RDS with mild or severe hypoxemia*, *RDS with severe or very severe hypoxemia*) state,
4. in the plan there occurs the *Mechanical ventilation MAP1 mode* (*Mechanical ventilation MAP2 mode*, *Mechanical ventilation MAP3 mode*, *Mechanical ventilation CPAP mode*, *Respiration unsupported*, *Mechanical ventilation MAP1 or CPAP mode*) action,
5. in the plan there occurs only the *RDS excluded* (*RDS with mild hypoxemia*, *RDS with severe hypoxemia*, *RDS with very severe hypoxemia*, *RDS with mild or severe hypoxemia*, *RDS with severe or very severe hypoxemia*) state,
6. in the plan there occurs only the *Mechanical ventilation MAP1 mode* (*Mechanical ventilation MAP2 mode*, *Mechanical ventilation MAP3 mode*, *Mechanical ventilation CPAP mode*, *Respiration unsupported*) action,
7. the *RDS with very severe hypoxemia* state occurs in the 70% of states of the plan,
8. from the middle of the plan to its end in 80% of the states there occurs the *RDS excluded* state,
9. if there occurs the *RDS with mild hypoxemia* state in the plan then the *RDS with severe hypoxemia* state occurs in the further part of this plan,
10. if there occurs the *Mechanical ventilation MAP3 mode* action in the plan then the *Mechanical ventilation MAP2 mode* action occurs in the further part of this plan,
11. if there occurs the *Mechanical ventilation MAP2 mode* action in the plan then *RDS with mild hypoxemia* state occurs in the further part of this plan.

□

A classifier may be constructed for the resolving table which we call *a resolving classifier*.

Definition 62 (*A resolving classifier*)**.** *Let us assume that:*

- $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t}, a_c)$ *is a temporal information system with actions,*
- $\mathbf{PG} = (S, A, E)$ *is a planning graph,*
- $s \in S$ *is a fixed state,*
- k *is a fixed length of path,*
- $\mathbf{RT}(s,k) = (U, A, d)$ *is a resolving table for the fixed state* s *from the planning graph* $\mathbf{PG}$ *constructed using the length of path* k*.*

1. *Each stratifying classifier constructed for table* $\mathbf{RT}(s,k)$ *is called a resolving classifier and is denoted in general by* $\mu_{\mathbf{RT}(s,k)}$*. This classifier serves the classification of paths which belong to the* $DPLAN(\mathbf{T}, \mathbf{PG}, s, k-2)$*.*
2. *For any* $p \in DPLAN(\mathbf{T}, \mathbf{PG}, s, k-2)$ *and resolving classifier* $\mu_{\mathbf{RT}(s,k)}$ *by* $PairList(\mu_{\mathbf{RC}(s,k)}(p))$ *we denote a list of pairs* $(action, state)$ *which the classifier* $\mu_{\mathbf{RC}(s,k)}$ *returns to the path* p *ordered in a decreasing order in relation to weigh values proposed by the classifier for all pairs, and at the same time only pairs with non-zero weight are returned.*

3. *If L is a $PairList(\mu_{\mathbf{RC}(s,k)}(p))$, then the i-th pair of this list is marked as $L[i]$. The first element of the pair $L[i]$ (action) is marked as $L[i].action$, whereas the second element of this pair (state) is marked as $L[i].state$.*

Such resolving classifiers can be constructed for all states, i.e., for all associated resolving tables. In addition, these classifiers make it possible to obtain a list of actions and states after usage of actions with their weights in descending order. This is possible using the stratifying classifier.

7.8 Algorithms of Automated Planning for Complex Objects

In the present subsection, we provide three algorithms of automated planning of the complex object behavior. The first one determines one plan of established length starting with the established initial state, and at the same time the final (target) state is not established.

The second algorithm determines the plan starting with the established initial state and ending with the established final state. The length of the generated plan is limited from the top by the established constant value.

The third algorithm, however, determines the list of plans starting with the established (for all plans) initial states, and at the same time the length of the generated plans is established. Similarly to the second algorithm, the length of the generated plan is limited from the top by the established constant value.

The first of the algorithms mentioned above is similar to the algorithm *Forward-search* known from literature (see, *e.g.*, [70, 76]), but instead of choosing randomly the actions to perform in a given state the algorithm goes to the next state on the basis of the decision obtained from the classifier solving conflicts between actions in a given state. Therefore, this algorithm is called *Expert forward search* (see Algorithm 7.1).

However, we assume that during execution of algorithms presented in this section the following elements should by available:

- a planning graph $\mathbf{PG} = (S, A, E)$ for complex objects of a fixed type T,
- a fixed length of path k in the planning graph $\mathbf{PG}$,
- a resolving classifiers $\mu_{\mathbf{RC}(s,k)}$ for all $s \in S$.

The Algorithm 7.1 starts the planning from path $h \in DPLAN(\mathbf{T},\mathbf{PG},k)$ which describes the previous states of the complex object and the actions applied for this object. Next, using the resolving classifier $\mu_{\mathbf{RC}(s,k)}$, the most appropriate pairs: *state+action* are generated in the next iterations until the plan of the expected length is obtained.

If it is assumed that each of classifiers $\mu_{\mathbf{RC}(s,k)}$ can classify paths for each $s \in S$ within the time of order $O(C)$, where C is a certain constant, then the time complexity of the Algorithm 7.1 is of order $O(n)$, where n is the length of the generated plan.

The Algorithm 7.1 is, on the one hand very fast, but on the other its final result does not always comply with our expectations. For example, in planning for a single complex object we usually wish our planning algorithm to find a

Algorithm 7.1. Expert forward search (EFS)

Input:
- path $h = (h_1, ..., h_k)$ in the planning graph **PG** representing history of a given complex object, that is finished by an initial state to start of automated planning,
- expected length l_p of generated plan.

Output: The plan p generated for the given complex object

```
1  Procedure EFS(h, l_p)
2  begin
3      p := "empty plan"
4      s := GetLastElementFrom(h)
5      p := p + s // Add s to the end of the plan p
6      while length(p) < l_p do
7          L := PairList(μ_RC(s,k)(h))
8          p := p + L[1].action + L[1].state
9          RemoveFirstTwoElementsFrom(h)
10         h := h + L[1].action + L[1].state
11         s := L[1].state;
12     end
13     return p
14 end
```

plan which leads the complex object to the established target state. Meanwhile, the final state of the planning using the algorithm EFS depends on classifier $\mu_{\mathbf{RC}(s,k)}$ and cannot be imposed. Therefore, we define the algorithm $EEFS$ which determines the plan starting with the established initial state and ending with the established final state (see Algorithm 7.2).

The Algorithm 7.2 works in such a way that at the stage of planning of a single action, its different variants are taken into consideration which may be performed in a given state. However, for regulation of computational time duration limitation, the value *ActionLimit* is used, that is, limitation of the number of actions which may be performed in a given state (see line 7). Thus, the classifier $\mu_{\mathbf{RC}(s,k)}$ returns the list of pairs ($action + state$) sorted decreasingly in relation to the weights obtained from classification, the actions most recommended by classifier $\mu_{\mathbf{RC}(s,k)}$ are always taken into consideration. In this way the algorithm constructs a certain type of a plan tree whose root is the initial state and the leaves are the states after performing the individual variants of the plan. If, during construction of this tree the final state appears, then the work of the algorithm is ended and as a solution a sequence of states and actions is returned which starts in the tree root and ends with the final state that is found. If, during the construction of the plan tree the algorithm does not encounter the

Algorithm 7.2. Exhaustive expert forward search ($EEFS$)

Input:
- path $h = (h_1, ..., h_k)$ in the planning graph **PG** representing history of a given complex object, that is finished by an initial state to start of automated planning,
- target state of planning s_t,
- maximal length of generated plan l_p.

Output: The plan p generated for the investigated complex object, ended by the state s_t

```
1  Procedure EEFS(h, s_t, l_p)
2  begin
3      p := "empty plan"
4      s := GetLastElementFrom(h)
5      p := p + s // Add s to the end of the plan p
6      L := PairList(μ_RC(s,k)(h))
7      for i = 1 to ActionLimit do
8          p_1 := Copy(p)
9          p_1 := p_1 + L[i].action + L[i].state
10         if (L[i].state = s_t) then return p_1
11         if (l_p > 1) then
12             h_1 := Copy(h)
13             RemoveFirstTwoElementsFrom(h_1)
14             h_1 := h_1 + L[i].action + L[1].state
15             p_2 := EEFS(h_1, s_t, l_p − 1)
16             if (p_2 is not empty ) then
17                 return p + p_2
18             end
19         end
20     end
21     return "empty plan"
22 end
```

final state, then an empty plan is returned which means that the algorithm has not found a solution.

The procedure $EEFS$ from the Algorithm 7.2 is recurrent. It is easy to notice that its time complexity is determined by a recurrent equation:

$$T(n) = \begin{cases} A \cdot m + B & \text{for } n = 1 \\ m \cdot T(n-1) + C \cdot m + B & \text{for } n > 1, \end{cases}$$

where A, B and C are certain constants, n is the duration of the plan under construction and m is limitation $ActionLimit$, that is, limitation of the number

Algorithm 7.3. Full exhaustive expert forward search ($FEEFS$)

Input:
- path $h = (h_1, ..., h_k)$ in the planning graph **PG** representing history of a given complex object, that is finished by an initial state to start of automated planning,
- length of generated plan l_p.

Output: The list of plans for the investigated complex object

```
1  Procedure FEEFS(h, l_p)
2  begin
3      plist := "empty list of plans"
4      p := "empty plan"
5      s := GetLastElementFrom(h)
6      p := p + s // Add s to the end of the plan p
7      L := PairList(μ_RC(s,k)(h))
8      for i = 1 to ActionLimit do
9          p_1 := Copy(p)
10         p_1 := p_1 + L[i].action + L[i].state
11         if (l_p > 1) then
12             h_1 := Copy(h)
13             RemoveFirstTwoElementsFrom(h_1)
14             h_1 := h_1 + L[i].action + L[i].state
15             plist_1 := FEEFS(h_1, l_p − 1)
16             for j := 1 to Length(plist_1) do
17                 p_2 := plist_1[j]
18                 p_3 := p_1 + p_2
19                 Add plan p_3 to the end of the list plist
20             end
21         else
22             Add plan p_1 to the end of the list plist
23         end
24     end
25     return plist
26 end
```

of actions which may be performed in a given state. The solution of the above recurrent equation is the following:

$$T(n) = A \cdot m^n + B \cdot m^{n-1} + C \cdot m \cdot \frac{m^{n-1} - 1}{m - 1} + B \cdot \frac{m^{n-1} - 1}{m - 1}.$$

Thus, the pessimistic time complexity of the procedure $EEFS$ is of order $O(m^n)$. Such a high pessimistic time complexity means that the effective application of this algorithm for non-trivially small n and m is practically impossible.

Therefore, it may be applied only in constructing very short plans with very small values m.

In the task of constructing a plan executing a meta-action for a structured object another planning algorithm for a single object is necessary (see Section 7.15). Namely, in this case the planning target state is also not known, but one has to generate all sensible (compatible with domain knowledge) plans of a given duration for a given complex object. Therefore, we define the algorithm $FEEFS$ (see Algorithm 7.3).

It is easy to notice that the analysis of time complexity of the Algorithm 7.3 is very similar to the case of Algorithm 7.2. Therefore, the pessimistic time complexity of the $FEEFS$ is of order $O(m^n)$ where n is the duration of the plan under construction and m is limitation of the number of actions which may be performed in a given state. This means, that similarly to the case of algorithm $EEFS$ the effective application of algorithm $FEEFS$ for non-trivially small n and m is practically impossible. Therefore, this algorithm is used only for construction of short plans for the need of planning of single meta-actions (see Section 7.15).

7.9 Partial Reconstruction of Plan

Having constructed the plan for a complex object, its execution may take place. For example, let us assume that for a certain complex object the plan (s_1, a_1, ..., a_{i-1}, s_i, a_i, ..., s_n, a_n, s_{n+1}) was constructed which consists of n actions $a_1, ..., a_n$ and $n+1$ states $s_1, ..., s_{n+1}$. The initial state in this plan is state s_1 and the target state is the state s_{n+1}. The execution of this plans works in such a way that after having identified the current state of the complex object as state s_1, actions from the plan are performed successively, with the changing states of object, until we reach target state s_{n+1}. However, in practice it is not always possible to execute the whole plan. It may, happen that during the execution of the plan such a state of an object appeared that is not compatible with the state proposed by the plan. For example, let us assume that s'_i is such a state which appeared instead of state s_i (see Fig. 45). Then, a question arises whether the execution of the plan should be continued or whether it should be reconstructed (changed). If state s'_i differs slightly from state s_i, then maybe the execution of the current state may be continued. If, however, state s'_i differs significantly from state s_i, then the current plan has to be reconstructed. It would seem that the simplest way to reconstruct a plan is to construct a new one, which starts in state s'_i and ends in target state s_{n+1}. Such a method of reconstruction we call a *total reconstruction*. However, in practical applications a total reconstruction may turn out to be too costly in terms of computation. Therefore, we propose a different method of plan reconstruction which is called *a partial reconstruction*. It consists in constructing a short so-called *repair plan* which brings the complex object to such state s_j that appears in the current state between the states $s_i, ..., s_{n+1}$. On the basis of the repair plan the reconstruction of the current plan is carried out by replacing its fragment beginning at s_i and ending at state s_j with the repair plan.

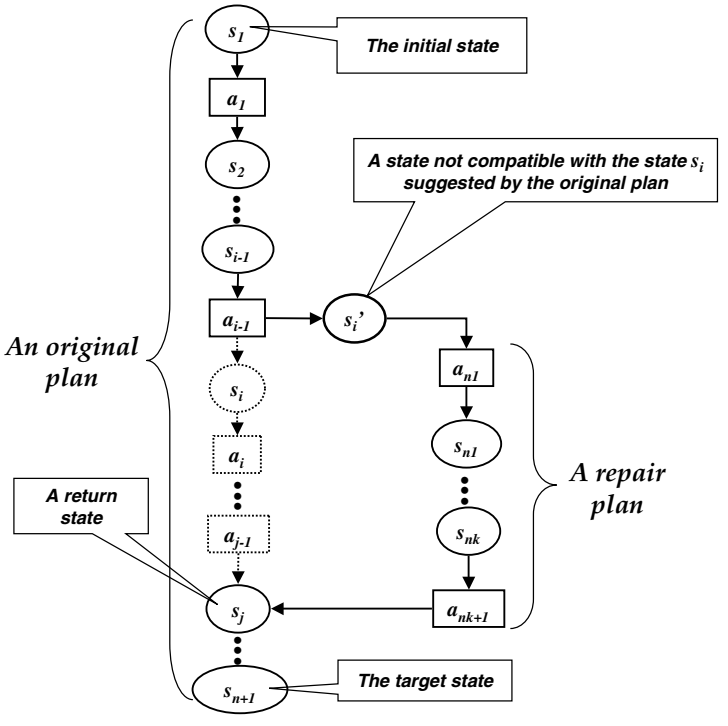


Fig. 45. The partial reconstruction of a plan

Of course the shorter the repair plan is, the more effective a partial reconstruction is in terms of time. If, however, state s_i' differs significantly from state s_i, then finding a short repair plan is impossible and a total reconstruction is the only solution.

We still have to face the problem of estimating the degree to which two states are different. It needs to be done in order to enable the determination when two states differ slightly, differ significantly or differ very much one from another.

There is yet another problem lying in the fact that the difference between two states in the context of plan execution depends not only on those states but also on the context in which those two states are compared, that is, on the fragment of the plan that has been carried out so far, together with the states that have appeared during the current plan execution (these states might have slightly differed from the states described in the plan). Therefore, the estimation of the difference between the plans should be made on the basis of the history of both states (the states and actions performed on the complex object over the period of time preceding the examination of the difference). Formally, the degree to which two states differ can be expressed with the help of the so-called *a function of dissimilarity between of states*.

Definition 63. *Let* $\mathbf{PG} = (S, A, E)$ *is a planning graph for complex objects of a fixed type* T *and* k *is a fixed length of paths in the graph* $\mathbf{PG}$*. Each function:*

$$DISM_{\mathbf{PG}} : PATH(\mathbf{PG}, k) \times PATH(\mathbf{PG}, k) \longrightarrow \{high, moderate, low\}$$

is called a function of dissimilarity between states from the planning graph $\mathbf{PG}$*.*

The values of the function $DISM_{\mathbf{PG}}$ which belong to set $\{high, moderate, low\}$ are proposed by the expert on the basis of domain knowledge. Value *high* means a high dissimilarity between states, value *moderate* means a moderate dissimilarity between states and value *low means* a low dissimilarity between states.

The definition of a specific function of dissimilarity between states may be given in an overt form, that is, using an expression calculating the value of dissimilarity function. It often happens, however, that experts from a given field are not able to present such an expression and limit themselves to presenting a set of examples of the values of that function, that is, a set of pairs of paths ended with compared states, labelled with the value of the dissimilarity function between states. In this case defining the dissimilarity function requires its approximation using a classifier; and at the same time to define the features of the paths preceding the compared states one may use a family of concepts of a specific ontology constructed for the comparison of the paths. The classifier approximating the function of the dissimilarity between states is called a classifier of dissimilarity between states.

Definition 64. *Let us assume that:*

- $\mathbf{PG} = (S, A, E)$ *is a planning graph for complex objects of a fixed type* T*,*
- k *is a fixed length of path in the planning graph* $\mathbf{PG}$*,*
- *a family of concepts* $C_1, ..., C_m \subseteq PATH(\mathbf{PG}, k) \times PATH(\mathbf{PG}, k)$*, which have been defined by experts in order to describe difference aspects of similarity between plans,*
- *a function of dissimilarity between states* $DISM_{\mathbf{PG}}$*.*

1. *A table of dissimilarity between states from the planning graph* $\mathbf{PG}$ *is a decision table* $\mathbf{DIT}_{\mathbf{PG}} = (U, A, d)$*, where:*
   - $U \subseteq PATH(\mathbf{PG}, k) \times PATH(\mathbf{PG}, k)$*,*
   - $A = \{a_1, ..., a_m\}$ *is a set of attributes created on the basis of concepts* $C_1, ..., C_m$*, where for any* $i \in \{1, ..., m\}$ *values of* a_i *are computed in the following way:*

$$\forall (p_1, p_2) \in U : a_i\left((p_1, p_2)\right) = \begin{cases} 1 \text{ if } (p_1, p_2) \in C_i \\ 0 \text{ otherwise} \end{cases},$$

   - d *is a decision attribute, where values of the attribute* d *are computed in the following way:*

$$\forall (p_1, p_2) \in U : d(\ (p_1, p_2)\ ) = DISM_{\mathbf{PG}}(\ (p_1, p_2)\ ).$$

2. *If* $\mathbf{DIT_{PG}} = (U, A, d)$ *is a table of dissimilarity between states from the graph* $\mathbf{PG}$*, then each classifier for the table* $\mathbf{DIT_{PG}}$ *is called a classifier of dissimilarity between states from the graph* $\mathbf{PG}$ *and is denoted in general by* $\mu_{DIT(\mathbf{PG})}$.

Let us notice that not all possible pairs of paths from the set $PATH(\mathbf{PG}, k) \times PATH(\mathbf{PG}, k)$ occur in the table of dissimilarity between states from the planning graph, but only a certain chosen subset of this set. In practice, this limitation is needed because the number of pairs of product $PATH(\mathbf{PG}, k) \times PATH(\mathbf{PG}, k)$ may be so large that the expert is not able to provide all values of decision attribute d for them. That is why in the table of dissimilarity between states there are usually only pairs chosen by an expert, which represent typical cases of determining the function of dissimilarity between states which may be generalized using a classifier.

Now, we may present the algorithm simulating the execution of the plan which foresees the reconstruction of the plan during its execution (see Algorithm 7.4).

However, we assume that during execution of algorithms presented in this section the following elements should by available:

- a planning graph $\mathbf{PG} = (S, A, E)$ for complex objects of a fixed type T,
- a fixed length of path k in the planning graph $\mathbf{PG}$,
- resolving classifiers $\mu_{\mathbf{RC}(s,k)}$ for all $s \in S$,
- a classifier of dissimilarity $\mu_{DIT(\mathbf{PG})}$ between states from the planning graph $\mathbf{PG}$.

The Algorithm 7.4 simulates the execution of the plan found earlier for the complex object. The simulation is performed based on the procedure *Simulate* which on the input takes the history of the current state together with its description of the current state and the action which is to be performed, and on the output the algorithm returns the state which is the effect of this action's application. Although it is possible to imagine this type of procedure as a part of a simulator of the behavior of the complex object (*e.g.*, a traffic simulator, illness development simulator), in this paper by this procedure we mean the changes in the real complex dynamical system which may be triggered by performing particular actions (*e.g.*, changes of the location of the vehicle, changes in the patient's states during treatment et.)

The Algorithm 7.4 uses the reconstruction procedure. Therefore, we present a plan reconstruction algorithm (see Algorithm 7.5).

The Algorithm 7.5 tries to find a short repair plan p_2 (not longer than l_p), which leads the initial state of the reconstruction (the last state in history h) to a state occurring in plan p_1 starting with position *pos* until reaching position $pos + 2 \cdot (d_r - 1)$. The maximum depth of reconstruction d_r is, thus, the number of states in plan p_1 (starting with the state in position *pos*), which the algorithm tries to reach with the help of the repair plan. The repair plan is searched with algorithm $EEFS$ (see Section 7.8), although it is possible to apply other planning algorithms that have at least two following parameters: the target state and the maximum duration of the created plan.

Algorithm 7.4. The simulation of the plan with reconstruction

Input:
- path $h = (h_1, ..., h_k)$ in the planning graph **PG** representing history of a given complex object, that is finished by an initial state to start of simulation,
- plan p generated for a given complex object,
- maximal depth d_r of the plan reconstruction,
- maximal length l_p of a repair plan during the reconstruction.

Output: The executed plan p

```
1  begin
2      if (length(p) < 3) then
3          return "plan p is too short for execution"
4      end
5      h_s := Copy(h); h_p := Copy(h)
6      i := 3
7      while (i < length(p)) do
8          s := Simulate(h_s, p[i − 1])
9          RemoveFirstTwoElementsFrom(h_s)
10         h_s := h_s + p[i − 1] + s
11         RemoveFirstTwoElementsFrom(h_p)
12         h_p := h_p + p[i − 1] + p[i]
13         if (s ≠ p[i]) then
14             dism := μ_DIT(PG)(h_s, h_p)
15             if (dism is "high") then
16                 return "the total reconstruction is necessary"
17             end
18             if (dism is not "low")) then
19                 p' := Reconstruction(h_s, p, i, d_r, l_p)
20                 if (p' is empty) then
21                     return "the total reconstruction is necessary"
22                 end
23                 p := p'
24             end
25         end
26         i := i + 2 // Go to the next state
27     end
28 end
```

The computational complexity of the Algorithm 7.5 depends linearly on the complexity of algorithm $EEFS$. However, in practice the application of this algorithm may significantly accelerate the execution of plans requiring approximation instead of a total reconstruction. Only a partial reconstruction of the plan is performed whose degree of computational difficulty is much smaller than

Algorithm 7.5. The partial reconstruction of a plan (*Reconstruction*)

Input:
- path $h = (h_1, ..., h_k)$ in the planning graph **PG** representing history of a given complex object, that is finished by an initial state to start of reconstruction,
- plan p_1 generated for a given complex object before the reconstruction,
- position *pos* of initial state of the reconstruction in the plan p_1,
- maximal depth d_r of the plan reconstruction,
- maximal length l_p of a repair plan during the reconstruction.

Output: The plan p_1 after reconstruction

```
1  Procedure Reconstruction(h, p_1, pos, d_r, l_p)
2  begin
3      j := pos
4      s := GetLastElementFrom(h)
5      while j ≤ pos + 2 · (d_r − 1) do
6          p_2 := EEFS(h, p_1[j], l_p)
7          if (p_2 is not empty) then
8              p_3 := Subpath(p_1, 1, pos−1)+p_2+Subpath(p_1, j+1, length(p_1))
9              return p_3
10         end
11         j := j + 2
12     end
13     return "empty plan"
14 end
```

the degree of difficulty of a total reconstruction (because of a smaller size of the problem of the partial reconstruction in relation to the size of the total reconstruction problem).

In practical applications there often occurs a situation that the reconstructed plan must have the same length as the original one. It happens that way when, *e.g.*, a plan proposed by the expert, which is to be executed over the established number of time units (*e.g.*, minutes, hours, days), must be reconstructed. A question arises, if in such a situation the duration time of partial reconstruction executed with the help of the algorithm $EEFS$ is in fact always shorter than the time of total reconstruction which is executed with the same algorithm $EEFS$? After all, the increase of the maximum reconstruction depth causes that in case of not finding the return state, the procedure $EEFS$ must be executed several times for the next reconstruction depths, that is, for $d_r = 1, 2, ..., n_p - 1$, where n_p is the length of reconstructed plan. If the reconstruction algorithm are used, assuming that the maximum reconstruction depth $d_r = n_p$ or even $d_r > n_p$, then obviously such a reconstruction would be more time costly for many plans than total reconstruction. Even in the case when $d_r < n_p$ it could seem that partial reconstruction executed using algorithm $EEFS$ may be for certain plans

more time costly than total reconstruction. However, this simple intuition is contradicted by the proposition presented below.

Proposition. *Let us assume that:*

- *p is a plan of n_p length which requires a reconstruction, where $n_p > 0$,*
- *AP is an automatic planning algorithm that its time cost is expressed using function $T(n) = C \cdot m^n$, where C is a constant, m is the maximum number of actions which are considered during the planning of an action in a given state and n is the maximum length of the plan under construction (the time complexity of AP is very similar to time complexity of algorithm $EEFS$).*

If a reconstruction reconstructs a plan of the same length as before this reconstruction, then for any maximum reconstruction depth $d_r < n_p$ the partial reconstruction executed with algorithm AP takes less time than total reconstruction.

Proof. The cost of total reconstruction of the plan p is $T(n_p)$, whereas the cost of partial reconstruction, with the maximum reconstruction depth d_r, is $T(1) + T(2) + ... + T(d_r)$. Therefore, partial reconstruction works faster than the total one when:

$$T(1) + T(2) + ... + T(d_r) < T(n_p)$$

that is:

$$C \cdot m^1 + C \cdot m^2 + ... + C \cdot m^{d_r} < C \cdot m^{n_p} \tag{7}$$

Hence:

$$m^{n_p+1} - m^{n_p} - m^{d_r+1} + m > 0 \tag{8}$$

It is sufficient to show that (8) is satisfied for any $m > 1$ and $0 < d_r < n_p$.

Starting from the left side of (8) we obtain:

$$m^{n_p+1} - m^{n_p} - m^{d_r+1} + m > m^{n_p+1} - m^{n_p} - m^{n_p-1+1} + m =$$

$$m^{n_p+1} - 2 \cdot m^{n_p} + m = m^{n_p}(m-2) + m \geq m^{n_p}(2-2) + m = m > 0$$

This completes the proof. □

On the basis of the above proposition one may state that partial reconstruction is always more effective than the total one, regardless of the plan length, maximum number of actions which may be performed in a given state and the maximum reconstruction depth. It must be stressed here, however, that partial reconstruction cannot always reconstruct a plan. In such a situation, the use of total reconstruction is the only option.

7.10 Automated Planning for Structured Complex Objects

In planning the behavior of structured objects, an effective planning of the behaviors of all objects which are parts of these objects at the same time is not possible. Therefore, in such cases the behavior of all objects which are parts of a structured object is planned separately. However, this approach to planning

of the behavior for a structured object requires a certain synchronization of the plans constructed for individual parts in such a way that these plans would not contradict each other and even complement each other in order to plan the best behavior for a structured object. For example, the treatment of illness A which is the resultant of two illnesses B and C requires such illnesses B and C treatment that the treatments of both illnesses would not be contradictory to each other, but even support and complement each other. For example, it may happen that in treating illness B a certain medicine M_1 may be used which is usually an appropriate medicine but it may be applied only when illness C does not occur. Hence, the synchronization of both illnesses' treatment should exclude the application of medicine M_1. In a different situation it may happen that as a result of application of medicine M_2 for illness C the treatment of illness B is safer, for instead of giving a certain strong medicine M_3, which has negative side effects, it is enough to give a safer medicine M_4 which leads to the same improvement in the patient's condition as in the case of giving medicine M_3.

In a few next subsections we present a generalization of the method for automated planning described in previous subsection for structured objects.

It is worth noticing that in literature one may observe the increase of interest in learning methods of common behaviors of structured objects. This issue is known under the term of learning communication protocols, cooperation and competition (see, *e.g.*, [351, 352]).

7.11 Planning Graphs for Structured Objects

In this paper, the elementary concept allowing planning the behavior of structured objects is the planning graph for structured objects.

Definition 65 (*A planning graph for structured objects*)**.** *A planning graph for structured objects of a fixed type T is a triple $\mathcal{PG} = (\mathcal{S}, \mathcal{A}, \mathcal{E})$ such that $(\mathcal{S}, \mathcal{A}, \mathcal{E})$ is the planning graph, where:*

- *elements of the set $\mathcal{S}$ are called meta states and they represent states of structured objects of the type T,*
- *elements of the set $\mathcal{A}$ are called meta actions and they represent actions for structured objects of the type T,*
- *elements of sets $PATH(\mathcal{PG}, k)$ (where $k > 1$) and $PATH(\mathcal{PG})$ are called meta paths,*
- *elements of sets $PLAN(\mathcal{PG}, k)$ (where $k > 1$) and $PLAN(\mathcal{PG})$ are called meta plans.*

In Fig. 46, we present an exemplary planning graph for a structured object, that is a group of four diseases: sepsis, Ureaplasma, RDS and PDA, related to the planning of the treatment of the infant during the respiratory failure (see Appendix B). This graph was created on the basis of observation of medical data sets (see Section 7.21) and with support of human experts.

As we see, there are two kinds of nodes in the planning graph for structured object, namely, *meta states nodes* (denoted by ovals) that represent the current

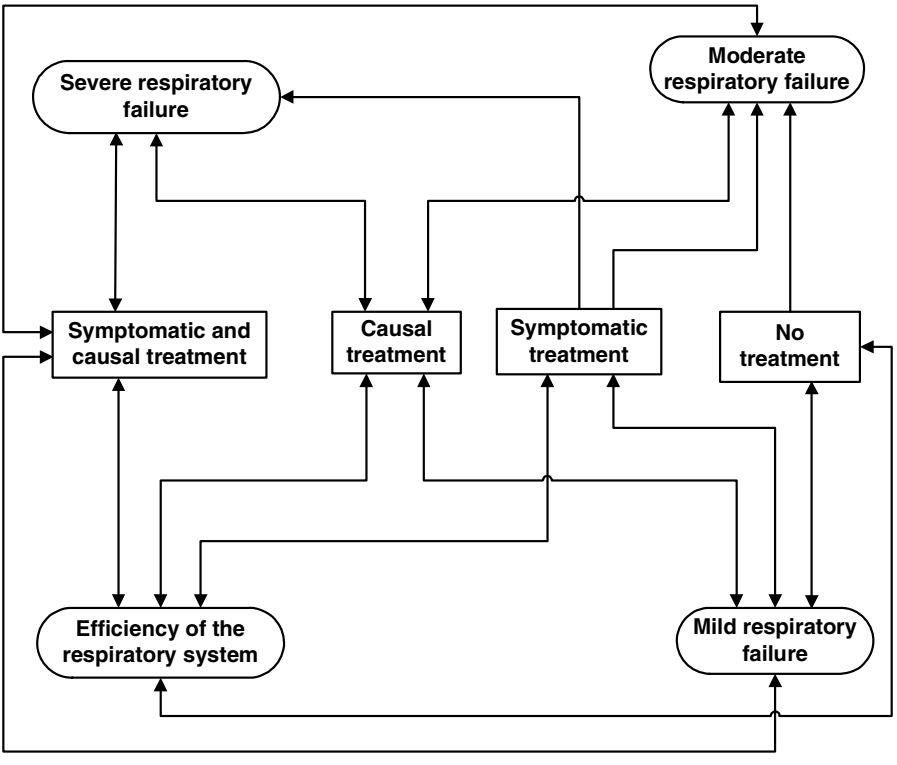


Fig. 46. A planning graph for the treatment of infants during the respiratory failure

state of a structured object specified as complex concepts by a human expert in natural language, and *meta action nodes* (denoted by rectangles) that represent actions defined for structured objects.

The major difference between the planning graph for the unstructured complex object and the planning graph for the structured object is that in the last one instead of actions performed at a single time point meta-actions occur which are performed over a longer period of time, that is, a time window.

Similarly to the case of unstructured complex objects the problem of planning for the structured object consists in constructing such a plan in planning graph $\mathcal{PG}$ that leads the structured object from the initial state to the expected target state.

7.12 The Identification of the Meta State

At the beginning of planning for a structured object, we identify the current meta state of this object. Any meta state node from a planning graph for structured objects can be treated as a complex spatio-temporal concept that is specified by a human expert in natural language. Such concepts can be approximated

by classifiers using data sets and domain knowledge accumulated for a given complex dynamical system. Similarly to states from the planning graph for unstructured complex objects, any state from the planning graph for structured objects can be approximated as a temporal concept for unstructured object (see Section 7.5). However, the state from the planning graph for structured objects can be also treated as the temporal concept for structured objects. Therefore, in this case the method of approximation from Section 6.22 can be used instead of the method from Section 6.9. As a result, it is possible to recognize the initial state at the beginning of planning for a particular structured object.

7.13 Planning of a Meta Action

Similarly to the single complex object, during planning for some structured object the path in the planning graph from the initial node-state to the target node-state should be found. At the beginning of planning for a structured object, we identify the current state of this object. As mentioned earlier, this can be done by classifiers that have been constructed for all states from the planning graph. Next, we plan a sequence of meta actions for transforming a structured object from the current meta state to the target meta state (more expected, safer or more comfortable). For example, in the case of the treatment of infants with respiratory failure, if the infant is suffering from severe respiratory failure, we try to change the patient status using some methods of treatment to change its status to moderate or mild respiratory failure (see Fig. 46). However, any meta action from such constructed path should be checked on the lower level, i.e., on the level of any part of the structured object separately, if such action can be realized in practice in case of particular part of this structured object. In other words, it means that for any part of the structured object the sequence of action should be planed in order to obtain meta-action on the level of the structured object.

The plan of execution of a single meta-action, which consists of short plans which execute this meta-action on the levels of individual parts of the structured object, is called *a g-plan*.

Definition 66 (*A g-plan*)**.** *Let us assume that:*

- $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t}, a_c)$ *is a temporal information system with actions,*
- T *is a type of structured objects, where objects of this type are composed of* l *parts of object of types* T_1,...,T_l,
- $\mathbf{PGF} = \{\mathbf{PG}_1, ..., \mathbf{PG}_l\}$ *is a family of planning graphs, where* $\mathbf{PG}_i = (S_i, A_i, E_i)$ *is a planning graph for complex objects of type* T_i*, for* $i = 1, ..., l$,
- k *is a fixed plan length in planning graphs from the family* $\mathbf{PGF}$.

1. *A g-plan with length* k *for the family of planing graphs* $\mathbf{PGF}$ *is a family of plans* $\{p_1, ..., p_l\}$ *(assigned to be executed for all parts of the established structured object) such that* $p_i \in PLAN(\mathbf{PG}_i, k)$ *for* $i = 1, ...l$. *The set of all g-plans with length* k *for the family of planing graphs* $\mathbf{PGF}$ *is denoted by* $GPLAN(\mathbf{PGF}, k)$.

2. *Any g-plan* $\{p_1, ..., p_l\} \in GPLAN(\mathbf{PGF}, k)$ *is called a g-plan occurring in system* $\mathbf{T}$*, if* $p_i \in DPLAN(\mathbf{T}, \mathbf{PG}_i, k)$*, for* $i = 1, ..., l$ *and in system* $\mathbf{T}$ *there exists such a family of time windows* $\{W_1, ..., W_l\} \subseteq TW(\mathbf{T}, k)$ *that the following conditions are satisfied:*
   - $\forall_{W_i \in \{W_1, ..., W_l\}} W_i \in TW(\mathbf{T}, p_i)$,
   - $\forall_{j \in \{1, ..., k\}}\ a_t(W_1[j]) = a_t(W_2[j]) = ... = a_t(W_l[j])$.
3. *A set of all g-plans occurring in the system* $\mathbf{T}$ *with length* k *and constructed for the family of planing graphs* $\mathbf{PGF}$*, is denoted by* $DGPLAN(\mathbf{T}, \mathbf{PGF}, k)$.

The g-plan is, thus, a family of plans assigned to be executed for all parts of the established structured object. The g-plan occurs in the temporal information system with actions if its performance is observed in the data.

Let us notice that determining the $DGPLAN(\mathbf{T}, \mathbf{PGF}, k)$ requires not only determining sets $DPLAN$ for all parts of the structured object but also synchronizing them in time. There arises a problem of isolating structured objects. If we assume, however, that the structured objects are created with the help of the sweeping algorithm around parts of structured objects (see Section 6.13), then the problem of determining the set $DGPLAN$ is significantly simpler.

In practise, all constructed plans for objects (parts) belonging to a given structured object should be compatible. Therefore, during planning a meta action for a structured object, we use a special tool for verifying the compatibility of plans generated for all members of a structured object. This verification can be performed by using some special decision rules that we call *elimination rules*. Such rules make it possible to eliminate combination of plans that are not compatible relative to domain knowledge. This is possible because elimination rules describe all important dependencies between plans that are joined together. If any combination of plans is not consistent with any elimination rule, then it is eliminated. A set of elimination rules can be specified by human experts or can be computed from data sets. In both of these cases, we need a set of attributes (features) defined for a single plan that are used for explaining elimination rules. Such attributes are specified by human experts on the basis of domain knowledge and they describe some important features of the plan (generated for some part of structured object) with respect to proper joining a plan with plans generated for other parts of structured object.

These features are used as a set of attributes in the special table that we call *an elimination table*. Any row of an elimination table represents information about features of plans assigned for structured objects from the training data.

Definition 67 (*An elimination table*)**.** *Let us assume that:*

- $\mathbf{T}$ *is a temporal information system with actions,*
- T *is a type of structured objects, where objects of this type are composed of* l *parts of object of types* T_1,...,T_l,
- $\mathbf{PGF} = \{\mathbf{PG}_1, ..., \mathbf{PG}_l\}$ *is a family of planning graphs, where* $\mathbf{PG}_i = (S_i, A_i, E_i)$ *is a planning graph for complex objects of type* T_i*, for* $i = 1, ..., l$,
- k *is a fixed plan length in planning graphs from the family* $\mathbf{PGF}$,

- $\Phi_i = \{\phi_i^1, ..., \phi_i^{m_i}\} \subseteq FPPG(\mathbf{PG}_i)$ *is a family of formulas defined by experts, for* $i = 1, ..., l$,
- $\Phi = \Phi_1 \cup \ ... \ \cup \ \Phi_l$,
- $\mathcal{P}_{GP} = (U, \Phi, \models_{GP})$ *is a property system, where*

$$U = DGPLAN(\mathbf{T}, \mathbf{PGF}, k) \text{ and}$$

the satisfiability relation $\models_{GP} \subseteq U \times \Phi$ *is defined in the following way:*

$$\forall gp = \{p_1, ..., p_l\} \in U \text{ and } \phi \in \Phi : \ gp \models_{GP} \phi \ \Leftrightarrow$$

$$p_i \models_{FPPG(\mathbf{PG}_i)} \phi, \quad \text{for } i \in \{1, .., l\} \text{ such that } \phi \in \Phi_i.$$

The information system defined by the property system $\mathcal{P}_{GP}$ *is called an elimination table of g-plans from the set* $GPLAN(\mathbf{PGF}, k)$.

It is easy to notice that if the set $DGPLAN(\mathbf{T}, \mathbf{PGF}, k)$ has already been determined and the examination of formula satisfiability may be executed over constant time, then the algorithm which determines the elimination table works over time of order $O(n \cdot m)$, where:

$$n = card(DGPLAN(\mathbf{T}, \mathbf{PGF}, k)) \text{ and } m = card(\Phi_1 \cup ... \cup \Phi_l).$$

Example 37. The respiratory failure may be treated as a result of four following diseases: RDS, PDA, sepsis and Ureaplasma. Therefore, treating respiratory failure requires simultaneous treatment of all of these diseases. This means that the treatment plan of respiratory failure comes into existence by joining the treatment plans for diseases RDS, PDA, sepsis and Ureaplasma, and at the same time the synchronization of the plans is very important. In this paper, one of the synchronizing tools for this type of plans is the elimination table. In constructing the elimination table for treatment of respiratory failure, patterns describing the properties of the joint plans are needed. Moreover, planning graphs for all four diseases are necessary. In Fig. 44 the planning graph for RDS treatment is shown, whereas in Example 36 we showed how the features of RDS treatment plans may be defined. In a very similar way the features of treatment plans for PDA, sepsis and Ureaplasma diseases may be defined. However, in this paper we do not present the planning graphs for treating these diseases. The reason for this is a high degree of complexity of these graphs in terms of medical knowledge (particularly in the case of treating disease sepsis). Therefore, we also cannot give examples of specific features which may be used to describe the treatment plans for diseases: PDA, sepsis and Ureaplasma (as we did in the case of RDS treatment) (see Example 36). □

On the basis of the elimination table a set of elimination rules can be computed that can be used to eliminate inappropriate plan arrangements for individual parts of the structured object.

Definition 68 (*An elimination rule*). *Let us assume that:*

- $\mathbf{T}$ *is a temporal information system with actions,*
- T *is a type of structured objects, where objects of this type are composed of* l *parts of object of types* T_1,...,T_l,
- $\mathbf{PGF} = \{\mathbf{PG}_1, ..., \mathbf{PG}_l\}$ *is a family of planning graphs, where* $\mathbf{PG}_i = (S_i, A_i, E_i)$ *is a planning graph for complex objects of type* T_i, *for* $i = 1, ..., l$,
- k *is a fixed plan length in planning graphs from the family* $\mathbf{PGF}$,
- $\mathbf{ET}(\mathbf{T}, \mathbf{PGF}, k) = (U, A)$ *is an elimination table.*

1. *If* $a \in A$ *then any decision rule with minimal number of descriptors (see Section 2.3 and Section 2.4) computed for a decision table* $(U, A \setminus \{a\}, a)$ *is called an elimination rule for the elimination table* $\mathbf{ET}(\mathbf{T}, \mathbf{PGF}, k)$.
2. *A set of all elimination rules for the elimination table* $\mathbf{ET}(\mathbf{T}, \mathbf{PGF}, k)$ *is denoted by* $\mathbf{ERUL}(\mathbf{T}, \mathbf{PGF}, k)$.
3. *If* $r \in \mathbf{ERUL}(\mathbf{T}, \mathbf{PGF}, k)$, *then an object* $u \in U$ *is eliminated by the elimination rule* r *iff* u *matches the predecessor of* r *and does not match the successor of the rule* r.

Algorithm 7.6. Generation of elimination rules

Input:
- temporal information system with actions $\mathbf{T}$,
- type T of structured objects, where objects of this type are composed of l parts of object of types T_1,...,T_l,
- family of planning graphs $\mathbf{PGF} = \{\mathbf{PG}_1, ..., \mathbf{PG}_l\}$, where $\mathbf{PG}_i = (S_i, A_i, E_i)$ is a planning graph for complex objects of type T_i, for $i = 1, ..., l$,
- fixed plan length k from planning graphs from the family $\mathbf{PGF}$,
- elimination table $\mathbf{ET}(\mathbf{T}, \mathbf{PGF}, k) = (U, A)$ such that $A = \{a_1, ..., a_m\}$,
- minimal support t_s of useful elimination rules.

Output: The set of elimination rules computed for the table $\mathbf{ET}(\mathbf{T}, \mathbf{PGF}, k)$

```
1  begin
2      Create empty set of rules ERUL
3      for any a ∈ A do
4          Create a decision table ET_a = (U_a, A_a, d) such that, U_a = U,
           A_a = A \ {a} and d = a
5          Generate a set RUL(ET_a) of decision rules with minimal number
           of descriptors (see Section 2.3 and Section 2.4) for the table ET_a
6          Add rules from the set RUL(ET_a) to the set ERUL
7      end
8      Remove from the set ERUL all rules with support less than t_s
9      return ERUL
10 end
```

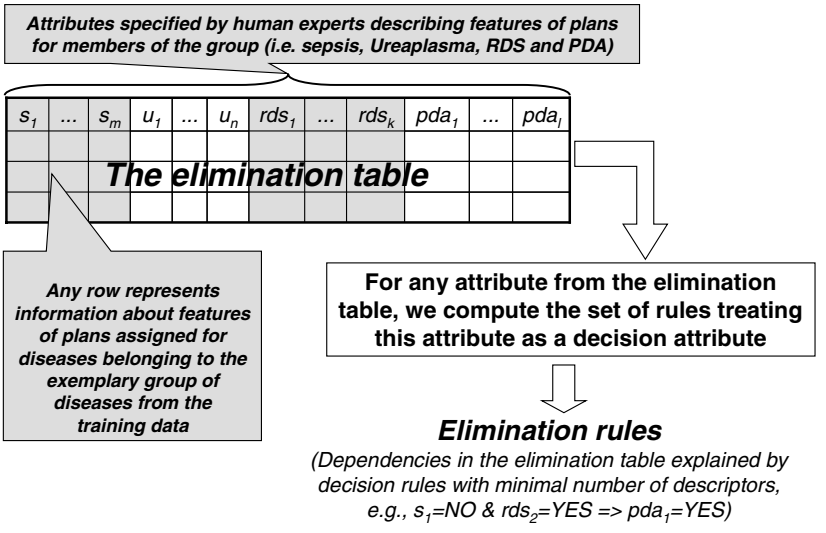


Fig. 47. The scheme of construction of elimination rules for group of four diseases: sepsis, Ureaplasma, RDS and PDA

So, the set of elimination rules can be used as a filter of inconsistent combinations of plans generated for members of groups. Any combination of plans is eliminated when there exists an elimination rule that is not supported by features of a combination while the combination matches a predecessor of this rule. In other words, a combination of plans is eliminated when the combination matches to the predecessor of some elimination rule and does not match the successor of a rule.

We propose the following method of calculation the set of elimination rules on the basis of the elimination table (see Algorithm 7.6).

As we see in the Algorithm 7.6, for any attribute from the elimination table, we compute the set of rules with minimal number o descriptors (see Section 2.3 and Section 2.4) treating this attribute as a decision attribute. In this way, we obtain a set of dependencies in the elimination table explained by decision rules. In practice, it is necessary to filter elimination rules to remove the rules with low support because such rules can be too strongly matched to the training data.

Fig. 47 shows the scheme of elimination rules of not-acceptable g-plans constructed in the case of the treatment of respiratory failure, which is a result of the four following diseases: sepsis, Ureaplasma, RDS and PDA.

On the basis of the set of elimination rules *an elimination classifier* may be constructed that enable elimination of inappropriate plan arrangements for individual parts of the structured object.

Definition 69 (*An elimination classifier*)**.** *Let us assume that:*

- **T** *is a temporal information system with actions,*
- *T is a type of structured objects, where objects of this type are composed of l parts of object of types* T_1,...,T_l,

- $\mathbf{PGF} = \{\mathbf{PG}_1, ..., \mathbf{PG}_l\}$ *is a family of planning graphs, where* $\mathbf{PG}_i = (S_i, A_i, E_i)$ *is a planning graph for complex objects of type* T_i*, for* $i = 1, ..., l$,
- k *is a fixed plan length in planning graphs from the family* $\mathbf{PGF}$,
- $\mathbf{ET}(\mathbf{T}, \mathbf{PGF}, k) = (U, A)$ *is an elimination table,*
- $\mathbf{ERUL}(\mathbf{T}, \mathbf{PGF}, k)$ *is a set of all elimination rules of g-plans from the set* $GPLAN(\mathbf{PGF}, k)$.

An elimination classifier based on the set $\mathbf{ERUL}(\mathbf{T}, \mathbf{PGF}, k)$ *of all elimination rules (or on some subset of this set) is a classifier denoted in general by* $\mu_{\mathbf{ET}(\mathbf{T},\mathbf{PGF},k)}$ *and classifying g-plans in the following way:*

$$\forall u \in U : \mu_{\mathbf{ET}(\mathbf{T},\mathbf{PGF},k)}(u) = \begin{cases} false & when \ \exists_{r \in ERUL} \ u \ is \ eliminated \ by \ r \\ true & otherwise. \end{cases}.$$

If the combination of plans for parts of the structured object is consistent (it was not eliminated by elimination rules), we should check if the execution of this combination allows us to realize the expected meta action from the level of structured objects. This can be done by a special classifier constructed for a table called a *meta action table*. The structure of a meta action table is similar to the structure of an elimination table, i.e., attributes are defined by human experts, where rows represent information about features of plans assigned for parts of exemplary structured objects from the training data. In addition, we add to this table a decision attribute. Values of such decision attributes represent names of meta actions which are realized as an effect of the execution of plans described in the current row of a training table.

Definition 70 (*A meta action table*)**.** *Let us assume that:*

- $\mathbf{T}$ *is a temporal information system with actions,*
- T *is a type of structured objects, where objects of this type are composed of* l *parts of object of types* T_1,...,T_l,
- $\mathcal{PG} = (\mathcal{S}, \mathcal{A}, \mathcal{E})$ *is a planning graph for structured objects of the type* T,
- $\mathbf{PGF} = \{\mathbf{PG}_1, ..., \mathbf{PG}_l\}$ *is a family of planning graphs, where* $\mathbf{PG}_i = (S_i, A_i, E_i)$ *is a planning graph for complex objects of type* T_i*, for* $i = 1, ..., l$,
- k *is a fixed plan length in planning graphs from the family* $\mathbf{PGF}$,
- $\Phi_i = \{\phi_i^1, ..., \phi_i^{m_i}\} \subseteq FPPG(\mathbf{PG}_i)$ *is a defined by experts family of formulas, for* $i = 1, ..., l$,
- $\Phi = \Phi_1 \cup \ ... \ \cup \ \Phi_l$,
- $\mathcal{P}_{GP} = (U, \Phi, \models_{GP})$ *is a property system, where*

$$U = DGPLAN(\mathbf{T}, \mathbf{PGF}, k) \text{ and}$$

the satisfiability relation $\models_{GP} \subseteq U \times \Phi$ *is defined in the following way:*

$$\forall gp = \{p_1, ..., p_l\} \in U \text{ and } \phi \in \Phi : \ gp \models_{GP} \phi \ \Leftrightarrow$$

$$p_i \models_{FPPG(\mathbf{PG}_i)} \phi, \quad for \ i \in \{1, .., l\} \ such \ that \ \phi \in \Phi_i.$$

1. *A meta action table of g-plans for structured objects is a decision table* $\mathbf{MAT}(\mathbf{T}, \mathbf{PGF}, k) = (U, A, d)$*, where:*
   - (U, A) *is an information system defined be the property system* $\mathcal{P}_{GP}$*,*
   - d *is a decision attribute that for any g-plan from the set* U*, represents a meta action corresponding to execution of this g-plan.*
2. *If* $\mathbf{MAT}(\mathbf{T}, \mathbf{PGF}, k)$ *is the meta action table, then any classifier computed for the table* $\mathbf{MAT}(\mathbf{T}, \mathbf{PGF}, k)$ *is called a meta action classifier and is denoted by* $\mu_{\mathbf{MAT}(\mathbf{T},\mathbf{PGF},k)}$*.*

The classifier computed for an action table makes it possible to predict the name of a meta action for a given combination of plans from the level of parts of a structured object. The last step is the selection of combinations of plans that makes it possible to obtain a target meta action with respect to a structured object (see Fig. 48).

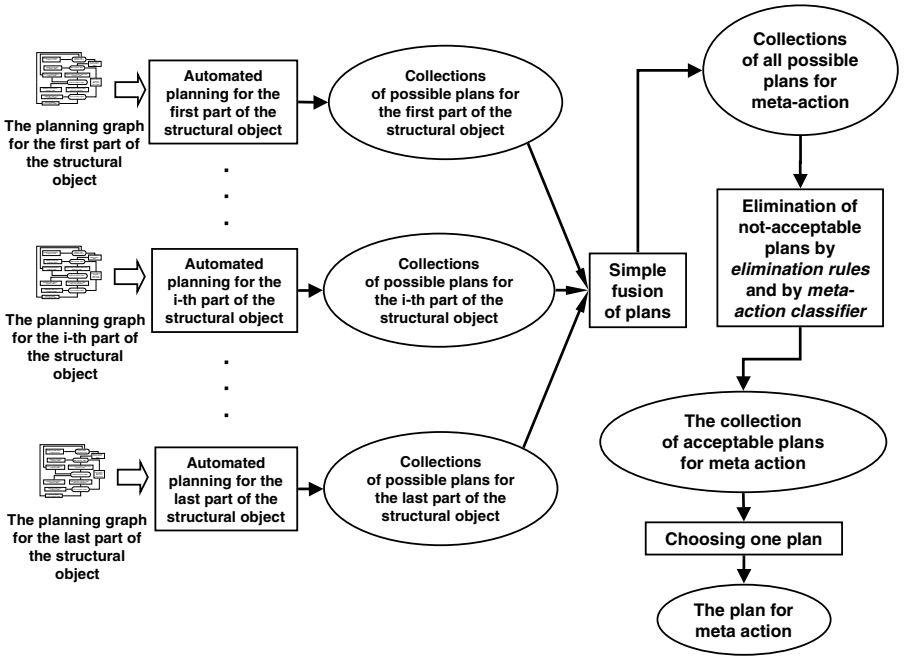


Fig. 48. The scheme of meta action planning

Example 38. The treatment of respiratory failure requires simultaneous treatment of RDS, PDA, sepsis and Ureaplasma. Therefore, the treatment plan for respiratory failure comes to existence by joining the treatment plans for RDS, PDA, sepsis and Ureaplasma, and at the same time the synchronization of those plans is very important. The first tool to synchronize these types of plans is the elimination classifier generated for the elimination table. The second tool, however, is the meta action classifier generated for the meta action table. Similarly to the case of the elimination table, also in constructing the meta action table,

patterns describing the properties of the joint treatment plans for RDS, PDA, sepsis and Ureaplasma are needed. These patterns are very similar as in the case of the patterns used to construct the elimination table (see Example 37). □

It was mentioned in Section 7.3 that the resolving classifier used for generation of a next action during the planning for a single object, gives us the list of actions (and states after usage of action) with their weights in descending order. This makes it possible to generate many alternative plans for any single object and many alternative combinations of plans for a structured object. Therefore, the chance of finding an expected combination of plans from a lower level to realize a given meta action (from the higher level) is relatively high.

After planning the selected meta action from the path of actions from the planning graph (for a structured object), the system begins the planning of the next meta action from this path. The planning is stopped, when the planning of the last meta action from this path is finished.

7.14 Data Structures, Algorithms and Numerical Constants Concerning the Planning for Structured Objects

In the next subsections, we present several algorithms which are needed to plan the behavior of structured objects. Therefore, in this subsection we formulate the problem of behavior planning of such objects and we mention elementary data structures, algorithms, and numerical constants which we use in the described algorithms. This allows avoiding repetitive descriptions of elements of such a kind.

When we speak about the structured object's behavior planning, we always mean the behavior planning of a certain structured object O of the established type T which consists of parts $O_1, ..., O_l$ which are respectively the objects of types $T_1, ..., T_l$. In practical applications, objects $O_1, ..., O_l$ are often unstructured objects, however, they also may be structured objects of lesser complexity than object O. Hence, the methodology of structured object behavior planning described here is of a hierarchical character.

In this paper, in automatic planning of complex object behavior we use data sets represented by the temporal information system with actions $\mathbf{T} = (U, A, a_{id}, \leq_{a_{id}}, a_t, \leq_{a_t}, a_c)$. We also use the following data structures, algorithms, and numerical constants:

- a planning graph $\mathcal{PG} = (\mathcal{S}, \mathcal{A}, \mathcal{E})$ for structured objects of a fixed type T,
- a fixed plan length K in planning graphs $\mathcal{PG}$,
- a classifier $\mu_{\mathbf{RC}(\mathcal{PG},S,K)}$, for all $S \in \mathcal{S}$,
- a classifier of dissimilarity $\mu_{DIT(\mathcal{PG})}$ between meta states from the planning graph $\mathcal{PG}$,
- a limitation *ActionLimit* of the number of meta actions which may be performed in a given state $S \in \mathcal{S}$,
- a maximal length L_p of generated plan for structured objects,
- a maximal depth D_r of a plan reconstruction for structured objects,
- a maximal length L_{rp} of a repair plan for structured objects (during reconstruction),

- a family of planning graphs $\mathbf{PGF} = \{\mathbf{PG}_1, ..., \mathbf{PG}_l\}$, where $\mathbf{PG}_i = (S_i, A_i, E_i)$ is a planning graph for complex objects of type T_i, for $i = 1, ..., l$,
- a fixed length o plans k in planning graphs from the family $\mathbf{PGF}$,
- a classifier of dissimilarity $\mu_{DIT(\mathbf{PG}_i)}$ between states from the planning graph $\mathbf{PG}_i$, for $i = 1, ..., l$,
- a length l_{gp} of generated g-plan constructed for execution of any meta action from the set $\mathcal{A}$,
- an elimination classifier $\mu_{\mathbf{ET}(\mathbf{T},\mathbf{PGF},k)}$ (see Definition 69),
- a meta action classifier $\mu_{\mathbf{MAT}(\mathbf{T},\mathbf{PGF},k)}$ (see Definition 70),
- a maximal depth d of reconstruction of g-plans,
- a maximal length l_{rp} of a repair of g-plan.

The mentioned above data structures, algorithms, and numerical constants are used in algorithms presented in further subsections.

7.15 Algorithms of the Meta Action Planning

The basic method of automated planning for the structured object is the method of planning of the meta-action. The planning of the established meta-action requires constructing such a g-plan that has a required length, it is compatible with domain knowledge and its execution corresponds to the performance of the established meta-action.

We present an example of an automated planning algorithm for a meta-action which has been used in our experiments (see Algorithm 7.7).

The procedure *SearchGPlanMA* (see Algorithm 7.7) constructs a g-plan for the structured object which executes the required meta-action. First, the sets of plans are generated for all the parts of the structured object separately. Then, the Cartesian product $PCList$ of these sets is created, whose elements are candidates for the g-plan that is being searched for. Finally, this product is overviewed until finding such a g-plan that is not eliminated by classifier $\mu_{\mathbf{ET}(\mathbf{T},\mathbf{PGF},k)}$ and performs the required meta-action on the structured object according to classifier $\mu_{\mathbf{MAT}(\mathbf{T},\mathbf{PGF},k)}$. If during the overview of the set $PCList$ such a g-plan occurs, then the execution of the algorithm is ended and exactly this g-plan is returned as a solution. If during the overview of the $PCList$ set the algorithm does not encounter such a g-plan, then an empty g-plan is returned which means that the algorithm has not found the solution. However, before the overview of the set $PCList$ takes place, it is sorted. The sorting is necessary for the first g-plan that is found which executes the required meta-action is the most recommended in terms of all parts of the structured object for which the plans are constructed. An example of such sorting g-plans may be the sorting in relation to the weight $GPlanWeight$ which is defined for a given g-plan $gp = (p_1, ..., p_l)$, where $p_i \in PList_i$ for $i = 1, ..., l$, in the following way:

$$GPlanWeight(gp) = \frac{1}{\sum_{i=1}^{l} \text{ Index of } p_i \text{ in the list } PList_i}.$$

The time complexity of the Algorithm 7.7 depends greatly on the size of the set $PCList$. Therefore, the length of the plan lists generated for individual parts

Algorithm 7.7. Searching for g-plan for a meta action ($SearchGPlanMA$)

Input:
- path $h_i \in PLAN(\mathbf{PG}, k)$ in the planning graph $\mathbf{PG}_i$ representing history of the object O_i, that is finished by an initial state to start of automated planning for object O_i, for $i = 1, ..., l$,
- target meta action $ma_t \in \mathcal{A}$,
- length of g-plan l_{gp}.

Output: The g-plan for execution the meta action ma_t

```
1  Procedure SearchGPlanMA({h_1, ..., h_l}, ma_t, l_gp)
2  begin
3      PList_1 := FEEFS(h_1, l_gp)
4      ...
5      PList_l := FEEFS(h_l, l_gp)
6      PCList := PList_1 × ... × PList_l
7      Sort PlanCompList by established method
8      for i := 1 to length(PCList) do
9          gplan := PCList[i]
10         if (μ_ET(T,PGF,k)(gplan) = true) then
11             if (μ_MAT(T,PGF,k)(gplan) = ma_t) then
12                 return gplan
13             end
14         end
15     end
16     return "empty g-plan"
17 end
```

of the structured object should be chosen in such a way that the size of the set $PCList$ would be feasible for searching over this set.

Sometimes, a slightly different algorithm of g-plans searching is necessary. We mean the situation when a g-plan is constructed, that satisfies certain conditions for all parts of the analyzed structured object. In such a situation the Algorithm 7.8 may be used.

As we see, the Algorithm 7.8 constructs such a g-plan for the structured object that ends with specific states for individual parts of the structured object.

7.16 Automated Planning Algorithm for Structured Objects

Now, we may present the planning algorithm for the structured object which uses the *SearchGPlanMA* procedure (see Algorithm 7.9).

The algorithm 7.9 takes into consideration different variants for a meta-action, that may be performed in a given meta-state. However, only those actions are taken into account which algorithm $SearchGPlanMA$ can execute using a g-plan on the level of single parts of the structured object. Similarly to the case of

Algorithm 7.8. Searching for g-plan for states (*SearchGPlan*)

Input:
- path $h_i \in PLAN(\mathbf{PG}, k)$ in the planning graph $\mathbf{PG}_i$ representing history of the object O_i, that is finished by an initial state to start of automated planning for object O_i, for $i = 1, ..., l$,
- target state s_i for the object O_i, for $i = 1, ..., l$,
- length of g-plan l_{gp}.

Output: The g-plan ends by states $s_1, ..., s_l$

```
1  Procedure SearchGPlan({h_1, ..., h_l}, {s_1, ..., s_l}, l_gp)
2  begin
3      PList_1 := FEEFS(h_1, l_gp)
4      ...
5      PList_l := FEEFS(h_l, l_gp)
6      PCList := PList_1 × ... × PList_l
7      Sort PCList by fixed order
8      for i := 1 to length(PCList) do
9          {p_1, ..., p_l} := PCList[i]
10         if (μ_ET(T,PGF,k)(gplan) = true) then
11             if (p_1[l_p] = s_1) and .... and (p_l[l_p] = s_l) then
12                 return {p_1, ..., p_l}
13             end
14         end
15     end
16     return "empty g-plan"
17 end
```

Algorithm 7.2, for the regulation of computing time duration, limitation *ActionLimit* is used, that is, limitation of the number of actions which may be performed in a given state. Besides that, classifier $\mu_{\mathbf{RC}(\mathcal{PG},S,K)}$ returns the list of pairs (meta action + meta state) sorted decreasingly in relation to the weights obtained from classification. Hence, the meta-actions are taken into account in the order from the ones most recommended by the classifier $\mu_{\mathbf{RC}(\mathcal{PG},S,K)}$ to the less recommended by this classifier. This way, the algorithm constructs a certain plan tree whose root is the initial meta-state and the leaves are the meta-states after performing the individual variants of the plan. If during construction of this tree the final meta-state occurs, then the performance of the algorithm is ended and as a solution a sequence of meta-states and meta-actions is returned which starts in the tree root and ends in the final meta-state that has been found. If during construction of plan tree the algorithm does not encounter the final meta-state, an empty plan is returned which means that the algorithm has not found the solution.

The analysis of the pessimistic time complexity of Algorithm 7.9 is very similar to the Algorithm 7.2. The only difference is that before applying the meta action

Algorithm 7.9. Exhaustive expert forward search for a structured object

Input:
- path $H = (H_1, ..., H_k) \in PLAN(\mathcal{PG}, k)$ representing history of a given structured object O, that is finished by an initial meta state to start of automated planning,
- family of paths $h_1, ..., h_l$, where h_i is a path from the planning graph $\mathbf{PG}_i$, representing history of the object O_i during execution the last meta action for this object, for $i = 1, ..., l$,
- target meta state $S_t \in \mathcal{S}$,
- a maximal length L_p of plan for structured objects.

Output: The plan P for structured object O ended by the meta state S_t

```
1  Procedure EEFSS(H, {h_1, ..., h_l}, S_t, L_p)
2  begin
3      S := GetLastElementFrom(H)
4      P := "empty plan"
5      P := P + S;
6      L := PairList(μ_RC(PG,S,K)(H))
7      for i = 1 to ActionLimit do
8          P_1 := Copy(P);
9          gplan := SearchGPlanMA({h_1, ..., h_l}, L[i].action)
10         if (gplan in not empty) then
11             P_1 := P_1 + L[i].action + L[i].state
12             if (L[i].state = S_t) then
13                 return P_1
14             end
15             if (L_p > 1) then
16                 H_1 := Copy(H)
17                 RemoveFirstTwoElementsFrom(H_1)
18                 H_1 := H_1 + L[i].action + L[i].state
19                 P_2 := EEFSS(H_1, gplan, S_t, L_p − 1)
20                 if (P_2 is not empty ) then
21                     return P + P_2
22                 end
23             end
24         end
25     end
26     return "empty meta plan"
27 end
```

the procedure $SearchGPlanMA$ should be executed in order to check how a given meta-action can be executed for the parts of the structured object. If it is even assumed that this checking takes place over constant time, then the pessimistic time complexity of the procedure $EEFSS$ is of order $O(m^n)$ where

n is the length of the constructed plan and m is the limitation of the number of meta-actions which may be performed in a given meta-state (*ActionLimit*). This means that similarly to the case of algorithm $EEFS$ the effective application of the algorithm $EEFSS$ for nontrivially small n and m is practically impossible. However, in practice the planning graphs for the structured object are relatively simple (see Fig. 46) and this algorithm may be used for them.

7.17 Reconstruction of Plan for Structured Objects

The main aim of this section is to present the algorithm of reconstruction of a plan constructed for structured objects. Similarly to the case of the unstructured complex object (see Section 7.9) such a reconstruction may be performed during the execution of the meta plan for the structured object when the initially established plan cannot be continued.

We may present the algorithm simulating the execution of the meta plan which foresees the reconstruction of the plan during its execution (see Algorithm 7.10).

The Algorithm 7.10 simulates the execution of the meta-plan found earlier for the structured object. The simulation is performed based on the procedure $SimulateMA$ which on the input takes the history of the current states of all parts of the structured object. The procedure $SimulateMA$ returns the g-plan gp which has been performed as a result of the simulation of the meta-action. This g-plan may differ from the input g-plan, because during the simulation of the input g-plan a reconstruction on the level of meta-action performance may have occurred (see Algorithm 7.11). Therefore, it may happen that the simulation of the meta-action leads to a different meta-state than the one expected in the original plan. Hence, the procedure *MReconstruction* can be executed in the further process a reconstruction for the structured object (see Algorithm 7.13).

Now, we present the simulation algorithm of meta-action for the structured object with the consideration of reconstruction (see Algorithm 7.11 and Fig. 49).

The simulation is performed based on the procedure *GSimulate* which on the input takes the history of the current states of all parts of the structured object and actions which are to be performed for all these parts. Although it is possible to imagine this type of procedure as a part of the behavior simulator of the complex object (*e.g.*, the traffic simulator, the illness development simulator), in this paper by this procedure we understand the changes in the real system of complex objects which may be triggered by performing specific meta-actions for the structured object.

The Algorithm 7.11 uses the procedure of reconstruction *GReconstruction*. Therefore, we present this procedure as the Algorithm 7.12.

The Algorithm 7.12 tries to find a short repair g-plan not longer than l_{rp} which brings the initial states of reconstruction (the last states in the histories $h_1, ..., h_l$) to the states appearing synchronically in the plans $p_1, ..., p_l$ starting from position *pos* as far as position $pos + 2 \cdot (d_r - 1)$. The maximum depth of reconstruction d_r is then the number of states in the plans $p_1, ..., p_l$ (starting from the states in position *pos*) which the algorithm tries to reach using the repair g-plan. The repair g-plan is searched for by algorithm *SearchGPlan*.

Algorithm 7.10. The simulation of plan execution for a structured object

Input:
- a path $H \in PLAN(\mathcal{PG}, k)$ representing history of a given structured object O, that is finished by an initial meta state to start of a simulation,
- a family of paths $h_1, ..., h_l$, where h_i is a path from the planning graph $\mathbf{PG}_i$, representing history of the object O_i during execution the last meta action for this object (before the simulation), for $i = 1, ..., l$,
- a plan $P \in PLAN(\mathcal{PG}, L_P)$ established for a given complex object O,
- a sequence of g-plans $gp_1, ..., gp_{\frac{L_P-1}{2}}$ implementing meta actions from P.

Output: The plan P executed for the structured object O

```
1  begin
2      if (length(P) < 3) return "meta plan P is too short for execution"
3      H_s := Copy(H)
4      H_p := Copy(H)
5      i := 1
6      while (i < length(P)) do
7          gp := SimulateMA({h_1, ..., h_l}, gp_{(i+1)/2})
8          if(gp is empty) return "start total reconstruction"
9          S := μ_{MAT(T,PGF,k)}(gp)
10         RemoveFirstTwoElementsFrom(H_s)
11         H_s := H_s + P[i+1] + S
12         RemoveFirstTwoElementsFrom(H_p)
13         H_p := H_p + P[i+1] + P[i+2]
14         if (S ≠ P[i+2]) then
15             dism := μ_{DIT(PG)}(H_s, H_p)
16             if (dism is "high") then return "start total reconstruction"
17             if (dism is not "low") then
18                 P' := MReconstruction(H_s, {h_1, ..., h_l}, P, i+2)
19                 if (P' is empty) then return "start total reconstruction"
20                 P := P'
21             end
22         end
23         i := i + 2 // Go to the next meta action from the plan P
24     end
25 end
```

Computational complexity of Algorithm 7.12 depends linearly on the complexity of the algorithm *SearchGPlan*. However, in practice using this algorithm may significantly accelerate the execution of plans which require reconstruction because instead of a total reconstruction, only a partial reconstruction is performed.

If the meta state S, achieved as a result of simulation, differs too much from its counterpart in plan P, then the total reconstruction of plan P is performed.

Algorithm 7.11. The simulation of meta-action for the structured object with the consideration of g-plan reconstruction

Input:
- family of paths $h_1, ..., h_l$, where h_i is a path from the planning graph $\mathbf{PG}_i$, representing history of the object O_i during execution the last meta action for this object, for $i = 1, ..., l$,
- meta action $ma \in \mathcal{A}$ and g-plan $gp = (p_1, ..., p_l)$ corresponding to meta action ma, that should be performed during a simulation.

Output: The g-plan executed for the structured object O

```
1  Procedure SimulateMA({h_1, ..., h_l}, {p_1, ..., p_l})
2  begin
3      for j := 1 to l do
4          hs_j := Copy(h_j); hp_j := Copy(h_j)
5      end
6      i := 1
7      while (i < l_gp) do
8          {s_1, ...., s_l} := GSimulate({hs_1, ..., hs_l}, {p_1[i+1], ..., p_l[i+1]})
9          DISM := false
10         j := 1
11         while (j ≤ l) and (DISM=false) do
12             RemoveFirstTwoElementsFrom(hs_j)
13             hs_j := hs_j + p_j[i+1] + s_j
14             RemoveFirstTwoElementsFrom(hp_j)
15             hp_j := hp_j + p_j[i+1] + p_j[i+2]
16             if (s_j ≠ p_j[i+2]) then
17                 dism := μ_DIT(PG_j)(hs_j, hp_j)
18                 if (dism is "high") then return "generate new g-plan"
19                 if (dism is not "low") then DISM := true
20             end
21             j := j + 1
22         end
23         if (DISM=true) then
24             gplan := GReconstruction({hs_1, ..., hs_l}, {p_1, ..., p_l}, j − 1)
25             if (gplan is empty ) "generate new g-plan"
26             for j := 1 to l do p_j := gplan[j]
27         end
28         i := i + 2
29     end
30     return {p_1, ..., p_l}
31 end
```

Finally, we present a plan reconstruction algorithm for a structured object on the level of the planning graph for objects of this type (see Algorithm 7.13).

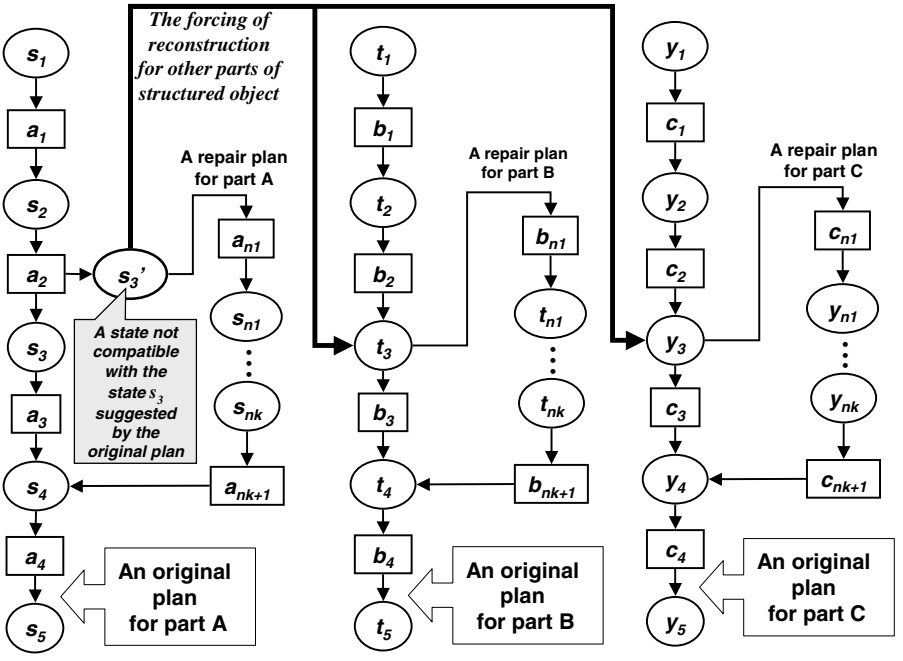


Fig. 49. The simulation of meta action execution with reconstruction for a structured object composed of parts: A, B, C

The Algorithm 7.13 tries to find a short repair plan P_2 (not longer than L_{rp}) which brings the initial state of the reconstruction (the last state in history H) to a state appearing in plan P_1, starting from position pos as far as position $pos+2\cdot(D_r-1)$. The maximum depth of reconstruction D_r is, therefore, the number of meta states in plan P_1 (starting from position pos), which the algorithm tries to reach using the repair plan. The repair plan is searched for by algorithm $EEFSS$ although it is possible to use also other planning algorithms.

Computational complexity of the above algorithm depends linearly on complexity of algorithm $EEFSS$. However, in practice using this algorithm may significantly accelerate the execution of meta plans which require reconstruction because instead of a total reconstruction, only a partial reconstruction of the meta-plan is performed whose degree of computational difficulty is much lower than the difficulty level of a total reconstruction (with regard to the smaller size of the problem).

7.18 Estimation of the Similarity between Plans

The problem of inducing classifiers for similarity relations is one of the challenging problems in data mining and knowledge discovery (see, *e.g.*,

Algorithm 7.12. Partially reconstruction of g-plan (*GReconstruction*)

Input:
- family of paths $h_1, ..., h_l$, where h_i is a path from the planning graph $\mathbf{PG}_i$, representing history of the object O_i during execution the last meta action for this object, for $i = 1, ..., l$,
- position *pos* of starting state of reconstruction in plans p_1, ..., p_l.

Output: The reconstructed g-plan $p_1, ..., p_l$

```
1  Procedure GReconstruction({h_1,...,h_l},{p_1,...,p_l},pos)
2  begin
3      j := pos
4      while (j ≤ pos + 2·(d_r − 1)) do
5          {q_1,...,q_l} := SearchGPlan({h_1,...,h_l},{p_1[j],...,p_l[j]},l_rp)
6          if ({q_1,...,q_l} is not empty) then
7              p_1 := Subpath(p_1,1,pos−1)+q_1+Subpath(p_1,j+1,length(p_1))
8              ...
9              p_l := Subpath(p_l,1,pos−1)+q_l+Subpath(p_l,j+1,length(p_l))
10             return {p_1,...,p_l}
11         end
12         j := j + 2
13     end
14     return "empty g-plan"
15 end
```

[162, 163, 164, 165, 166, 167, 168, 169, 170, 171]). The existing methods are based on building models for similarity functions using simple strategies for fusion of local similarities. The optimization of the assumed parameterized similarity formula is performed by tuning parameters relative to local similarities and their fusion. For instance, if we want to compare two medical plans of treatments, e.g., one plan generated automatically by our computer system and another one proposed by medical expert, we need a tool to estimate the similarity. This problem can be solved by introducing a function measuring the similarity between medical plans. For example, in the case of our medical data (see Section 7.21), a formula is used to compute a similarity between two plans as the arithmetic mean of similarity between all corresponding pairs of actions (nodes) from both plans, where the similarity for the single corresponding pair of actions is defined by a consistence measure of medicines and medical procedures comprised in these actions. For example, let $M = \{m_1, ..., m_k\}$ be a set consisting of k medicines. Let us assume that actions in medical plans are specified by subsets of M. Hence, any medical plan P determines a sequence of actions $\mathcal{A}(P) = (A_1, ..., A_n)$, where $A_i \subseteq M$ for $i = 1, \ldots, n$ and n is the number of actions in P. In our example, the similarity between plans is defined by a similarity function Sim established

Algorithm 7.13. Reconstruction of the plan for structured objects (*MReconstruction*)

Input:
- path H in the planning graph $\mathcal{PG}$ representing history of a given structured complex object O, that is finished by an initial state to start of reconstruction,
- family of paths $h_1, ..., h_l$, where h_i is a path from the planning graph $\mathbf{PG}_i$, representing history of the object O_i during execution the last meta action,
  for $i = 1, ..., l$,
- plan P_1 generated for a given structured object O before reconstruction,
- position *pos* of starting state of reconstruction in the plan P_1.

Output: The plan P_1 after reconstruction

```
1  Procedure MReconstruction(H, {h_1, ..., h_l}, P_1, pos)
2  begin
3      j := pos
4      while (j ≤ pos + 2 · (D_r − 1)) do
5          P_2 := EEFSS(H, {h_1, ..., h_l}, P_1[j], L_rp)
6          if (P_2 is not empty) then
7              P_3 :=
               Subpath(P_1, 1, pos − 1) + P_2 + Subpath(P_1, j + 1, length(P_1))
8              return P_3
9          end
10         j := j + 2
11     end
12     return "empty plan"
13 end
```

on pairs of medical plans (P_1, P_2) (of the same length) with the sequences of actions $\mathcal{A}(P_1) = (A_1, ..., A_n)$ and $\mathcal{A}(P_2) = (B_1, ..., B_n)$, respectively as follows

$$Sim(P_1, P_2) = \frac{1}{n} \sum_{i=1}^{n} \frac{|A_i \cap B_i| + |M \setminus (A_i \cup B_i)|}{|M|}.$$

However, such an approach seems to be very abstract and ad hoc, because it does not take into account any deeper knowledge about the similarity of plans, e.g., domain knowledge. Whereas, the similarity relations for real-life problems are usually more complex objects, i.e., their construction from local similarities cannot be obtained by simple fusion functions. Hence, such similarity relations cannot be approximated with the satisfactory quality by employing the existing simple strategies. For this reason we treat this similarity measure, Sim, only as an example and do not take into account in our further research (and in our proposed method). Whereas, to support the process of similarity relation approximation,

we propose to use domain knowledge represented by concept ontology expressed in natural language. The ontology consists of concepts used by expert in his explanation of similarity and dissimilarity cases. Approximation of the ontology makes it possible to obtain some relevant concepts for approximation of the similarity relation.

7.19 Ontology of the Similarity between Plans

According to the domain knowledge, it is quite common, that there are many aspects of similarity between plans. For example, in case of comparison of medical plans used for the treatment of infants with respiratory failure, we should take into consideration, *e.g.*, the similarity of the antibiotics use, the ventilation mode and the similarity of PDA closing (see Appendix B for mor medical details). Moreover, every aspect of the similarity should be understood in a different way. For example, in estimation of the similarity in the antibiotic treatment, it should be evaluated the kind of antibiotic, as well as the time of administration. Therefore, it is necessary to investigate and take into account all incompatibilities of the antibiotic use between corresponding pairs of nodes from both plans. Excessive doses are rather acceptable (based on expert knowledge), whilst the lack of medicine (if it is necessary) should be taken as a very serious mistake. In such situation, the difference in our assessment is estimated as very significant. A bit different interpretation of similarity should be used in case of the ventilation. As in antibiotic use, we investigate all incompatibilities of the ventilation mode between corresponding pairs of nodes from both plans. However, sometimes, according to expert knowledge, we simplified our assessments, *e.g.*, respiration unsupported and CPAP are estimated as similar (see Example 36 for more medical details). More complicated situation is present if we want to judge the similarity in treatment of PDA. We have to assign the ventilation mode, as well as the similarity of PDA closing procedure. In summary, any aspect of the similarity between plans should be taken into account in the specific way and the domain knowledge is necessary for joining all these similarities (obtained for all aspects). Therefore, the similarity between plans should be assigned on the basis of a special ontology specified in a dialog with human experts. Such ontology we call *similarity ontology*. Using such similarity ontology we developed methods for inducing classifiers predicting the similarity between two plans (generated automatically and proposed by human experts).

In the paper, we assume that each similarity ontology between plans has a tree structure. The root of this tree is always one concept representing general similarity between plans. In each similarity ontology there may exist concepts of two-way type. In this paper, the concepts of the first type will be called *internal concepts* of ontology. They are characterized by the fact that they depend on other ontology concepts. The concept of the second type will be called *input concepts* of ontology (in other words the concepts of the lowest ontology level). The input concepts are characterized by the fact that they do not depend on other ontology concepts. Fig. 50 shows an exemplary ontology of similarity between plans of the treatment of newborn infants with the respiratory failure.

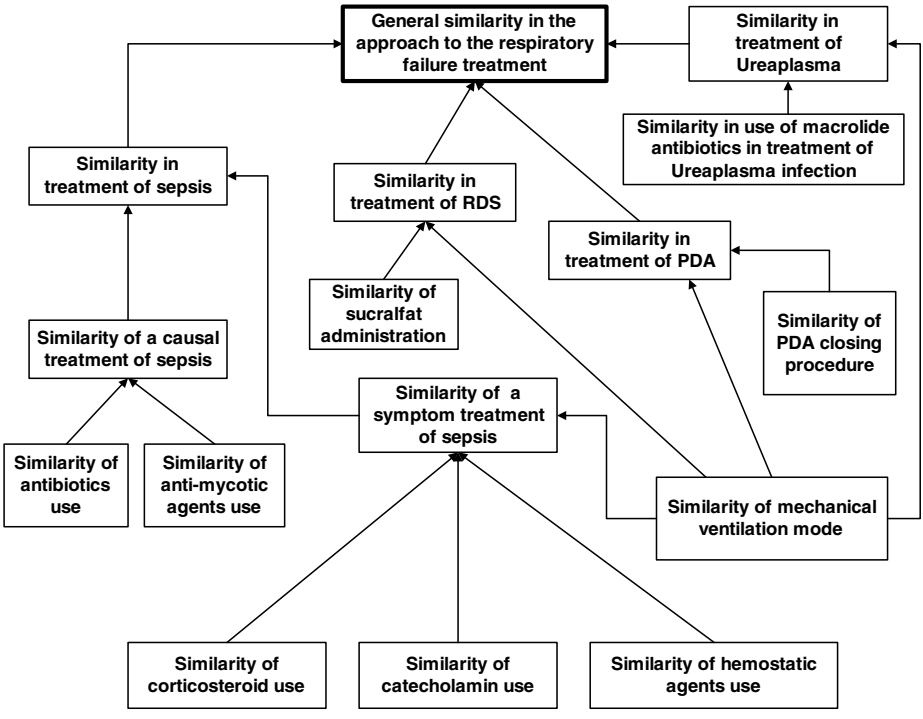


Fig. 50. An exemplary ontology of similarity between plans of the treatment of newborn infants with respiratory failure

This ontology has been provided by human experts. However, it is also possible to present some other versions of such ontology, instead of that presented above, according to opinions of some other group of human experts.

7.20 Similarity Classifier

Using the similarity ontology (*e.g.*, the ontology presented in Fig. 50), we developed methods for inducing classifiers predicting the similarity between two plans (generated automatically and proposed by human experts).

The method for construction of such classifier can be based on *a similarity table of plans*. The similarity table of plans is the decision table which may be constructed for any concept from the similarity ontology. The similarity table is created in order to approximate a concept for which the table has been constructed. The approximation of the concept takes place with the help of classifiers generated for the similarity table. However, because of the fact that in the similarity ontology there occur two types of concepts (internal and input), there are also two types of similarity tables. Similarity tables of the first type are constructed for internal concepts, whereas the tables of the second type are constructed for input concepts.

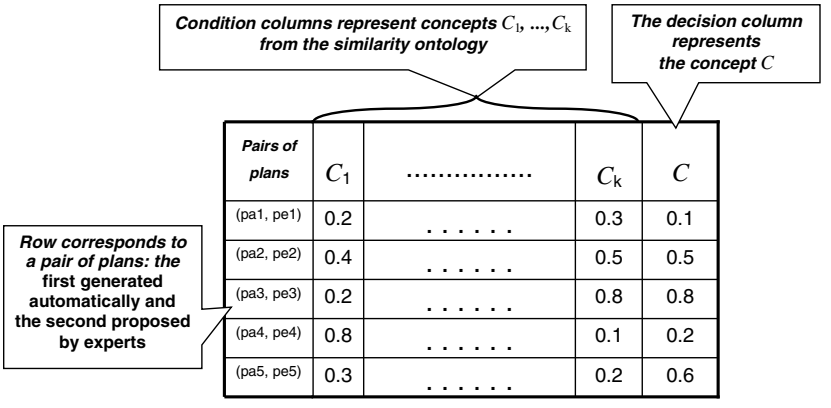


Fig. 51. The scheme of the similarity table of plans

Similarity tables for internal concepts of similarity ontology are constructed for a certain fragment of similarity ontology which consists of a concept of this ontology and concepts on which this concept depends. In the case of ontology from Fig. 50 it may be for instance the concept *Similarity of a symptom treatment of sepsis* and concepts *Similarity of corticosteroid use*, *Similarity of catecholamin use* and *Similarity of hemostatic agents use*. To simplify further discussion let us assume that it is the concept C that depends in the similarity ontology on the concepts C_1, ..., C_k. The aim of constructing a similarity table is approximation of concept C using concepts C_1, ..., C_k (see Fig. 51). Condition columns of such similarity table represent concepts C_1, ..., C_k. Any row corresponds to a pair of plans: generated automatically and proposed by experts. Values of all attributes have been provided by experts from the set $\{0.0, 0.1, ..., 0.9, 1.0\}$. Finally, the decision column represents the concept C.

Definition 71. *Let us assume that:*

- $\mathbf{T} = (U,\ A,\ a_{id},\ \leq_{a_{id}},\ a_t,\ \leq_{a_t},\ a_c)$ *is a temporal information system with actions,*
- $\mathbf{PG} = (S, A, E)$ *is a planning graph,*
- k *is a fixed length of plans in the planning graph* $\mathbf{PG}$*,*
- C *is an internal concept, dependent in some similarity ontology on the concepts* C_1,...,C_k*, where* $C, C_1, ..., C_k \subseteq PLAN(\mathbf{PG}, k) \times PLAN(\mathbf{PG}, k)$*.*

A similarity table of plans from planning graph $\mathbf{PG}$ *constructed for the internal concept* C *on the basis of the system* $\mathbf{T}$ *is a decision table* (U, A, d)*, where:*

- $U \subseteq DPLAN(\mathbf{T}, \mathbf{PG}, k) \times DPLAN(\mathbf{T}, \mathbf{PG}, k)$,
- $A = \{a_1, ..., a_m\}$ *is the set of attributes created on the basis of concepts from the family* $C_1, ..., C_m$ *such that for any* $i \in \{1, ..., m\}$ *values of* a_i *describe membership of objects from the set* U *to the concept* C_i *and these values are determined by experts from the set* $\{\ 0.0,\ 0.1,\ ....,\ 1.0\ \}$,

– *the values of decision attribute d which also belong to the set* { 0.0, 0.1, ..., 1.0 } *are proposed on the basis of the dissimilarity function value (proposed by an expert) of plans for individual objects from the set U.*

Let us notice that in the similarity table defined above there are no all the possible pairs of plans from set $DPLAN(\mathbf{T}, \mathbf{PG}, k) \times DPLAN(\mathbf{T}, \mathbf{PG}, k)$, but only a certain selected subset of the set of these pairs. In practice, this limitation is very necessary because the number of pairs of product $DPLAN(\mathbf{T}, \mathbf{PG}, k) \times DPLAN(\mathbf{T}, \mathbf{PG}, k)$ may be so large that the expert is not able to provide for them all values of decision attribute d. Therefore, usually in the similarity table there are only pairs selected by the expert which represent typical cases of determining similarity functions of plans which may be generalized using a classifier.

The stratifying classifier computed for a similarity table (called *a similarity classifier*) can be used to determine the similarity between plans (generated by our methods of automated planning and plans proposed be human experts) relatively to a given internal concept C.

Such stratifying classifiers may be constructed for all concepts from the similarity ontology which depend, in this ontology, on other concepts. However, we also need stratifying classifiers for input concepts of ontology, that is, those lying on the lowest level of the ontology. Hence, they are the concepts which do not depend on other concepts in this ontology. To approximate them we do not use other ontology concepts but we apply the features of comparable plans which are expressed in the form of patterns defined in the language $FPPG$ (see Section 7.6). Obviously, such types of patterns are also concepts determined in the set of pairs of plans. However, they are usually not placed in the similarity ontology between plans. Therefore, approximation tables of input concepts of the similarity ontology should be treated as a specific type of similarity table.

Definition 72. *Let us assume that:*

– $\mathbf{T} = (U,\ A,\ a_{id},\ \leq_{a_{id}},\ a_t,\ \leq_{a_t},\ a_c)$ *is a temporal information system with actions,*
– $\mathbf{PG} = (S, A, E)$ *is a planning graph,*
– k *is a fixed length of plans in the planning graph* $\mathbf{PG}$*,*
– $\phi_1, ..., \phi_m \in FPPG(\mathbf{PG})$ *is a family of formulas defined by experts,*
– $C \subseteq PLAN(\mathbf{PG}, k) \times PLAN(\mathbf{PG}, k)$ *is an input concept of the similarity ontology between plans.*

A similarity table of plans from planning graph $\mathbf{PG}$ *constructed for the input concept* C *on the basis of the system* $\mathbf{T}$ *is a decision table* (U, A, d)*, where:*

– $U \subseteq DPLAN(\mathbf{T}, \mathbf{PG}, k) \times DPLAN(\mathbf{T}, \mathbf{PG}, k)$,
– $A = \{a_1, ..., a_m, a_{m+1}, ..., a_{2m}\}$ *is a set of attributes created on the basis of formulas* $\phi_1, ..., \phi_m$*, where for any* $i \in \{1, ..., 2m\}$ *values of* a_i *are computed in the following way:*

$$\forall p = (p_1, p_2) \in U : a_i(p) = \begin{cases} 1 & \text{if } i \leq m \text{ and } p_1 \models_{FPPG(\mathbf{PG})} \phi_i \\ 1 & \text{if } i > m \text{ and } p_2 \models_{FPPG(\mathbf{PG})} \phi_i \\ 0 & \text{otherwise} \end{cases},$$

– *the values of decision attribute d describe membership of objects from the set U to the concept C and these values are determined by experts from the set* { 0.0, 0.1, ..., 1.0 }.

Let us notice that the similarity table defined above is constructed in the way that concept C is approximated on the basis of the features of both plans corresponding to a given object from the set U.

It is worth noticing that for approximation of complex concepts from the similarity ontology one can use also features (attributes) describing relations between plans. Such features are formulated in a natural language using special questions about both plans. Examples of such questions are: *Were antibiotics used simultaneously in both plans?*, *Was the average difference between mechanical ventilation mode in both plans significant?*. However, it requires a simple extension of the language $FPPG(\mathbf{PG})$.

Classifiers constructed for similarity tables corresponding to all concepts from the similarity ontology may be used to construct a complex classifier which gives the general similarity between plans (represented by the concept lying in the root of the similarity ontology). We provide an example of how such a classifier works. Let us assume that there is a certain similarity ontology between pairs of plans in which there occur six following concepts: C_1, C_2, C_3, C_4, C_5 and C_6. The concept C_1 depends on concepts C_2 and C_3, the concept C_2 depends on concepts C_4 and C_5, and the concept C_3 depends on concepts C_5 and C_6. In this ontology concept C_1 is the concept of general similarity between plans, whereas concepts C_4, C_5 and C_6 are input concepts of the similarity ontology (see Fig. 52).

Firstly, we construct similarity tables for concepts C_4, C_5, C_6 and stratifying classifiers μ_{C_4}, μ_{C_5}, μ_{C_6} corresponding to them. Let us also assume that there are given stratifying classifiers μ_{C_1}, μ_{C_2}, μ_{C_3} which were constructed for similarity tables which correspond to concepts C_1, C_2 and C_3. Tested object $u = (p_1, p_2)$ which is a pair of compared plans is classified to the layer of concept C corresponding to it in the following way. At the beginning, the object u is classified by classifiers μ_{C_4}, μ_{C_5} and μ_{C_6}. This way we obtain values $\mu_{C_4}(u)$, $\mu_{C_5}(u)$ and $\mu_{C_6}(u)$. Next, values $\mu_{C_4}(u)$ and $\mu_{C_5}(u)$ are used as the values of conditional attributes in the similarity table constructed for concept C_2. Thus, the object u may be classified by classifier μ_{C_2}, which gives us value $\mu_{C_2}(u)$. At the same

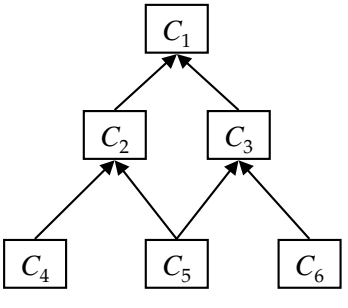


Fig. 52. The scheme of a simple similarity ontology

time, values $\mu_{C_5}(u)$ and $\mu_{C_6}(u)$ are used as the values of conditional attributes in the similarity table constructed for concept C_3. It gives the possibility to classify object u by classifier μ_{C_3} and obtain value $\mu_{C_3}(u)$. Finally, values $\mu_{C_2}(u)$ and $\mu_{C_3}(u)$ are used as the values of conditional attributes of the similarity table constructed for concept C_1. Thus, the object u may be classified by classifier μ_{C_1} to layer $\mu_{C_1}(u)$.

The complex classifier described above can be used to determine the general similarity between plans generated by our methods of automated planning and plans proposed by human experts, *e.g.*, during the real-life clinical treatment (see Section 7.21).

7.21 Experiments with Medical Data

To verify the effectiveness of presented in this paper methods of automated planning, we have implemented the algorithms in an *Automated Planning* library (AP-lib), which is an extension of the RSES-lib library forming the computational kernel of the RSES system (see [15]).

It should be emphasized that, in general, automated planning of treatment is a very difficult and complicated task because it requires extensive medical knowledge combined with sensor information about the state of a patient. Even so, the proposed approach makes it possible to obtain quite satisfactory results in the short-term planning of treatment of infants with respiratory failure. The reason is that medical data sets have been accurately prepared for purposes of our experiments using the medical knowledge. For example, the collection of medical actions, that are usually used during the treatment of infants with respiratory failure, has been divided into a few groups of similar actions (for example: antibiotics, anti-mycotic agents, mechanical ventilation, catecholamines, corticosteroids, hemostatic agents). It is very helpful in the prediction of actions because the number of actions is significantly decreased.

The experiments have been performed on the medical data sets obtained from Neonatal Intensive Care Unit, First Department of Pediatrics, Polish-American Institute of Pediatrics, Collegium Medicum, Jagiellonian University, Krakow, Poland (see also Section 6.26). We used one data table, that consists of 11099 objects. Each object of this table describes parameters of one patient in single time point. There were prepared 7022 situations on the basis of this data table, where the plan of treatment has been proposed by human experts during the real-life clinical treatment.

We have applied the *train-and-test* method. However, analogously to experiments from Subsection 6.26 the method of dividing data differed slightly from the standard method described in Section 2.9. Namely, in each experiment the whole set of patients was randomly divided into two groups (training and tested one). Each of these groups allowed creating approximately 4000 time windows which have duration of 7 time points. Time windows created on the basis of patients from the training part created a training table for a given experiment (when plans of treatment have been assigned), whereas time windows created on the basis of patients from the tested part created a test table for the experiment

(when plans have been generated by automated method and expert plans are known in order to compare both plans).

In the discussed experiments, the distance between time points recorded for a specific patient was constant (one day). In a single experiment concerning a patient's treatment, a 7-point sequence of time points was used. In terms of planning the treatment each such sequence may be written as s_1, a_1, s_2, a_2, s_3, a_3, s_4, a_4, s_5, a_5, s_5, a_6, s_7, where s_i (for $i = 1, ..., 7$) is a patient state and a_i (for $i = 1, ..., 6$) is a complex medical action performed in the state s_i. The first part of the above sequence of states and actions, that is, from state s_1 to state s_3, was used by the method of automated planning as the input information (corresponding to the values of conditional attributes in the classic approach to constructing classifiers). The remaining actions and states were automatically generated to create plan (s_3, a'_3, s'_4, a'_4, s'_5, a'_5, s'_6, a'_6, s'_7). This plan may be treated as a certain type of a complex decision value. Verification of the quality of the generated plan consisted in comparing plan (s_3, a'_3, s'_4, a'_4, s'_5, a'_5, s'_6, a'_6, s'_7) with plan (s_3, a_3, s_4, a_4, s_5, a_5, s_5, a_6, s_7). It is worth adding that a single complex action concerned one time point, meta action concerned two time points and a single experiment consisted in planning two meta actions. Hence, in a single experiment four actions were planned (patient's treatment for four days). In other words, at the beginning of the automated planning procedure the information about the patient's state in the last three days of his hospitalization was used (s_1, s_2, s_3) together with the information about complex medical actions undertaken one or two days before (a_1, a_2). The generated plan included information about a suggested complex medical action on a given day of hospitalization (a'_3), information about actions which should be undertaken in the three following days of hospitalization (a'_4, a'_5, a'_6) and information about the patient's state anticipated as a result of the planned treatment in the four following days of hospitalization (s'_4, s'_5, s'_6, s'_7).

As a measure of planning success (or failure) in our experiments, we use the special classifier that can predict the similarity between two plans as a number between 0.0 (very low similarity between two plans) and 1.0 (very high similarity between two plans) (see Section 7.20). We use this classifier to determine the similarity between plans generated by our methods of automated planning and plans proposed be human experts during the real-life clinical treatment. In order to determine the standard deviation of the obtained results each experiment was repeated for 10 random divisions of the whole data set.

The average similarity between plans for all tested situations was **0.802**. The corresponding standard deviations was **0.041**. The coverage of tested situation by generated plans was **0.846** with standard deviation **0.018**.

Due to the fact that the average similarity is not too high (less than 0.9) and the standard deviation is relatively high for our algorithm, we present also the distribution of the results. We describe results in such a way that we present how many generated plans belong to the specified interval of similarity. For this reason we divided interval [0.0, 1.0] into 5 equal intervals, i.e., [0.0, 0.2], [0.2, 0.4], [0.4, 0.6], [0.6, 0.8] and [0.8, 1.0]. Table 10 shows the average percent of the

Table 10. The average percent of plans belonging to the specified interval and the average similarity of plans in this interval

Intervals	Average percent of plans	Average similarity of plans
[0.0, 0.2]	**12.1%** ± 4.5%	**0.139** ± 0.002
(0.2, 0.4]	**6.2%** ± 1.5%	**0.349** ± 0.003
(0.4, 0.6]	**7.1%** ± 1.7%	**0.563** ± 0.002
(0.6, 0.8]	5.8% ± 0.9%	0.773 ± 0.004
(0.8, 1.0]	68.9% ± 5.6%	0.987 ± 0.002

plans belonging to the specified interval and the average similarity of plans in this interval.

It is easy to see that some group of plans generated automatically is not enough similar to the plans proposed by the experts. If we assume that inadequate similarity is lower than 0.6, in this group we found about 25% of all plans (see Table 10). To explain this issue, we should observe more carefully plans, which are incompatible with the proposals prepared by experts. In practice, the main medical actions influencing the similarity of plans in accordance with ontology of the similarity from Fig. 50 are mechanical ventilation, antibiotics, anti-mycotic agents and macrolide antibiotics. Therefore, it may be interesting how the treatment similarity changed in the range of applying these actions in the individual intervals of similarity between the plans.

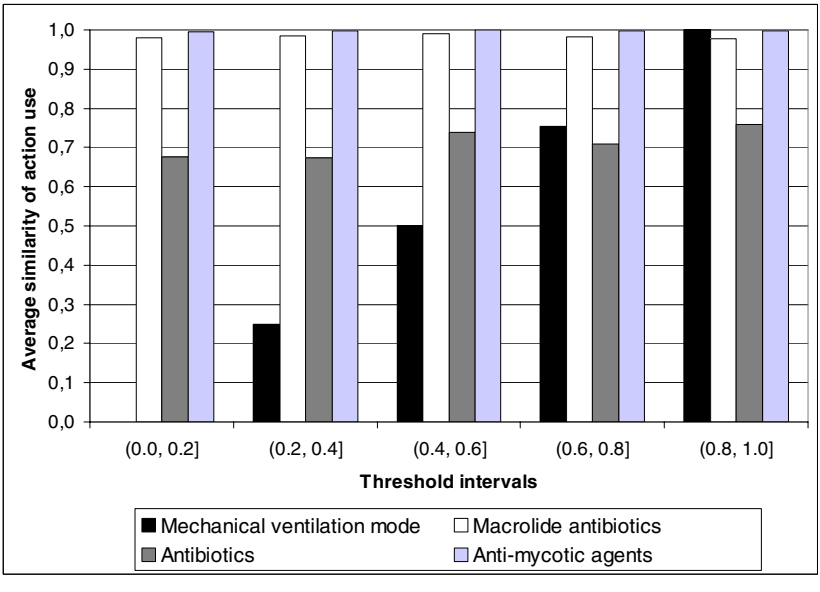


Fig. 53. The average similarity of plans in the specified interval for medical actions

On Fig. 53 we can see that a significant incompatibility of treatment plans most often concerns mechanical ventilation and perhaps antibiotic therapy - the situation when a patient develops a sudden and severe infection (*e.g.*, sepsis). Such circumstances cause rapid exacerbation of respiratory failure are required higher level of mechanical ventilation and immediate antibiotic treatment. For example, although microbiological confirmation of current infection is achieved after 2–3 days, physician starts treatment after first symptoms of suspected disease and often intensify mechanical ventilation mode. It would seem that the algorithms of automated planning presented in this paper may imitate the strategy of treatment described above. Unfortunately, in practice, these algorithms are not able to learn this strategy for a lot of information because they were not introduced to the base records or were introduced with delay. For instance, hemoglobin saturation which is measured for the whole time, as the dynamic marker of patients respiratory status, was not found in the data, whilst results of arterial blood gases were introduced irregularly, with many missing values. So, the technical limitation of the current data collection lead to the intensive work modifying and extending both, the equipment and software, served for gathering clinical data. It may be expected that in several years the automated planning algorithms, described in this paper, will achieve much better and useful results.

A separate problem is a relatively low coverage of the algorithms described in this paper which equals averagely 0.846. Such a low coverage results from the specificity of the automated planning method used which synchronizes the treatment of four diseases (RDS, PDA, sepsis and Ureaplasma). We may identify two reasons of a low coverage. Firstly, because of data shortage the algorithm in many situations may not synchronize the treatment of the above mentioned diseases. It happens this way because each proposed comparison of plans may be debatable in terms of the knowledge gathered in the system. Therefore, in these cases the system does not suggest any treatment plan and says *I do not know*. The second reason for low coverage is the fact that the automated planning method used requires application of a complex classifier which consists of many classifiers of lesser complexity. Putting these classifiers together often causes the effect of decreasing the complex classifier coverage. For instance, let us assume that making decision for tested object u requires application of complex classifier μ, which consists of two classifiers μ_1 and μ_2. We apply classifier μ_1 directly to u, whereas classifier μ_2 is applied to the results of classification of classifier μ_1. In other words, to make classifier μ_2 work for a given tested object u we need value $\mu_1(u)$. Let us assume that the coverage for classifiers μ_1 and μ_2 equals respectively 0.94 and 0.95. Hence, the coverage of classifier μ is equal $0.94 \cdot 0.95 = 0.893$, that is the coverage of classifier μ is smaller than the coverage of classifier μ_1 as well as the coverage of classifier μ_2.

It is worth noticing that applying other automated planning methods (see [353, 354]), a higher coverage for the data sets analyzed here may be obtained. However, in this paper we prefer algorithms *EEFS* and *EEFSS*, because they need not only data sets but also great domain knowledge to work. Moreover, the quality of their performance depends greatly on the provided domain

knowledge. Therefore, it may be expected that providing greater and more reliable knowledge and more extensive data sets on the input of this algorithm will allow the quality of automatically generated treatment plans to be more similar to the quality of treatment plans proposed by the medical experts.

In summation, we conclude that experimental results showed that the proposed automated planning method gives good results, also in the opinion of medical experts (compatible enough with the plans suggested by the experts), and may be applied in medical practice as a supporting tool for planning the treatment of infants suffering from respiratory failure.

8 Summary

The aim of this paper was to present new methods of approximating complex concepts on the basis of experimental data and domain knowledge which is mainly represented using concept ontology.

At the beginning of the paper a number of methods of constructing classical classifiers were overviewed (see Section 2) and methods of constructing stratifying classifiers were presented (see Section 3). Next, in Section 4, a general methodology of approximating complex concepts with the use of data sets and domain knowledge was presented. In the further part of the paper, this methodology was applied to approximate spatial complex concepts (see Section 5), spatio-temporal complex concepts for unstructured and structured objects (see Section 6), to identify the behavioral patterns for this type of objects (see Section 6), and to the automated planning of behavior of such objects when the states of objects are represented by spatio-temporal concepts which require an approximation (see Section 7).

We have also described the results of computer experiments conducted on real-life data sets which were obtained from the road traffic simulator (see Appendix A) and on medical data which were made available by Neonatal Intensive Care Unit, First Department of Pediatrics, Polish-American Institute of Pediatrics, Collegium Medicum, Jagiellonian University, Krakow, Poland.

In light of theoretical discourse and the results of computer experiments presented in the paper the following conclusions may be drawn:

1. The methodology of approximating complex concepts with the use of data sets and domain knowledge (represented mainly by a concept ontology), which was proposed in this paper (see Section 4), fills the gap which exists between spatio-temporal complex concepts and sensor data. It enables an effective approximation of complex concepts; however, it requires a domain knowledge represented mainly in the form of a concept ontology.
2. Stratifying classifiers proposed in the paper (see Section 3) are effective tools for stratifying concepts, that is, they enable to classify objects to different layers of the concept corresponding to different degrees of certainty of membership of a tested object to the concept. Particularly noteworthy here is the new method of constructing such classifiers based on shortening of

decision rules with respect to different coefficients of consistency. It makes an automatic stratification of concepts possible.

3. The method of approximation of complex spatial concepts, described in the paper, with the help of approximate reasoning schemes (AR-schemes) leads to better results than the classical methods based on decision rules induced directly from sensor data because the quality of classifier classification based on AR-schemes is higher than the quality of classification obtained by classifiers based on decision rules, particularly for small decision classes representing atypical cases in the recognition of which we are most interested in, *e.g.*, a dangerous driving vehicle on a highway (see Section 5). Moreover, for larger data sets, the time of constructing classifiers based on AR-schemes is much shorter than the time of inducing classifiers based on decision rules, and the structure of classifiers based on AR-schemes is less complex than the structure of classifiers based on decision rules. It is also worth mentioning that the classifiers based on AR-schemes are more robust (stable or tolerant) when it comes to changes in training data sets serving the construction of classifiers, that is, a classifier based on AR-schemes, constructed for one data set, often proves itself good for another data set. For example, a classifier constructed for data generated from the traffic simulator with one simulation scenario proves itself useful in classification of objects generated by the simulator with the use of another simulation scenario.
4. The methodology of modeling complex object behavior with the use of behavioral graphs of these objects, proposed in the paper (see Section 6), is a convenient and effective tool for identifying behavioral patterns of complex objects. On the one hand this methodology, enables to represent concepts on a high abstraction level, and on the other hand, owing to the use of a domain knowledge, it enables to approximate these concepts on the basis of sensor data and using a domain knowledge.
5. The sweeping method around complex objects is a very fast and convenient method of isolating objects with complex structure from complex dynamical systems (see Section 6). A certain difficulty in using this method is the fact that each of its applications requires appropriate sweeping heuristics which must be proposed by the expert.
6. The method of eliminating complex objects in behavioral pattern identification based on the rules of fast elimination of behavioral patterns greatly accelerates the identification of behavioral patterns in complex dynamical systems monitoring (see Section 6). A certain inconvenience of applying this method is the fact that although it is based on rules of fast elimination of behavioral patterns obtained from data, in order to determine those rules we need special temporal patterns supporting the process of elimination, and the patterns themselves must be proposed by experts.
7. The methods of automated planning of complex object behavior proposed in the paper facilitate an effective planning of behavior of both unstructured and structured objects whose states are defined in a natural language using vague spatio-temporal conditions (see Section 7). The authenticity of conditions of this type is usually not possible to be verified on the basis

of a simple analysis of available information about the object and that is why these conditions must be treated as spatio-temporal complex concepts and their approximation requires methods described in this paper which are based on data sets and domain knowledge.

8. The method of solving conflicts between actions during automated planning of complex object behavior, based on a domain knowledge and data sets (see Section 7), is ideal for the situation where the choice of one action from many possible actions, to be performed at a given state should not be random but it should be made in accordance with the domain knowledge (as in the example of planning of a patient's treatment). A significant novelty of the method presented here in relation to the already existing ones is the fact that in order to solve conflicts between actions we use classifiers based on data sets and domain knowledge.
9. The method of synchronizing plans constructed for parts of a structured object, described in the paper, is an important element of the method of planning of structured object behavior (see Section 7). If we assume that plans constructed for parts of a structured object are processes of some kind, then the method of synchronizing those plans is a method of synchronizing processes corresponding to the parts of the structured object. It should be emphasized, however, that the significant novelty of the method of synchronizing processes presented herein in relation to the ones known from literature is the fact that the synchronization is carried out by using classifiers determined on the basis of data sets and domain knowledge.
10. The method of partial reconstruction of a plan, proposed in the paper, can accelerate the execution of plans constructed for complex objects significantly (see Section 7). It is due to the fact that in the case of incompatibility of the state occurring in the plan with the actual state of the complex object, the plan needs not be generated from the beginning but it may be reconstructed using the repair plan.
11. The method of similarity relation approximation based on data set usage and a domain knowledge expressed in the form of a concept ontology which has frequently been used throughout our research is an important proposal of solving a difficult problem of similarity relation approximation.

The main advantage of the methods of construction of classifiers for spatial and spatio-temporal complex concepts, presented in the paper, is the fact that besides data sets they also intensely use a domain knowledge expressed, above all, in the form of a concept ontology. Although using a domain knowledge causes an increase in effectiveness of the methods presented in this paper, it can be, however, a source of many difficulties arising during the application of those methods. The difficulties arise from the fact that the domain knowledge is often available only to the experts from the given domain. Therefore, projects which use the methods described in this paper require an active participation of experts from different domains. Hence, difficulties may often occur to incline experts who are consent to devote their time to participate in a computer science project.

In order to overcome these difficulties, there is a need of construction of special methods of data analysis which might be useful in automatic search of knowledge to replace, at least partially, the domain knowledge obtained from experts. In the case of methods presented herein, we mean the methods of data analysis which allow an automatic detection of concepts, significant for construction of complex classifiers. For example, we are concerned with the following data analysis methods here:

1. Automatic detection of spatial or spatio-temporal concepts which may be used in approximating other more complex concepts.
2. Automatic detection of complex spatio-temporal concepts occurring in behavioral patterns (understood as behavioral graphs) and temporal dependencies among those concepts.
3. Automatic detection of complex spatio-temporal concepts which are states of unstructured or structured objects in automated planning of behavior of such objects.
4. Automatic detection of complex spatio-temporal concepts which are meta-actions to be performed for structured objects in automated planning of behavior of such objects.

We plan to construct the data analysis methods mentioned above in the future.

In summation, it may be concluded that in executing real-life projects related to the construction of the intelligent systems supporting decision-making, apart from data sets it is necessary to apply domain knowledge. Without its application successful execution of many such projects becomes extremely difficult or impossible. On the other hand, appropriate space must be found for the automated methods of classifier construction wherever it is feasible. It means, thus, finding a certain type of "the golden mean" to apply appropriate proportions in domain knowledge usage and automated methods of data analysis. Certainly, it will determine the success or failure of many projects.

Acknowledgement

I would like to express my deepest gratitude to my mentor, Professor Andrzej Skowron for the inspiration and comprehensive support during writing this paper. It goes without saying that without his great involvement in scientific research and creative influence he exerts on the scientific development of the Group of Mathematical Logic at the Institute of Mathematics, Warsaw University which remains under his supervision, this work would never have been written.

I am grateful to my wife, Anna, for her love and unconditional support which, without a doubt, were some of the main factors without which this paper would not be possible.

I would like to extend my gratefulness to Professor James F. Peters (Department of Electrical and Computer Engineering, University of Manitoba, Canada), Dr. Anna Gomolińska (Department of Mathematics, University of Białystok,

Poland) and Dr. Piotr Synak (Polish-Japanese Institute of Information Technology, Warsaw, Poland) for their insights and many helpful comments and corrections made in this paper.

I would also like to thank Professor Jacek Pietrzyk and Dr. Piotr Kruczek (both from First Department of Pediatrics, Polish-American Institute of Pediatrics, Collegium Medicum, Jagiellonian University, Krakow, Poland) for providing for computer experiments valuable medical data which have been collected for many years in Neonatal Intensive Care Unit, First Department of Pediatrics, Polish-American Institute of Pediatrics, Collegium Medicum, Jagiellonian University, Krakow, Poland.

Special gratitude also goes to my sister Dr. Stanisława Bazan-Socha (Second Department of Internal Medicine, Collegium Medicum, Jagiellonian University, Krakow, Poland) and Dr. Piotr Kruczek for many invaluable discussions which helped me enormously to understand medical issues which was indispensable in preparation of medical data for the need of computer experiments.

Many of the results presented or mentioned in this paper are the results of long cooperation with other co-authors including Stanisława Bazan-Socha, Grzegorz Góra, Piotr Kruczek, Rafał Latkowski, Hung Son Nguyen, Sinh Hoa Nguyen, Antoni Osmólski, James F. Peters, Jacek J. Pietrzyk, Andrzej Skowron, Jarosław Stepaniuk, Piotr Synak, Marcin S. Szczuka, Dominik Ślęzak, Roman Świniarski, Hui Wang, Arkadiusz Wojna, Marcin Wojnarski, Jakub Wróblewski. At this point I want to thank them all for effective and pleasurable cooperation.

The research has been partially supported by the grant 3 T11C 002 26 from Ministry of Scientific Research and Information Technology of the Republic of Poland, by the grant N N516 368334 from Ministry of Science and Higher Education of the Republic of Poland and by the grant Innovative Economy Operational Programme 2007-2013 (Priority Axis 1. Research and development of new technologies) managed by Ministry of Regional Development of the Republic of Poland.

References

1. Murray, J.A., Bradley, H., Craigie, W., Onions, C.: The Oxford English Dictionary. Oxford University Press, Oxford (1933)
2. Devlin, K.: Logic and Information. Cambridge University Press, Cambridge (1991)
3. Joseph, H.W.B.: An Introduction to Logic. Clarendon Press, Oxford (1916)
4. Ogden, C.K., Richards, I.A.: The Meaning of Meaning. A Study of the Influence of Language Upon Thought and of the Science of Symbolism. Harcourt, Brace and Company, New York (1923)
5. Mendelson, E.: Introduction to Mathematical Logic. International Thomson Publishing (1987)
6. Friedman, J., Hastie, T., Tibshirani, R.: The Elements of Statistical Learning: Data Mining, Inference and Prediction, vol. I-V. Springer, Heidelberg (2001)
7. Gabbay, D.M., Hogger, C.J., Robinson, J.A. (eds.): Handbook of Logic in Artificial Intelligence and Logic Programming, vol. I-V. Oxford University Press, New York (1994)

8. Ignizio, J.P.: An Introduction to Expert Systems. McGraw-Hill, New York (1991)
9. Kloesgen, E., Zytkow, J. (eds.): Handbook of Knowledge Discovery and Data Mining. Oxford University Press, Oxford (2002)
10. Michalski, R., et al. (eds.): Machine Learning, vol. I-IV. Morgan Kaufmann, Los Altos (1983, 1986, 1990, 1994)
11. Michie, D., Spiegelhalter, D.J., Taylor, C.C.: Machine learning, neural and statistical classification. Ellis Horwood Limited, England (1994)
12. Mitchel, T.M.: Machine Learning. McGraw-Hill, Boston (1997)
13. Ripley, B.D.: Pattern Recognition and Neural Networks. Cambridge University Press, Cambridge (1996)
14. Bazan, J.G., Szczuka, M.: The Rough Set Exploration System. In: Peters, J.F., Skowron, A. (eds.) Transactions on Rough Sets III. LNCS, vol. 3400, pp. 37–56. Springer, Heidelberg (2005)
15. RSES: Project web site, `http://logic.mimuw.edu.pl/~rses`
16. Pawlak, Z.: Rough Sets: Theoretical Aspects of Reasoning about Data. D: System Theory, Knowledge Engineering and Problem Solving, vol. 9. Kluwer Academic Publishers, Dordrecht (1991)
17. Pawlak, Z., Skowron, A.: Rudiments of rough sets. Information Sciences 177, 3–27 (2007)
18. Peters, J.F., Skowron, A.: Zdzisław Pawlak life and work (1926–2006). Information Sciences 177, 1–2 (2007)
19. Brown, E.M.: Boolean Reasoning. Kluwer Academic Publishers, Dordrecht (1990)
20. Pawlak, Z., Skowron, A.: Rough sets and boolean reasoning. Information Sciences 177, 41–73 (2007)
21. Skowron, A., Rauszer, C.: The discernibility matrices and functions in information systems. In: Słowiński, R. (ed.) Intelligent Decision Support - Handbook of Applications and Advances of the Rough Sets Theory. D: System Theory, Knowledge Engineering and Problem Solving, vol. 11, pp. 331–362. Kluwer Academic Publishers, Dordrecht (1992)
22. Borrett, S.R., Bridewell, W., Langley, P., Arrigo, K.R.: A method for representing and developing process models. Ecological Complexity 4, 1–12 (2007)
23. Bridewell, W., Langley, P., Todorovski, L., Dzeroski, S.: Inductive process modeling. Machine Learning (to appear, 2008)
24. Langley, P.: Cognitive architectures and general intelligent systems. AI Magazine 27, 33–44 (2006)
25. Langley, P., Cummings, K., Shapiro, D.: Hierarchical skills and cognitive architectures. In: Proceedings of the Twenty-Sixth Annual Conference of the Cognitive Science Society, Chicago, IL, pp. 779–784 (2004)
26. Langley, P., Laird, J.E.: Cognitive architectures: Research issues and challenges. Technical report, Institute for the Study of Learning and Expertise, Palo Alto, CA (2002)
27. Langley, P., Laird, J.E., Rogers, S.: Cognitive architectures: Research issues and challenges. Technical report, Computational Learning Laboratory, CSLI, Stanford University, Palo Alto, CA (2006)
28. Langley, P., Shiran, O., Shrager, J., Todorovski, L., Pohorille, A.: Constructing explanatory process models from biological data and knowledge. AI in Medicine 37, 191–201 (2006)
29. Pal, S.K., Polkowski, L., Skowron, A. (eds.): Rough-Neural Computing: Techniques for Computing with Words. Cognitive Technologies. Springer, Heidelberg (2003)

30. Pancerz, K., Suraj, Z.: Discovering concurrent models from data tables with the ROSECON system. Fundamenta Informaticae 60, 251–268 (2004)
31. Pancerz, K., Suraj, Z.: Discovery of asynchronous concurrent models from experimental tables. Fundamenta Informaticae 61(2), 97–116 (2004)
32. Pancerz, K., Suraj, Z.: Rough sets for discovering concurrent system models from data tables. In: Hassanien, A.E., Suraj, Z., Ślęzak, D., Lingras, P. (eds.) Rough Computing: Theories, Technologies and Applications. Idea Group, Inc. (2007)
33. Pat, L., George, D., Bay, S., Saito, K.: Robust induction of process models from time-series data. In: Fawcett, T., Mishra, N. (eds.) Proceedings of the Twentieth International Conference on Machine Learning, Washington, D.C, pp. 432–439. AAAI Press, Menlo Park (2003)
34. Skowron, A., Suraj, Z.: Discovery of concurrent data models from experimental tables: A rough set approach. In: Fayyad, U.M., Uthurusamy, R. (eds.) Proceedings of the First International Conference on Knowledge Discovery and Databases Mining (KDD 1995), pp. 288–293. AAAI Press, Menlo Park (1995)
35. Soar: Project web site, `http://sitemaker.umich.edu/soar/home`
36. Suraj, Z.: Discovery of concurrent data models from experimental tables. Fundamenta Informaticae 28, 353–376 (1996)
37. Suraj, Z.: The synthesis problem of concurrent systems specified by dynamic information systems. In: Polkowski, L., Skowron, A. (eds.) Rough Sets in Knowledge Discovery 2. Applications, Case Studies and Software Systems. Studies in Fuzziness and Soft Computing, pp. 418–448. Physica-Verlag, Heidelberg (1998)
38. Suraj, Z.: Rough set methods for the synthesis and analysis of concurrent processes. In: Polkowski, L., Tsumoto, S., Lin, T.Y. (eds.) Rough set methods and applications: new developments in knowledge discovery in information systems. Studies in Fuzziness and Soft Computing, pp. 379–488. Physica-Verlag, Heidelberg (2000)
39. Suraj, Z.: Discovering concurrent data models and decision algorithms from data: A rough set approach. International Journal on Artificial Intelligence and Machine Learning IRSI, 51–56 (2004)
40. Unnikrishnan, K.P., Ramakrishnan, N., Sastry, P.S., Uthurusamy, R.: Service-oriented science: Scaling escience impact. In: Proceedings of the Fourth KDD Workshop on Temporal Data Mining: Network Reconstruction from Dynamic Data, The Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data (KDD 2006), Philadelphia, USA, August 20-23 (2006)
41. Breiman, L.: Statistical modeling: the two cultures. Statistical Science 16(3), 199–231 (2001)
42. Poggio, T., Smale, S.: The mathematics of learning: Dealing with data. Notices of the American Mathematical Society (AMS) 5, 537–544 (2003)
43. Vapnik, V. (ed.): Statistical Learning Theory. Wiley, New York (1998)
44. Zadeh, L.A.: From computing with numbers to computing with words – from manipulation of measurements to manipulation of perceptions. IEEE Transactions on Circuits and Systems – I: Fundamental Theory and Applications 1, 105–119 (1999)
45. Zadeh, L.A.: A new direction in AI: Toward a computational theory of perceptions. AI Magazine 1, 73–84 (2004)
46. Zadeh, L.A.: Toward a generalized theory of uncertainty (GTU) - an outline. Information Sciences 171, 1–40 (2005)

47. Ambroszkiewicz, S., Bartyna, W., Faderewski, M., Terlikowski, G.: An architecture of multirobot system based on software agents and the SOA paradigm. In: Czaja, L. (ed.) Proceedings of the Workshop on Concurrency, Specification, and Programming (CS&P 2007), Łagów, Poland, Warsaw, Poland, Warsaw University, September 27–29, pp. 21–32 (2007)
48. Domingos, P.: Toward knowledge-rich data mining. Data Mining and Knowledge Discovery 1, 21–28 (2007)
49. Foster, I.T.: Service-oriented science: Scaling escience impact. In: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Hong Kong, China, December 18-22. IEEE Computer Society, Los Alamitos (2006)
50. Kriegel, H.P., Borgwardt, K.M., Kröger, P., Pryakhin, A., Schubert, M., Zimek, A.: Future trends in data mining. Data Mining and Knowledge Discovery 1, 87–97 (2007)
51. Kuipers, B.: The spatial semantic hierarchy. Artificial Intelligence 119, 191–233 (2000)
52. Stone, P., Sridharan, M., Stronger, D., Kuhlmann, G., Kohl, N., Fidelman, P., Jong, N.K.: From pixels to multi-robot decision-making: A study in uncertainty. Robotics and Autonomous Systems 54, 933–943 (2006)
53. Guarino, N.: Formal ontology and information systems. In: Proceedings of the First International Conference on Formal Ontology in Information Systems (FOIS 1998), Trento, Italy, June 6-8, pp. 3–15. IOS Press, Amsterdam (1998)
54. Jarrar, M.: Towards Methodological Principles for Ontology Engineering. Ph.D thesis, Vrije Universiteit Brussel (2005)
55. Guyon, I., Gunn, S., Nikravesh, M., Zadeh, L.A. (eds.): Feature Extraction Foundations and Applications. Studies in Fuzziness and Soft Computing, vol. 207. Springer-Verlag, Berlin (2006)
56. Liu, H., Motoda, H. (eds.): Feature Extraction, Construction and Selection: A Data Mining Perspective. The Springer International Series in Engineering and Computer Science, vol. 453. Springer, Berlin (1998)
57. Liu, H., Motoda, H. (eds.): Feature Selection for Knowledge Discovery and Data Mining. The Springer International Series in Engineering and Computer Science, vol. 454. Springer, Berlin (1998)
58. Dejong, G., Mooney, R.: Explanation-based learning: An alternative view. Machine Learning 1, 145–176 (1986)
59. Ellman, T.: Explanation-based learning: a survey of programs and perspectives. ACM Computing Surveys 21, 163–221 (1989)
60. Mitchell, T.M., Keller, R.M., Kedar-Cabelli, S.T.: Explanation-based generalization: A unifying view. Machine Learning 1, 47–80 (1986)
61. Mitchell, T.M., Thrun, S.B.: Learning analytically and inductively. In: Steier, D.M., Mitchell, T.M. (eds.) Mind Matters: A Tribute to Allen Newell, pp. 85–110. Lawrence Erlbaum Associates, Inc., Mahwah (1996)
62. Penczek, W., Pólrola, A.: Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach. Studies in Computational Intelligence. Springer, Secaucus (2006)
63. Roddick, J.F., Hornsby, K., Spiliopoulou, M.: An updated bibliography of temporal, spatial and spatio-temporal data mining research. In: Roddick, J.F., Hornsby, K. (eds.) TSDM 2000. LNCS, vol. 2007, pp. 147–163. Springer, Heidelberg (2001)

64. Roddick, J.F., Hornsby, K., Spiliopoulou, M.: Yabtsstdmr - yet another bibliography of temporal, spatial and spatio-temporal data mining research. In: Unnikrishnan, K.P., Uthurusamy, R. (eds.) SIGKDD Temporal Data Mining Workshop. ACM Press, pp. 167–175. Springer, San Francisco (2001)
65. Ichise, R., Shapiro, D., Langley, P.: Learning hierarchical skills from observation. In: Lange, S., Satoh, K., Smith, C.H. (eds.) DS 2002. LNCS, vol. 2534, pp. 247–258. Springer, Heidelberg (2002)
66. Makar, R., Mahadevan, S., Ghavamzadeh, M.: Hierarchical multi-agent reinforcement learning. In: Müller, J.P., Andre, E., Sen, S., Frasson, C. (eds.) Proceedings of the Fifth International Conference on Autonomous Agents, Montreal, Canada, pp. 246–253. ACM Press, New York (2001)
67. Paine, R.W., Tani, J.: Motor primitive and sequence self-organization in a hierarchical recurrent neural network. Neural Networks 17, 1291–1309 (2004)
68. Zhang, L., Zhang, B.: Hierarchical machine learning - a learning methodology inspired by human intelligence. In: Wang, G.-Y., Peters, J.F., Skowron, A., Yao, Y. (eds.) RSKT 2006. LNCS, vol. 4062, pp. 28–30. Springer, Heidelberg (2006)
69. Paine, R.W., Tani, J.: How hierarchical control self-organizes in artificial adaptive systems. Adaptive Behavior 13, 211–225 (2005)
70. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Elsevier, Morgan Kaufmann, CA (2004)
71. Haslum, P.: Modern AI planning: Reading list, `http://www.ida.liu.se/~pahas/maip/reading.ps`
72. ICAPS 2006: Proceedings of the Sixteenth International Conference on Automated Planning & Scheduling, The English Lake District, Cumbria, UK, June 6-10. AAAI Press, Menlo Park (2006)
73. LaValle, S.M.: Planning Algorithms. Cambridge University Press, New York (2006)
74. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice Hall, New Jersey (2003)
75. Vlahavas, I., Vrakas, D. (eds.): Intelligent Techniques for Planning. Idea Group Publishing, New York (2004)
76. Wezel, W.V., Jorna, R., Meystel, A.: Planning in Intelligent Systems: Aspects, Motivations, and Methods. John Wiley & Sons, Hoboken, New Jersey (2006)
77. Bazan, J.G., Skowron, A., Peters, J.F., Synak, P.: Spatio-temporal approximate reasoning over complex objects. Fundamenta Informaticae 67, 249–269 (2005)
78. Bazan, J.G., Skowron, A., Stepaniuk, J.: Modelling complex patterns by information systems. Fundamenta Informaticae 67, 203–217 (2005)
79. Marcus, S.: The paradox of the heap of grains, in respect to roughness, fuzziness and negligibility. In: Polkowski, L., Skowron, A. (eds.) RSCTC 1998. LNCS, vol. 1424, pp. 19–23. Springer, Heidelberg (1998)
80. Nguyen, S.H., Nguyen, H.S.: Rough set approach to approximation of concepts from taxonomy. In: Proceedings of Knowledge Discovery and Ontologies Workshop (KDO 2004) at ECML/PKDD 2004, Pisa, Italy, University of Pisa, September 24, pp. 13–24 (2004)
81. Nguyen, S.H., Nguyen, T.T., Nguyen, H.S.: Rough set approach to sunspot classification problem. In: Ślęzak, D., et al. (eds.) RSFDGrC 2005. LNCS, vol. 3642, pp. 263–272. Springer, Heidelberg (2005)
82. Orłowska, E.: Semantics of vague concepts. In: Dorn, G., Weingartner, P. (eds.) Foundation of Logic and Linguistics, pp. 465–482. Plenum Press, New York (1984)
83. Skowron, A.: Rough sets and vague concepts. Fundamenta Informaticae 64, 417–431 (2005)

84. Skowron, A., Stepaniuk, J.: Hierarchical modelling in searching for complex patterns: Constrained sums of information systems. Journal of Experimental and Theoretical AI 17, 83–102 (2005)
85. Skowron, A., Synak, P.: Complex patterns in spatio-temporal reasoning. In: Czaja, L. (ed.) Proceedings Workshop on Concurrency, Specification, and Programming (CS&P 2003), Czarna, Poland, September 25-27, vol. 2, pp. 487–499 (2003)
86. Skowron, A., Synak, P.: Complex patterns. Fundamenta Informaticae 60, 351–366 (2004)
87. Zadeh, L.A.: Selected papers. In: Yager, R.R., Ovchinnokov, S., Tong, R., Nguyen, H. (eds.) Fuzzy Sets and Applications: Selected Papers by L.A. Zadeh. John Wiley & Sons, New York (1987)
88. Bazan, J.G.: Rough sets and granular computing in behavioral pattern identification and planning. In: Pedrycz, W., Skowron, A., Kreinovich, V. (eds.) Handbook of Granular Computing, pp. 777–799. John Wiley & Sons, Chichester (2008)
89. Doherty, P., Łukaszewicz, W., Skowron, A., Szałas, A.: Knowledge Engineering: A Rough Set Approach. Springer, Heidelberg (2006)
90. Nguyen, H.S., Skowron, A., Stepaniuk, J.: Granular computing: A rough set approach. Computational Intelligence 17, 514–544 (2001)
91. Skowron, A.: Towards granular multi-agent systems. In: Pal, S.K., Ghosh, A. (eds.) Soft Computing Approach to Pattern Recognition and Image Processing, pp. 215–234. World Scientific, Singapore (2002)
92. Bar-Yam, Y.: Dynamics of Complex Systems. Addison Wesley, New York (1997)
93. Desai, A.: Adaptive complex enterprises. Communications ACM 5, 32–35 (2005)
94. Gell-Mann, M.: The Quark and the Jaguar. Freeman and Co., New York (1994)
95. Luck, M., McBurney, P., Shehory, O., Willmott, S.: Agent technology: Computing as interaction. A roadmap for agent-based computing. Agentlink iii, the european coordination action for agent-based computing, University of Southampton, UK (2005)
96. Sun, R.: Cognition and Multi-Agent Interaction. From Cognitive Modeling to Social Simulation. Cambridge University Press, New York (2006)
97. Urmson, C., et al.: High speed navigation of unrehearsed terrain: Red team technology for grand challenge. Report CMU-RI-TR-04-37, The Robotics Institute, Carnegie Mellon University (2004)
98. Hoen, P.J., Tuyls, K., Panait, L., Luke, S., Poutré, J.A.L.: Overview of cooperative and competitive multiagent learning. In: Tuyls, K., Hoen, P.J., Verbeeck, K., Luke, S. (eds.) LAMAS 2005. LNCS, vol. 3898, pp. 1–46. Springer, Heidelberg (2006)
99. Liu, J.: Autonomous Agents and Multi-Agent Systems: Explorations in Learning, Self-Organization and Adaptive Computation. World Scientific Publishing, Singapore (2001)
100. Liu, J., Jin, X., Tsui, K.: Autonomy Oriented Computing: From Problem Solving to Complex Systems Modeling. Kluwer/Springer, Heidelberg (2005)
101. Luck, M., McBurney, P., Preist, C.: Agent technology: Enabling next generation. a roadmap for agent based computing. Agentlink, University of Southampton, UK (2003)
102. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm intelligence. From natural to artificial systems. In: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI 1999). Oxford University Press, UK (1999)
103. Dorigo, M., Birattari, M., Blum, C., Gambardella, L.M., Monada, F., Stutzle, T. (eds.): ANTS 2004. LNCS, vol. 3172. Springer, Heidelberg (2004)

104. Peters, J.F.: Rough ethology: Towards a biologically-inspired study of collective behavior in intelligent systems with approximation spaces. In: Peters, J.F., Skowron, A. (eds.) Transactions on Rough Sets III. LNCS, vol. 3400, pp. 153–174. Springer, Heidelberg (2005)
105. Gruber, T.R.: A translation approach to portable ontologies. Knowledge Acquisition 5, 199–220 (1993)
106. Smith, B.: Ontology and information systems. Technical report, The Buffalo Ontology Site (2001), http://ontology.buffalo.edu/ontologyPIC.pdf
107. W3C: OWL Web Ontology Language, use cases and requirements, W3C recommendation. Technical report, The World Wide Web Consortium Technical Report, http://www.w3.org/TR/2004/REC-webont-req-20040210/ (2004)
108. Bradbrook, K., Winstanley, G., Glasspool, D., Fox, J., Griffiths, R.: AI planning technology as a component of computerised clinical practice guidelines. In: Miksch, S., Hunter, J., Keravnou, E.T. (eds.) AIME 2005. LNCS, vol. 3581. Springer, Heidelberg (2005)
109. Dojat, M.: Systemes cognitifs pur le traitement d'informations clinques ou biologiques. Habilitation Thesis. L'universite Joseph Fourier, Grenoble, France (1999) (in French)
110. Dojat, M., Pachet, F., Guessoum, Z., Touchard, D., Harf, A., Brochard, L.: Neoganesh: A working system for the automated control of assisted ventilation in icus. Arificial Intelligence in Medicine 11, 97–117 (1997)
111. Glasspool, D., Fox, J., Castillo, F.D., Monaghan, V.E.L.: Interactive decision support for medical planning. In: Dojat, M., Keravnou, E.T., Barahona, P. (eds.) AIME 2003. LNCS, vol. 2780, pp. 335–339. Springer, Heidelberg (2003)
112. Miksch, S.: Plan management in the medical domain. AI Communications 12, 209–235 (1999)
113. Miller, R.: Medical diagnostic decision support systems - past, present, and future. Journal of the American Medical Informatics Association 1, 8–27 (1994)
114. Nilsson, M., Funk, P., Olsson, E.M.G., von Scheele, B., Xiong, N.: Clinical decision-support for diagnosing stress-related disorders by applying psychophysiological medical knowledge to an instance-based learning system. Arificial Intelligence in Medicine 36, 159–176 (2006)
115. OpenClinical, http://www.openclinical.org
116. Peek, N.: Decision-theoretic reasoning in medicine: bringing out the diagnosis. In: Proceedings of the Thirteenth Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2001), pp. 203–210 (2001)
117. Sacchi, L., Verduijn, M., Peek, N., de Jonge, E., de Mol, B., Bellazzi, R.: Describing and modeling time series based on qualitative temporal abstraction. In: Peek, N., Combi, C. (eds.) Working notes of the workshop on Intelligent Data Analysis in bioMedicine and Pharmacology (IDAMAP 2006), pp. 31–36 (2006)
118. Shahar, Y., Combi, C.: Temporal reasoning and temporal data maintenance: Issues and challenges. Computers in Biology and Medicine 27, 353–368 (2005)
119. Spyropoulos, C.D.: AI planning and scheduling in the medical hospital environment. Arificial Intelligence in Medicine 20, 101–111 (2000)
120. Stacey, M., McGregor, C.: Temporal abstraction in intelligent clinical data analysis: A survey. Arificial Intelligence in Medicine 39, 1–24 (2007)
121. Verduijn, M., Sacchi, L., Peek, N., Bellazzi, R., de Jonge, E., de Mol, B.: Temporal abstraction for feature extraction: A comparative case study in prediction from intensive care monitoring data. Arificial Intelligence in Medicine 41, 1–12 (2007)
122. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)

123. Peters, J.F.: Approximation spaces in off-policy monte carlo learning. In: Burczynski, T., Cholewa, W., Moczulski, W. (eds.) Recent Methods in Artificial Intelligence Methods. AI-METH Series, Gliwice, Poland, pp. 139–144 (2005)
124. Peters, J.F., Henry, C.: Reinforcement learning with approximation spaces. Fundamenta Informaticae 71, 323–349 (2006)
125. Kleinberg, J., Papadimitriou, C., Raghavan, P.: A microeconomic view of data mining. Data Mining and Knowledge Discovery 2, 311–324 (1998)
126. Keefe, R.: Theories of Vagueness. Cambridge University Press, New York (2000)
127. Keefe, R., Smith, P.: Vagueness: A Reader. MIT Press, Massachusetts (1997)
128. Read, S.: Thinking about Logic: An Introduction to the Philosophy of Logic. Oxford University Press, New York (1994)
129. Zadeh, L.: The concept of linguistic variable and its application to approximate reasoning-I. Information Sciences 8, 199–249 (1975)
130. Zadeh, L.: The concept of linguistic variable and its application to approximate reasoning-II. Information Sciences 8, 301–357 (1975)
131. Zadeh, L.: The concept of linguistic variable and its application to approximate reasoning-III. Information Sciences 9, 43–80 (1975)
132. Dynamics, N.: Special issue on modelling and control of intelligent transportation systems (9 articles). Journal Nonlinear Dynamics 49 (2006)
133. Peters, J.F.: Approximation spaces for hierarchical intelligent behavioral system models. In: Dunin-Keplicz, B., Jankowski, A., Skowron, A., Szczuka, M. (eds.) Monitoring, Security, and Rescue Techniques in Multiagent Systems. Advances in Soft Computing, pp. 13–30. Springer, Heidelberg (2005)
134. Peters, J.F., Henry, C., Ramanna, S.: Rough ethograms: Study of intelligent system behavior. In: Klopotek, M.A., Wierzchon, S.T., Trojanowski, K. (eds.) Proceedings of the International Conference on Intelligent Information Processing and Web Mining (IIS 2005), Gdańsk, Poland, June 13-16, pp. 117–126 (2005)
135. Birattari, M., Caro, G.D., Dorigo, M.: Toward the formal foundation of ant programming. In: Dorigo, M., Di Caro, G.A., Sampels, M. (eds.) Ant Algorithms 2002. LNCS, vol. 2463, pp. 188–201. Springer, Heidelberg (2002)
136. Sukthankar, G., Sycara, K.: A cost minimization approach to human behavior recognition. In: Proceedings of Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (2005)
137. Sukthankar, G., Sycara, K.: Automatic recognition of human team behaviors. In: Proceedings of Modeling Others from Observations (MOO), Workshop at the International Joint Conference on Artificial Intelligence (IJCAI 2005) (July 2005)
138. Adam, N.R., Janeja, V.P., Atluri, V.: Neighborhood based detection of anomalies in high dimensional spatio-temporal sensor datasets. In: Proceedings of the, ACM symposium on Applied computing (SAC 2004), pp. 576–583. ACM Press, New York (2004)
139. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion detection using sequence of system calls. Journal of Computer Security 2, 151–180 (1998)
140. Lazarevic, A., Ertoz, L., Kumar, V., Ozgur, A., Srivastava, J.: A comparative study of anomaly detection schemes in network intrusion detection. In: Proceedings of the Third SIAM International Conference on Data Mining, San Francisco, CA, USA, May 1-3. SIAM, Philadelphia (2003)
141. Li, X., Han, J., Kim, S., Gonzalez, H.: Roam: Rule- and motif-based anomaly detection in massive moving object data sets. In: Proceedings of the Seventh SIAM International Conference on Data Mining, Minneapolis, Minnesota, USA, April 26-28. SIAM, Philadelphia (2007)

142. Mahoney, M., Chan, P.K.: Learning rules for anomaly detection of hostile network traffic. In: Proceedings of the Third IEEE International Conference on Data Mining (ICDM 2003), Melbourne, Florida, USA, December 19-22, pp. 601–604. IEEE Computer Society, Los Alamitos (2003)
143. Sekar, R., Bendre, M., Dhurjati, D., Bollineni, P.: A fast automaton-based method for detecting anomalous program behaviors. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy (SP 2001), Washington, DC, USA, pp. 144–155. IEEE Computer Society Press, Los Alamitos (2001)
144. Shavlik, J.W., Shavlik, M.: Selection, combination, and evaluation of effective software sensors for detecting abnormal computer usage. In: Kim, W., Kohavi, R., Gehrke, J., DuMouchel, W. (eds.) Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, WA, USA, August 22-25, pp. 276–285 (2004)
145. Flach, P.A., Lachiche, N.: Naive bayesian classification of structured data. Machine Learning 57, 233–269 (2004)
146. Grandidier, F., Sabourin, R., Suen, C.Y.: Integration of contextual information in handwriting recognition systems. In: Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR 2003), Washington, DC, USA, p. 1252. IEEE Computer Society Press, Los Alamitos (2003)
147. Guo, G., Li, S., Chan, K.: Face recognition by support vector machines. In: Proceedings of the International Conferences on Automatic Face and Gesture Recognition, pp. 196–201 (2000)
148. Li, Z.C., Suen, C.Y., Guo, J.: Analysis and recognition of alphanumeric handprints by parts. In: Proceedings of the ICPR 1992, pp. 338–341 (1992)
149. Li, Z.C., Suen, C.Y., Guo, J.: A regional decomposition method for recognizing handprinted characters. SMC 25, 998–1010 (1995)
150. Nguyen, T.T.: Understanding domain knowledge: Concept approximation using rough mereology. In: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2005), Washington, DC, USA, pp. 217–222. IEEE Computer Society, Los Alamitos (2005)
151. Pavlidis, T.: Structural Pattern Recognition. Springer, Heidelberg (1980)
152. Yousfi, K., Ambroise, C., Cocquerez, J.P., Chevelu, J.: Supervised learning for guiding hierarchy construction: Application to osteo-articular medical images database. In: Proceedings of the 18th International Conference on Pattern Recognition (ICPR 2006), Washington, DC, USA, pp. 484–487. IEEE Computer Society Press, Los Alamitos (2006)
153. WITAS, `http://www.ida.liu.se/ext/witas/eng.html`
154. Hoare, C.A.R.: Process algebra: A unifying approach. In: Abdallah, A.E., Jones, C.B., Sanders, J.W. (eds.) Communicating Sequential Processes. LNCS, vol. 3525. Springer, Heidelberg (2005)
155. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Springer, New York (2006)
156. Roscoe, A.W., Hoare, C.A.R., Bird, R.: The Theory and Practice of Concurrency. Prentice Hall, New York (1997)
157. Suraj, Z.: Tools for generating concurrent models specified by information systems. In: Lin, T.Y., Wildberger, A.M. (eds.) Soft Computing: Rough Sets, Fuzzy Logic, Neural Networks, Uncertainty Management, Knowledge Discovery, pp. 107–110. Society For Computer Simulation, San Diego (1995)
158. Taubenfeld, G.: Synchronization Algorithms and Concurrent Programming. Prentice-Hall, New York (2006)

159. Winskel, G., Nielsen, M.: Models for concurrency. In: Handbook of logic in computer science: semantic modelling, vol. 4, pp. 1–148. Oxford University Press, Oxford (1995)
160. Kauppinen, T., Hyvonen, E.: Modeling and reasoning about changes in ontology time series. In: Kishore, R., Ramesh, R., Sharman, R. (eds.) Ontologies: A Handbook of Principles, Concepts and Applications in Information Systems, pp. 319–338. Springer, New York (2007)
161. Voigt, K.: Reasoning about changes and uncertainty in browser customization. In: Proceedings of the AAAI Fall Symposium, Working Notes of AI Applications to Knowledge Navigation and Retrieval (1995)
162. Bar-Hillel, A., Hertz, T., Shental, N., Weinshall, D.: Learning a mahalanobis metric from equivalence constraints. Journal of Machine Learning Research 6, 937–965 (2005)
163. Dzeroski, S., Lavrac, N. (eds.): Relational Data Mining. Springer, New York (2001)
164. Gartner, T.: A survey of kernels for structured data. SIGKDD Explorations 5, 48–58 (2003)
165. Nguyen, S.H.: Regularity Analysis and Its Applications in Data Mining. PhD thesis, Warsaw University, Warsaw, Poland (2000)
166. Shalev-Shwartz, S., Singer, Y., Ng, A.Y.: Online and batch learning of pseudometrics. In: Proceedings of the twenty-first international conference on Machine learning (ICML 2004), vol. 4585. ACM Press, New York (2004)
167. Weinberger, K., Blitzer, J., Saul, L. (eds.): Distance metric learning for large margin nearest neighbor classification. Advances in Neural Information Processing Systems, vol. 18. MIT Press, Cambridge (2006)
168. Wettschereck, D., Aha, D.W., Mohri, T.: A review and empirical evaluation of feature weighting methods for aclass of lazy learning algorithms. Artifficial Intelligence Review 11, 273–314 (1997)
169. Wojna, A.: Analogy-based Reasoning in Classifier Construction. PhD thesis, Warsaw University; Faculty of Mathematics, Informatics and Mechanics, Warsaw, Poland (2004); Defense in 2005. Awarded with honours
170. Xing, E.P., Ng, A.Y., Jordan, M.I., Russell, S. (eds.): Distance metric learning with application to clustering with side-information. Advances in NIPS, vol. 15. MIT Press, Cambridge (2003)
171. Yang, L., Jin, R., Sukthankar, R., Liu, Y.: An efficient algorithm for local distance metric learning. In: Proceedings of the Twenty-first National Conference on Artificial Intelligence (AAAI 2006). ACM Press, New York (2006)
172. Bazan, J.G., Skowron, A.: Classifiers based on approximate reasoning schemes. In: Dunin-Kęplicz, B., Jankowski, A., Skowron, A., Szczuka, M. (eds.) Monitoring, Security, and Rescue Techniques in Multiagent Systems. Advances in Soft Computing, pp. 191–202. Springer, Heidelberg (2005)
173. Bazan, J.G.: Behavioral pattern identification through rough set modeling. Fundamenta Informaticae 72, 37–50 (2006)
174. Bazan, J.G., Kruczek, P., Bazan-Socha, S., Skowron, A., Pietrzyk, J.: Risk pattern identification in the treatment of infants with respiratory failure through rough set modeling. In: Proceedings of the Eleventh Conference of Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2006), July 2-7, Paris, France, pp. 2650–2657 (2006)
175. Bazan, J.G., Kruczek, P., Bazan-Socha, S., Skowron, A., Pietrzyk, J.: Rough set approach to behavioral pattern identification. Fundamenta Informaticae 75, 27–47 (2007)

176. Bazan, J.G., Peters, J.F., Skowron, A.: Behavioral pattern identification through rough set modelling. In: Ślęzak, D., et al. (eds.) RSFDGrC 2005. LNCS, vol. 3642, pp. 688–697. Springer, Heidelberg (2005)
177. Bazan, J.G., Skowron, A.: On-line elimination of non-relevant parts of complex objects in behavioral pattern identification. In: Pal, S.K., et al. (eds.) PReMI 2005. LNCS, vol. 3776, pp. 720–725. Springer, Heidelberg (2005)
178. Bazan, J.G., Skowron, A.: Perception based rough set tools in behavioral pattern identification. In: Czaja, L. (ed.) Proceedings Workshop on Concurrency, Specification, and Programming (CS&P 2005), September 28-30, pp. 50–56. Warsaw University, Warsaw (2005)
179. Nguyen, S.H., Bazan, J., Skowron, A., Nguyen, H.S.: Layered learning for concept synthesis. LNCS Transactions on Rough Sets 3100, 187–208 (2004)
180. Bazan, J.G., Kruczek, P., Bazan-Socha, S., Skowron, A., Pietrzyk, J.: Automatic planning based on rough set tools: Towards supporting treatment of infants with respiratory failure. In: Proceedings of the Workshop on Concurrency, Specification, and Programming (CS&P 2006), Wandlitz, Germany, September 27-29. Informatik-Bericht, vol. 170, pp. 388–399. Humboldt University, Berlin (2006)
181. Bazan, J.G., Kruczek, P., Bazan-Socha, S., Skowron, A., Pietrzyk, J.: Automatic planning of treatment of infants with respiratory failure through rough set modeling. In: Greco, S., Hata, Y., Hirano, S., Inuiguchi, M., Miyamoto, S., Nguyen, H.S., Słowiński, R. (eds.) RSCTC 2006. LNCS, vol. 4259, pp. 418–427. Springer, Heidelberg (2006)
182. Stone, P.: Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer. The MIT Press, Cambridge (2000)
183. Bolc, L., Szałas, A. (eds.): Time and Logic: A Computational Approach. UCL Press, London (1995)
184. Clark, E., Emerson, E., Sistla, A.: Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. ACM Transactions on Programming Languages and Systems 8, 244–263 (1986)
185. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science: Volume B: Formal Models and Semantics, pp. 995–1072. Elsevier, Amsterdam (1990)
186. Skowron, A., Stepaniuk, J.: Constrained sums of information systems. In: Tsumoto, S., Słowiński, R., Komorowski, J., Grzymała-Busse, J.W. (eds.) RSCTC 2004. LNCS, vol. 3066, pp. 300–309. Springer, Heidelberg (2004)
187. Skowron, A., Stepaniuk, J.: Ontological framework for approximation. In: Ślęzak, D., et al. (eds.) RSFDGrC 2005. LNCS, vol. 3641, pp. 718–727. Springer, Heidelberg (2005)
188. Komorowski, J., Polkowski, L., Skowron, A.: Towards a rough mereology-based logic for approximate solution synthesis. Special Issue on Logics with Incomplete Information, Studia Logica 58, 143–184 (1997)
189. Komorowski, J., Polkowski, L., Skowron, A.: Rough sets: a tutorial. In: Pal, S.K., Skowron, A. (eds.) Rough Fuzzy Hybridization: A New Trend in Decision-Making, pp. 3–98. Springer, Singapore (1999)
190. Polkowski, L., Skowron, A.: Rough mereology. In: Raś, Z.W., Zemankova, M. (eds.) ISMIS 1994. LNCS, vol. 869, pp. 85–94. Springer, Heidelberg (1994)
191. Polkowski, L., Skowron, A.: Rough mereological approach to knowledge-based distributed ai. In: Lee, J.K., Liebowitz, J., Chae, Y.M. (eds.) Proceedings of the Third World Congress on Expert Systems, February 5-9, Soeul, Korea, pp. 774–781. Cognizant Communication Corporation (1996)

192. Polkowski, L., Skowron, A.: Rough mereology: A new paradigm for approximate reasoning. International Journal of Approximate Reasoning 15, 333–365 (1996)
193. Polkowski, L., Skowron, A.: Rough mereology in information systems. a case study: Qualitative spatial reasoning. In: Polkowski, L., Lin, T.Y., Tsumoto, S. (eds.) Rough Set Methods and Applications: New Developments in Knowledge Discovery in Information Systems. Studies in Fuzziness and Soft Computing, vol. 56. Springer, Heidelberg (2000)
194. Polkowski, L., Skowron, A.: Rough mereological calculi of granules: A rough set approach to computation. Computational Intelligence 17, 472–492 (2001)
195. Bazan, J.G.: Dynamic reducts and statistical inference. In: Proceedings of the Sixth International Conference on Information Processing and Management of Uncertainty on Knowledge Based Systems (IPMU 1996), Granada, Spain, July 1-5, vol. III, pp. 1147–1152 (1996)
196. Bazan, J.G.: A comparison of dynamic and non-dynamic rough set methods for extracting laws from decision tables. In: Polkowski, L., Skowron, A. (eds.) Rough Sets in Knowledge Discovery 1: Methodology and Applications. Studies in Fuzziness and Soft Computing, vol. 18, pp. 321–365. Physica-Verlag, Heidelberg (1998)
197. Bazan, J.G.: Discovery of decision rules by matching new objects against data tables. In: Polkowski, L., Skowron, A. (eds.) RSCTC 1998. LNCS (LNAI), vol. 1424, pp. 521–528. Springer, Heidelberg (1998)
198. Bazan, J.G.: Methods of approximate reasoning for synthesis of decision algorithms. PhD thesis, Warsaw University, Warsaw, Poland (1999) (in Polish)
199. Bazan, J.G., Nguyen, H.S., Nguyen, S.H., Synak, P., Wróblewski, J.: Rough set algorithms in classification problems. In: Polkowski, L., Lin, T.Y., Tsumoto, S. (eds.) Rough Set Methods and Applications: New Developments in Knowledge Discovery in Information Systems. Studies in Fuzziness and Soft Computing, vol. 56, pp. 49–88. Springer, Heidelberg (2000)
200. Bazan, J.G., Nguyen, H.S., Szczuka, M.: A view on rough set concept approximations. Fundamenta Informaticae 59, 107–118 (2004)
201. Bazan, J.G., Skowron, A., Synak, P.: Discovery of decision rules from experimental data. In: Lin, T.Y. (ed.) Proceedings of the Third International Workshop on Rough Sets and Soft Computing (RSSC 1994), San Jose, CA, USA, pp. 526–533 (1994)
202. Bazan, J.G., Skowron, A., Synak, P.: Dynamic reducts as a tool for extracting laws from decision tables. In: Raś, Z.W., Zemankova, M. (eds.) ISMIS 1994. LNCS (LNAI), vol. 869, pp. 346–355. Springer, Heidelberg (1994)
203. Bazan, J.G., Skowron, A., Synak, P.: Market data analysis: A rough set approach. ICS Research Reports 6, Warsaw University of Technology, Warsaw, Poland (1994)
204. Barwise, J., Seligman, J.: The logic of distributed systems. Cambridge University Press, Cambridge (1997)
205. Fahlman, S.E., Lebiere, C.: The Cascade-Correlation Learning Architecture. Advances in Neural Information Processing Systems, vol. II. Morgan Kaufmann, San Mateo (1990)
206. Cedrowska, J.: Prism: An algorithm for inducing modular rules. In: Gaines, B.R., Boose, J.H. (eds.) Knowledge Acquisition for Knowledge-based Systems. Academic Press, New York (1988)
207. Cestnik, B., Kononenko, I., Bratko, I.: Assistant 86: A knowledge elicitation tool for sophisticated users. In: Proceedings of EWSL 1987, Bled, Yugoslavia, pp. 31–47 (1997)

208. Goodman, R.M., Smyth, P.: The induction of probabilistic rule sets – the algorithm. In: Proceedings of the Sixth International Workshop on Machine Learning, San Mateo, CA, pp. 129–132. Morgan Kaufmann, San Francisco (1989)
209. Quinlan, J.R.: The cascade-correlation learning architecture. In: Machine Learning 1, pp. 81–106. Kluwer Academic, Boston (1990)
210. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo (1993)
211. Utgoff, P.E.: Incremental learning of decision trees. In: Machine Learning 1, vol. IV, pp. 161–186. Kluwer Academic, Boston (1990)
212. Velde, W.V.D.: Idl, or taming the multiplexer. In: Proceedings of the Fourth European Working Session on Learning, Pitman, London, pp. 31–47 (1989)
213. Bloedorn, E., Michalski, R.S.: The multi-purpose incremental learning system AQ15 and its testing to three medical domains. In: Proceedings of the Third International Conference on Tools for AI, San Jose, CA (1991)
214. Clark, P., Niblett, T.: The CN2 induction algorithm. In: Machine Learning 3, pp. 261–284. Kluwer Academic, Boston (1989)
215. Grzymała-Busse, J.W.: LERS - a system for learning from examples based on rough sets. In: Słowiński, R. (ed.) Intelligent Decision Support - Handbook of Applications and Advances of the Rough Sets Theory. D: System Theory, Knowledge Engineering and Problem Solving, vol. 11, pp. 3–18. Kluwer Academic Publishers, Dordrecht (1992)
216. Grzymała-Busse, J.W.: LERS - a knowledge discovery system. In: Polkowski, L., Skowron, A. (eds.) Rough Sets in Knowledge Discovery 2. Applications, Case Studies and Software Systems. Studies in Fuzziness and Soft Computing, pp. 562–565. Physica-Verlag, Heidelberg (1998)
217. Michalski, R.S., Mozetic, I., Hong, J., Lavrac, N.: The multi-purpose incremental learning system AQ15 and its testing to three medical domains. In: Proceedings of the AAAI 1986, San Mateo, CA, pp. 1041–1045. Morgan Kaufmann, San Francisco (1986)
218. Michalski, R.S., Wnęk, J.: Constructive induction: An automated improvement of knowledge representation spaces for machine learning. In: Proceedings of a Workshop on Intelligent Information Systems, Practical Aspect of AI II, Augustów, Poland, pp. 188–236. Morgan Kaufmann, San Francisco (1993)
219. Mienko, R., Słowiński, R., Stefanowski, J., Susmaga, R.: Roughfamily - software implementation of rough set based data analysis and rule discovery techniques. In: Tsumoto, S. (ed.) Proceedings of the Fourth International Workshop on Rough Sets, Fuzzy Sets and Machine Discovery, Tokyo, Japan, November 6-8, pp. 437–440 (1996)
220. Øhrn, A., Komorowski, J.: ROSETTA - a rough set tool kit for analysis of data. In: Tsumoto, S. (ed.) Proceedings of the Fifth International Workshop on Rough Sets and Soft Computing (RSSC 1997) at the Third Joint Conference on Information Sciences (JCIS 1997), Research Triangle Park, NC, USA, March 2-5, pp. 403–407 (1997)
221. Słowiński, R., Stefanowski, J.: 'RoughDAS' and 'RoughClass' software implementations of the rough set approach. In: Słowiński, R. (ed.) Intelligent Decision Support - Handbook of Applications and Advances of the Rough Sets Theory. D: System Theory, Knowledge Engineering and Problem Solving, vol. 11. Kluwer Academic Publishers, Dordrecht (1992)

222. Stefanowski, J.: On rough set based approaches to induction of decision rules. In: Polkowski, L., Skowron, A. (eds.) Rough Sets in Knowledge Discovery 1: Methodology and Applications. Studies in Fuzziness and Soft Computing, vol. 18, pp. 500–529. Physica-Verlag, Heidelberg (1998)
223. Stefanowski, J.: Algorithims of rule induction for knowledge discovery. Habilitation Thesis published as Series Rozprawy no. 361. Poznań University of Technology Press, Poznań (2001) (in Polish)
224. Dzeroski, S.: Handling noise in inductive logic programming. Master's thesis, Dept. of EE and CS, University of Ljubljana, Ljubljana, Slovenia (1991)
225. Bazan, J.G., Latkowski, R., Szczuka, M.: DIXER - distributed executor for rough set exploration system. In: Ślęzak, D., et al. (eds.) RSFDGrC 2005. LNCS, vol. 3642, pp. 39–47. Springer, Heidelberg (2005)
226. Bazan, J.G., Szczuka, M.: RSES and RSESlib – a collection of tools for rough set computations. In: Ziarko, W., Yao, Y. (eds.) RSCTC 2000. LNCS (LNAI), vol. 2005, pp. 106–113. Springer, Heidelberg (2001)
227. Bazan, J.G., Szczuka, M., Wojna, A., Wojnarski, M.: On the evolution of rough set exploration system. In: Tsumoto, S., Słowiński, R., Komorowski, J., Grzymała-Busse, J.W. (eds.) RSCTC 2004. LNCS, vol. 3066, pp. 592–601. Springer, Heidelberg (2004)
228. Bazan, J.G., Szczuka, M.S., Wróblewski, J.: A new version of rough set exploration system. In: Alpigini, J.J., Peters, J.F., Skowron, A., Zhong, N. (eds.) RSCTC 2002. LNCS, vol. 2475, pp. 397–404. Springer, Heidelberg (2002)
229. Synak, P.: Rough Set Expert System, User's Guide. Technical report, Warsaw University, Warsaw, Poland (1995)
230. Synak, P., Bazan, J.G., Cykier, A.: RSES Core Classes: The Technical Documentation. Technical report, Warsaw University, Warsaw, Poland (1996)
231. Szczuka, M.: Mikołajczyk, M., Bazan, J.G.: RSES 2.2 user's guide. Technical report, Warsaw University, Warsaw, Poland (2005)
232. Polkowski, L.: Rough Sets: Mathematical Foundations. Advances in Soft Computing. Springer-Verlag/Physica-Verlag, Heidelberg (2002)
233. Lipski, W.: On databases with incomplete information. Journal of the ACM 28, 41–70 (1981)
234. Pawlak, Z.: Classification of objects by means of attributes. Polish Academy of Sciences 429 (1981)
235. Orłowska, E., Pawlak, Z.: Representation of nondeterministic information. Theoretical Computer Science 29, 27–39 (1984)
236. Orłowska, E. (ed.): Incomplete Information: Rough Set Analysis. Studies in Fuzziness and Soft Computing, vol. 13. Physica-Verlag, Heidelberg (1998)
237. Nguyen, H.S.: Discretization of Real Value Attributes, Boolean Reasoning Approach. Ph.D thesis, Warsaw University, Warsaw, Poland (1997)
238. Nguyen, H.S.: Approximate boolean reasoning: Foundations and applications in data mining. In: Peters, J.F., Skowron, A. (eds.) Transactions on Rough Sets V. LNCS, vol. 4100, pp. 334–506. Springer, Heidelberg (2006)
239. Polkowski, L., Skowron, A.: Synthesis of decision systems from data tables. In: Lin, T.Y., Cecerone, N. (eds.) Rough Sets and Data Mining. Analysis for Imprecise Data. Advances in Soft Computing, pp. 259–299. Kluwer Academic, Dordrecht (1997)
240. Pawlak, Z., Skowron, A.: A rough set approach for decision rules generation. In: ICS Research Report 23/93, Warsaw University of Technology and Proceedings of the IJCAI 1993 Workshop W12: The Management of Uncertainty in AI, France (1993)

241. Tsumoto, S., Tanaka, H.: PRIMEROSE: Probabilistic rule induction method based on rough sets and resampling methods. Computational Intelligence 11, 389–405 (1995)
242. Synak, P.: Methods of approximate reasoning in searching for rough dependencies. Master's thesis, Warsaw University, Warsaw, Poland (1996) (in Polish)
243. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
244. Bennett, B.: The role of definitions in construction and analysis of formal ontologies. In: Sixth Symposium on Logical Formalizations of Commonsense Reasoning, Palo Alto, CA, USA (2003)
245. Tarski, A.: Some methodological investigations on the definability of concepts. In: Logic, Semantics, Metamathematics. Clarendon Press, Oxford (1956)
246. Bazan, J.G., Skowron, A., Świniarski, R.W.: Rough sets and vague concept approximation: From sample approximation to adaptive learning. In: Peters, J.F., Skowron, A. (eds.) Transactions on Rough Sets V. LNCS, vol. 4100, pp. 39–62. Springer, Heidelberg (2006)
247. Skowron, A., Stepaniuk, J., Peters, J.F., Świniarski, R.W.: Calculi of approximation spaces. Fundamenta Informaticae 72, 363–378 (2006)
248. Skowron, A., Świniarski, R.: Rough sets and higher order vagueness. In: Ślęzak, D., et al. (eds.) RSFDGrC 2005. LNCS, vol. 3641, pp. 33–42. Springer, Heidelberg (2005)
249. Skowron, A., Świniarski, R.W., Synak, P.: Approximation spaces and information granulation. In: Peters, J.F., Skowron, A. (eds.) Transactions on Rough Sets III. LNCS, vol. 3400, pp. 175–189. Springer, Heidelberg (2005)
250. Zadeh, L.: Fuzzy sets. Information and Control, 338–353 (1965)
251. Ajdukiewicz, K.: Pragmatic Logic. Reidel, Dordrecht (1974)
252. Holland, J.H., Holyoak, K.J., Nisbett, R.E., Thagard, P.R.: Induction: processes of inference, learning, and discovery. MIT Press, Cambridge (1989)
253. Skowron, A., Stepaniuk, J.: Tolerance approximation spaces. Fundamenta Informaticae 27, 245–253 (1996)
254. Pawlak, Z., Skowron, A.: Rough sets: Some extensions. Information Sciences 177, 28–40 (2007)
255. Stepaniuk, J.: Knowledge discovery by application of rough set models. In: Polkowski, L., Lin, T.Y., Tsumoto, S. (eds.) FCT 1977. Studies in Fuzziness and Soft Computing, vol. 56, pp. 137–233. Springer-Verlag/Physica-Verlag, Heidelberg (2000)
256. Ziarko, W.: Variable precision rough set model. Journal of Computer and System Sciences 46, 39–59 (1993)
257. Provost, F., Kohavi, R.: On applied research in machine learning. Machine Learning 30, 127–132 (1998)
258. Weiss, S.M., Kulikowski, C.A.: Computer Systems That Learn. Morgan Kaufmann, San Mateo (1991)
259. ROSETTA: Project web site, `http://rosetta.lcb.uu.se/general`
260. Altman, D.G.: Practical Statistics for Medical Research. Chapman and Hall/CRC, London (1997)
261. Fawcett, T.: An introduction to ROC analysis. Pattern Recognition Letters 27, 861–874 (2006)
262. Swets, J.A.: Measuring the accuracy of diagnostic systems. Science 240, 1285–1293 (1988)
263. Øhrn, A., Komorowski, J., Skowron, A., Synak, P.: The design and implementation of a knowledge discovery toolkit based on rough sets: The ROSETTA system. In: Polkowski, L., Skowron, A. (eds.) Rough Sets in Knowledge Discovery 1: Methodology and Applications. Studies in Fuzziness and Soft Computing, vol. 18, pp. 376–399. Physica-Verlag, Heidelberg (1998)

264. Øhrn, A., Komorowski, J., Skowron, A., Synak, P.: The ROSETTA software system. In: Polkowski, L., Skowron, A. (eds.) Rough Sets in Knowledge Discovery 2. Applications, Case Studies and Software Systems. Studies in Fuzziness and Soft Computing, pp. 572–576. Physica-Verlag, Heidelberg (1998)
265. Efron, B.: Estimating the error rate of a prediction rule: improvement on cross validation. Journal of American Statistics Association 78, 316–331 (1983)
266. Stefanowski, J.: Classification and decision supporting based on rough set theory. Foundations of Computing and Decision Sciences 18, 371–380 (1993)
267. Delimata, P., Moshkov, M., Skowron, A., Suraj, Z.: Two families of classification algorithms. In: An, A., Stefanowski, J., Ramanna, S., Butz, C.J., Pedrycz, W., Wang, G. (eds.) RSFDGrC 2007. LNCS, vol. 4482, pp. 297–304. Springer, Heidelberg (2007)
268. Bazan, J.G., Nguyen, H.S., Skowron, A., Szczuka, M.S.: A view on rough set concept approximations. In: Wang, G., Liu, Q., Yao, Y., Skowron, A. (eds.) RSFDGrC 2003. LNCS, vol. 2639, pp. 181–188. Springer, Heidelberg (2003)
269. Skowron, A., Peters, J.F.: Rough sets: Trends and challenges - plenary paper. In: Wang, G., Liu, Q., Yao, Y., Skowron, A. (eds.) RSFDGrC 2003. LNCS, vol. 2639, pp. 25–34. Springer, Heidelberg (2003)
270. Webster: Webster's New Collegiate Dictionary. Merriam-Webster, Springfield (1991)
271. Liliana, A.: Material and formal ontology. In: Roberto, P., Peter, S. (eds.) Formal ontology. Advanced in Soft Computing, pp. 199–232. Kluwer, Dordrecht (1996)
272. Guarino, N., Poli, R.: Formal ontology in conceptual analysis and knowledge representation. Kluwer, Dordrecht (1993)
273. Shapiro, S.C.: Encyclopedia of Artificial Intelligence. John Wiley and Sons, New York (1992)
274. Booch, G.: Object-oriented Analysis and Design with Applications. Addison-Wesley Publishing Company, Santa Clara (1994)
275. Taylor, D.A.: Object-Oriented Information Systems: Planning and Implementation. John Wiley & Sons, New York (1992)
276. Jones, D., Bench-Capon, T., Visser, P.: Ontology-based support for human disease study. In: Proceedings of the IT&KNOWS Conference, XV IFIP World Computer Congress (August 1998)
277. Uschold, M., Grüninger, M.: Ontologies: principles, methods, and applications. Knowledge Engineering Review 11, 93–155 (1996)
278. Uschold, M.: Building ontologies: Towards a unified methodology. In: Proceedings 16th Annual Conference of the British Computer Society Specialist Group on Expert Systems, Cambridge, UK (1996)
279. Dublin Core: Project web site, `http://dublincore.org/`
280. General Formal Ontology (GFO): Project web site, `http://www.onto-med.de/`
281. OpenCyc/ResearchCyc: Project web site, `http://research.cyc.com/`
282. Suggested Upper Merged Ontology (SUMO): Project web site, `http://www.articulatesoftware.com/`
283. WordNet: Project web site, `http://wordnet.princeton.edu/`
284. Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE): Project web site, `http://www.loa-cnr.it/DOLCE.html`
285. W3C: RDF Primer, W3C Recommendation. Technical report, The World Wide Web Consortium Technical Report (2004), `http://www.w3.org/RDF/`
286. OIL - Ontology Inference Layer: Project web site, `www.ontoknowledge.org/oil/`
287. DAML – DARPA Agent Markup Language: Project web site, `www.daml.org`

288. Cyc: Project web site, http://www.cyc.com
289. OpenCyc: Project web site, http://opencyc.org
290. Protege: Project web site, http://protege.stanford.edu
291. OntoStudio: Project web site, http://www.ontoprise.de
292. Ontolingua: Project web site, www.ksl.stanford.edu/software/ontolingua/
293. Chimaera: Project web site, http://ksl.stanford.edu/software/chimaera/
294. OilEd: Project web site, http://oiled.man.ac.uk/
295. W3C: RDQL - a query language for RDF, W3C member submission. Technical report, The World Wide Web Consortium Technical Report (2004), http://www.w3.org/Submission/RDQL
296. Fahle, M., Poggio, T.: Perceptual Learning. MIT Press, Cambridge (2002)
297. Harnad, S.: Categorical Perception: The Groundwork of Cognition. Cambridge University Press, New York (1987)
298. McCarthy, J.: Notes on formalizing context. In: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI 1993). Morgan Kaufmann, Chambéry (1993)
299. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. IV, pp. 463–502. Edinburgh University Press, Edinburgh (1969)
300. Anderson, J.R.: Rules of the mind. Lawrence Erlbaum, Hillsdale (1993)
301. Kieras, D., Meyer, D.E.: An overview of the epic architecture for cognition and performance with application to human-computer interaction. Human-Computer Interaction 12, 391–438 (1997)
302. Laird, J., Newell, A., Rosenbloom, P.: Soar: An architecture for general intelligence. Artificial Intelligence 33, 1–64 (1987)
303. Veloso, M.M., Carbonell, J.G.: Derivational analogy in prodigy: Automating case acquisition, storage, and utilization. Machine Learning 10, 249–278 (1993)
304. Tarski, A.: The semantic concept of truth. Philosophy and Phenomenological Research 4, 341–375 (1944)
305. Dictionary, T.F.: Project web site, http://www.thefreedictionary.com
306. Bazan, J.G., Nguyen, S.H., Nguyen, H.S., Skowron, A.: Rough set methods in approximation of hierarchical concepts. In: Tsumoto, S., Słowiński, R., Komorowski, J., Grzymała-Busse, J.W. (eds.) RSCTC 2004. LNCS, vol. 3066, pp. 346–355. Springer, Heidelberg (2004)
307. Nguyen, S.H., Nguyen, H.S.: Improving Rough Classifiers Using Concept Ontology. In: Ho, T.B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS, vol. 3518, pp. 312–322. Springer, Heidelberg (2005)
308. Greco, S., Matarazzo, B., Słowiński, R.: A new rough set approach to multicriteria and multiattribute classification. In: Polkowski, L., Skowron, A. (eds.) RSCTC 1998. LNCS (LNAI), vol. 1424, pp. 60–67. Springer, Heidelberg (1998)
309. Greco, S., Matarazzo, B., Słowiński, R.: Rough approximation of preference relation by dominance relations. ICS Research Report 16/1996, Warsaw University of Technology, Warsaw, Poland; also in Journal of Operational Research 117, 63–83 (1999)
310. Greco, S., Matarazzo, B., Słowiński, R.: Multicriteria classification. In: Kloesgen, W., Zytkow, J. (eds.) Handbook of Data Mining and Knowledge Discovery, pp. 318–328. Oxford University Press, Inc., New York (2002)
311. Greco, S., Matarazzo, B., Słowiński, R.: Rough approximation by dominance relations. International Journal of Intelligent Systems 17, 153–171 (2002)

312. Błaszczyński, J., Greco, S., Słowiński, R.: Multi-criteria classification – a new scheme for application of dominance-based decision rules. Journal of Operational Research 181, 1030–1044 (2007)
313. Błaszczyński, J., Słowiński, R.: Incremental induction of decision rules from dominance-based rough approximations. In: Skowron, A., Szczuka, M. (eds.) Electronic Notes in Theoretical Computer Science, vol. 82. Springer, Heidelberg (2003)
314. Błaszczyński, J., Słowiński, R.: Incremental induction of satisfactory decision rules from dominance based rough approximations. In: Skowron, A., Szczuka, M. (eds.) Proceedings of the International Workshop on Rough Sets in Knowledge Discovery and Soft Computing (RSKD 2003), Warsaw, Poland, April 12-13, pp. 40–51 (2003)
315. Greco, S., Matarazzo, B., Słowiński, R., Stefanowski, J.: An algorithm for induction of decision rules consistent with the dominance principle. In: Ziarko, W.P., Yao, Y. (eds.) RSCTC 2000. LNCS, vol. 2005, pp. 304–313. Springer, Heidelberg (2001)
316. Peters, J.F.: Time and clock information systems: Concepts and rough fuzzy petri net models. In: Polkowski, L., Skowron, A. (eds.) Rough Sets in Knowledge Discovery 2. Applications, Case Studies and Software Systems. Studies in Fuzziness and Soft Computing, pp. 385–417. Springer-Verlag, Berlin (1998)
317. Polkowski, L.: Granulation of knowledge in decision systems: The approach based on rough inclusions. The method and its applications. In: Kryszkiewicz, M., Peters, J.F., Rybinski, H., Skowron, A. (eds.) RSEISP 2007. LNCS, vol. 4585, pp. 69–79. Springer, Heidelberg (2007)
318. Synak, P.: Temporal Aspects of Data Analysis: A Rough Set Approach. Ph.D thesis, The Institute of Computer Science of the Polish Academy of Sciences, Warsaw, Poland (2003) (in Polish) (defended in 2004)
319. Polkowski, L., Artiemjew, P.: On granular rough computing with missing values. In: Kryszkiewicz, M., Peters, J.F., Rybinski, H., Skowron, A. (eds.) RSEISP 2007. LNCS, vol. 4585, pp. 271–279. Springer, Heidelberg (2007)
320. Skowron, A.: Toward intelligent systems: Calculi of information granules. In: Terano, T., Nishida, T., Namatame, A., Tsumoto, S., Ohsawa, Y., Washio, T. (eds.) JSAI-WS 2001. LNCS (LNAI), vol. 2253, pp. 251–260. Springer, Heidelberg (2001)
321. Stepaniuk, J.: Approximation spaces, reducts and representatives. In: Polkowski, L., Skowron, A. (eds.) Rough Sets in Knowledge Discovery 1: Methodology and Applications. Studies in Fuzziness and Soft Computing, vol. 18, pp. 109–126. Physica-Verlag, Heidelberg (1998)
322. Stepaniuk, J.: Knowledge discovery by application of rough set models. In: Polkowski, L., Lin, T.Y., Tsumoto, S. (eds.) Rough Set Methods and Applications: New Developments in Knowledge Discovery in Information Systems. Studies in Fuzziness and Soft Computing, vol. 56, pp. 137–233. Physica-Verlag, Heidelberg (2000)
323. Yao, Y.Y.: Perspectives of granular computing. In: Proceedings of the Conference on Granular Computing (GrC 2005), Beijing, China, New York. IEEE Press, Los Alamitos (2005)
324. Bazan, J.G., Osmólski, A., Skowron, A., Ślęzak, D., Szczuka, M., Wróblewski, J.: Rough set approach to survival analysis. In: Suraj, Z. (ed.) Proceedings of the Sixth International Conference on Soft Computing and Distributed Processing (SCDP 2002), June 24-25, pp. 45–48. University of Information Technology and Management in Rzeszów Press, Rzeszów (2002)

325. Bazan, J.G., Osmólski, A., Skowron, A., Ślęzak, D., Szczuka, M., Wróblewski, J.: Rough set approach to the survival analysis. In: Alpigini, J.J., Peters, J.F., Skowron, A., Zhong, N. (eds.) RSCTC 2002. LNCS, vol. 2475, pp. 522–529. Springer, Heidelberg (2002)
326. Bazan, J.G., Skowron, A., Ślęzak, D., Wróblewski, J.: Searching for the complex decision reducts: The case study of the survival analysis. In: Raś, Z.W., Zhong, N., Tsumoto, S., Suzuku, E. (eds.) ISMIS 2003. LNCS, vol. 2871, pp. 160–168. Springer, Heidelberg (2003)
327. Vincent, J.L., de Mendonca, A., Cantraine, F., et al.: Use of the sofa score to assess the incidence of organ dysfunction. Crit Care Medicine 26, 1793–1800 (1998)
328. Hurford, W.E., Bigatello, L.M., Haspel, K.L., Hess, D.R., Warren, R.L.: Critical care handbook of the Massachusetts General Hospital, 3rd edn. Lippincott Williams & Wilkins, Philadelphia (2000)
329. Revelation Software: Web site, `http://www.revelation.com/`
330. Ginsberg, M.L.: Approximate planning. Artificial Intelligence 76, 89–123 (1995)
331. Ginsberg, M.L.: A new algorithm for generative planning. In: Aiello, L.C., Doyle, J., Shapiro, S. (eds.) KR 1996: Principles of Knowledge Representation and Reasoning, pp. 186–197. Morgan Kaufmann, San Francisco (1996)
332. Fikes, R., Nilsson, N.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, 189–208 (1971)
333. Blum, A., Furst, M.: Fast planning through planning graph analysis. Artificial Intelligence 90, 281–300 (1997)
334. Ernst, M., Millstein, T.D., Weld, D.S.: Automatic SAT-compilation of planning problems. In: Proceedings of the IJCAI 1997, pp. 1169–1177 (1997)
335. Kautz, H., Selman, B.: Unifying SAT-based and graph-based planning. In: Minker, J. (ed.) Proceedings of the Workshop on Logic-Based Artificial Intelligence, Washington, DC, College Park, Maryland, Computer Science Department, University of Maryland, June 14–16 (1999)
336. Kautz, H.A., McAllester, D., Selman, B.: Encoding plans in propositional logic. In: Proceedings of the Fifth International Conference on the Principle of Knowledge Representation and Reasoning (KR 1996), pp. 374–384 (1996)
337. Kautz, H.A., Selman, B.: Planning as satisfiability. In: Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI 1992), pp. 359–363 (1992)
338. Gerevini, A., Serina, I.: LPG: a planner based on planning graphs with action costs. In: Proceedings of the Sixth International Conference on AI Planning and Scheduling, pp. 13–22. AAAI Press, Menlo Park (2002)
339. Bylander, T.: A linear programming heuristic for optimal planning. In: Proceedings of the 14th National Conference on Artificial Intelligence (AAAI 1997), Providence, Rhode Island, pp. 694–699. AAAI Press/MIT Press, Menlo Park (1997)
340. Gałuszka, A., Świerniak, A.: Translation STRIPS planning in multi-robot environment to linear programming. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS, vol. 3070, pp. 768–773. Springer, Heidelberg (2004)
341. van Beek, P., Chen, X.: Cplan: A constraint programming approach to planning. In: Proceedings of the 16th National Conference on Artificial Intelligence (IJCAI 1999), pp. 585–590 (1999)
342. Veloso, M.: Planning and Learning by Analogical Reasoning. Springer, Heidelberg (1994)
343. Vere, S.: Planning in time: Windows and durations for activities and goals. IEEE Transactions on Pattern Analysis and Machine Intelligence 5, 246–267 (1983)

344. Sacerdoti, E.: The nonlinear nature of plans. In: Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI 1975), pp. 206–214 (1975)
345. Tate, A.: Generating project networks. In: Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI 1977), pp. 888–893 (1977)
346. TLPlan: Project web site, http://www.cs.toronto.edu/~fbacchus/tlplan.html
347. Doherty, P., Kvarnstrom, J.: TALplanner: A temporal logic-based planner. AI Magazine 22, 95–102 (2001)
348. TALplanner: Project web site, http://www.ida.liu.se/~patdo/aiicssite1/kplab/projects/talplanner/
349. Anzai, Y.: Pattern recognition and machine learning. Academic Press, San Diego (1992)
350. Bernardinello, L., Cindio, F.D.: A survey of basic net models and modular net classes. In: Rozenberg, G. (ed.) APN 1992. LNCS, vol. 609. Springer, Heidelberg (1992)
351. Suyama, T., Yokoo, M.: Strategy/false-name proof protocols for combinatorial multi-attribute procurement auction. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), New York, NY, USA, August 19-23, 2004, pp. 160–167. IEEE Computer Society, Los Alamitos (2005)
352. Yokoo, M.: Protocol/mechanism design for cooperation/competition. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), New York, NY, USA, August 19-23, pp. 3–7. IEEE Computer Society, Los Alamitos (2005)
353. Góra, G., Bazan, J.G., Kruczek, P., Bazan-Socha, S., Skowron, A., Pietrzyk, J.: Case-based planning of treatment of infants with respiratory failure. In: Czaja, L. (ed.) Proceedings Workshop on Concurrency, Specification, and Programming (CS&P 2007), Łagów, Poland, Warsaw, Warsaw University, September 27-28, pp. 223–234 (2007)
354. Góra, G., Bazan, J.G., Kruczek, P., Bazan-Socha, S., Skowron, A., Pietrzyk, J.: Case-based planning of treatment of infants with respiratory failure. Fundamenta Informaticae (to appear, 2008)
355. Simulator, R.: Project web site, http://logic.mimuw.edu.pl/~bazan/simulator

A Road Simulator

Road simulator is a tool for generating data sets recording vehicle movement on the road and at the crossroads (see [179, 355]). Such data is extremely crucial in testing complex decision systems monitoring the situation on the road that are working on the basis of information coming from different devices. The simulator was constructed by the author of this paper.

Driving simulation takes place on a board (see Fig. 54) which presents a crossroad together with the access roads.

During the simulation the vehicles may enter the board from all four directions, that is, East, West, North, and South. The vehicles coming to the crossroad from South and North have the right of way in relation to the vehicles coming from West and East.

Each of the vehicles entering the board has only one aim, i.e., to drive through the crossroad safely and to leave the board. The simulation takes place step by

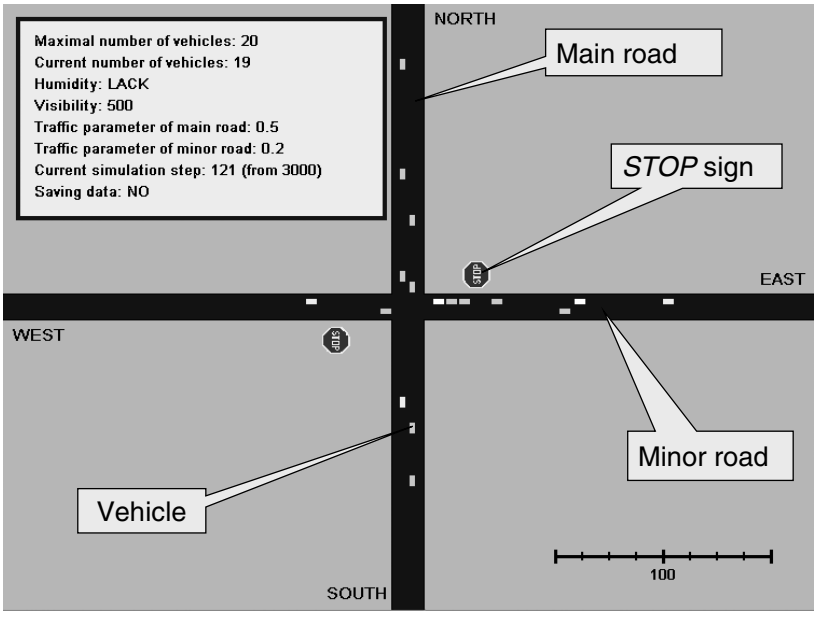


Fig. 54. The board of simulation

step and at each of its steps the vehicles may perform the following maneuvers during the simulation: passing, overtaking, changing direction (at the crossroad), changing lane, entering the traffic from the minor road into the main road, stopping, and pulling out.

Planning further steps of each vehicle takes place independently at every step of the simulation. Each vehicle is "observing" the surrounding situation on the road, keeping in mind its destination and its own parameters (driver's profile), makes an independent decision about its further steps; whether it should accelerate, decelerate and what (if any) maneuver should be commenced, continued, ended or stopped.

Making decisions concerning further driving, a given vehicle takes under consideration its parameters and the driving parameters of five vehicles next to it which are marked by FR1, FR2, FL, BR, and BL (see Fig. 55).

During the simulation the system registers a series of parameters of the local simulations, that is, simulations connected with each vehicle separately, as well as two global parameters of the simulation, that is, parameters connected with driving conditions during the simulation. The value of each simulation parameter may vary and can be treated as a certain attribute taking values in a specified value set.

We associate the simulation parameters with the readouts of different measuring devices or technical equipment placed inside the vehicle or in the outside environment (*e.g.*, by the road, in a helicopter observing the situation on the road, in a police car). These are devices and equipment playing the role of detecting devices

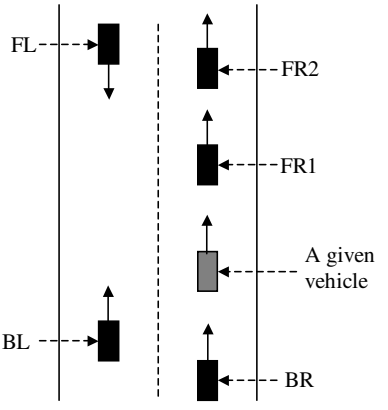


Fig. 55. A given vehicle and five vehicles next to it

or converters meaning sensors (*e.g.*, a thermometer, range finder, video camera, radar, image, and sound converter). The attributes taking the simulation parameter values, by analogy to devices providing their values, are called sensors.

The exemplary sensors are the following: initial and current road (four roads), distance from the crossroad (in screen units), current lane (two lanes), position of the vehicle on the road (values from 0.0 to 1.0), vehicle speed (values from 0.0 to 10.0), acceleration and deceleration, distance of a given vehicle from the vehicles FR1, FL, BR, and BL and between FR1 and FR2 (in screen units), appearance of the vehicle at the crossroad (binary values), visibility (expressed in screen units, values from 50 to 500), humidity (slipperiness) of the road (three values: lack of humidity (dry road), low humidity, and high humidity).

If, for some reason, the value of one of the sensors is not determined, the value of the parameter will become equal to NULL (missing value).

Apart from sensors, the simulator registers a few more attributes whose values are determined using the sensor values in a way determined by an expert. These parameters take the binary values and are therefore called concepts in the present simulator version. The results returned by testing concepts are very often in the form YES, NO or DOES NOT CONCERN (NULL value).

Here are exemplary concepts:

1. Is a vehicle forcing the right of way at the crossroad?
2. Is there free space on the right lane in order to end the overtaking maneuver?
3. Will a vehicle be able to overtake easily before the oncoming car?
4. Will a vehicle be able to brake before the crossroad?
5. Is the distance from the vehicle FR1 too short or do we predict that it may happen shortly?
6. Is a vehicle overtaking safely?
7. Is a vehicle driving safely?

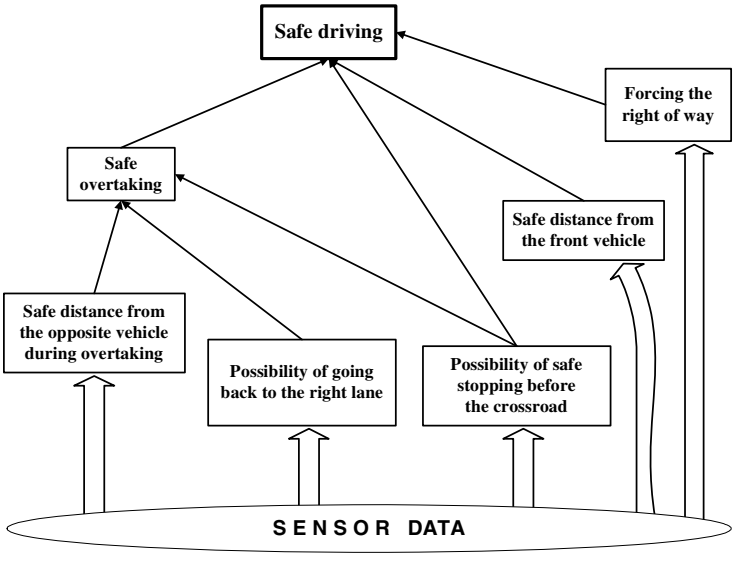


Fig. 56. The relationship diagram for the presented concepts

Besides binary concepts, simulator registers for any such concept one special attribute that approximates binary concept by six linearly ordered layers: *certainly YES*, *rather YES*, *possibly YES*, *possibly NO*, *rather NO*, and *certainly NO*.

Some concepts related to the situation of the road are simple and classifiers for them can be induced directly from sensor measurement but for more complex concepts this is infeasible. In searching for classifiers for such concepts domain knowledge can be helpful. The relationships among concepts represented in a domain knowledge can be used to construct hierarchical relationship diagrams. Such diagrams can be used to induce multi-layered classifiers for complex concepts (see, *e.g.*, [52, 182]). In Fig. 56 there is an exemplary relationship diagram for the concepts mentioned above .

The concept specification and concept dependencies are usually not given automatically in accumulated data sets. Therefore, they should be extracted from a domain knowledge. Hence, the role of human experts is very important in our approach.

During the simulation, when a new vehicle appears on the board, its so-called driver's profile is determined. It may take one of the following values: a very careful driver, a careful driver, and a careless driver. Driver's profile is the identity of the driver and according to this identity further decisions as to the way of driving are made.

Driver's profile is determined at the beginning when the vehicle appears on the board and cannot be changed until it disappears from the board.

For a given vehicle the driver's profile is determined randomly by the following probability distribution: a very careful driver 0.4, careful driver 0.25, and careless driver 0.35.

Depending on the driver's profile and weather conditions (humidity of the road and visibility), speed limits are determined which cannot be exceeded.

The humidity of the road influences the length of braking distance for depending on humidity, different speed changes take place within one simulation step, with the same braking mode.

The driver's profile influences the speed limits dictated by visibility. If another vehicle is invisible for a given vehicle, this vehicle is not taken into consideration in the independent planning of further driving by a given car. Because this may cause dangerous situations, depending on the driver's profile, there are speed limits for the vehicle.

The data generated during the simulation are stored in a data table (information system). Each row of the table depicts the situation of a single vehicle and the sensor and concept values are registered for a given vehicle and the vehicles FR1, FR2, FL, BL, and BR (associated with a given vehicle). Within each simulation step, descriptions of situations of all the vehicles on the road are saved to a file.

B Neonatal Respiratory Failure

The new possibilities in medical intensive care have appeared during last decades thanks to the progress in medical and technical sciences. This progress allowed us to save the live of prematurely born infants including the smallest born between the 20th and the 24th week of gestation with the birth weight above 500g.

Prematurely born infants demonstrate numerous abnormalities in their first weeks of life. Their survival, especially without severe multiorgan complications is possible with appropriate treatment. Prematurity can be characterized as inappropriate maturity of systems and organs leading to their dysfunction after birth.

The respiratory system dysfunction appearing in the first hours of life and leading to respiratory failure is the most important single factor limiting survival of our smallest patients. The respiratory failure is defined as inappropriate blood oxygenation and accumulation of carbon dioxide and is diagnosed based on arterial blood gases measurements. Clinical symptoms increased rate of breathing, accessory respiratory muscles use as well as X-ray lung examination are also included in assessment of the severity of respiratory failure (see, e.g, [328] for more details).

The most important cause of respiratory failure in prematurely born infants is RDS (*respiratory distress syndrome*). RDS is evoked by lung immaturity and surfactant deficiency. The other co-existing abnormalities PDA[5] (*patent duc-*

[5] PDA (patent ductus arteriosus) is a heart problem that occurs soon after birth in some infants. In PDA, there is an abnormal circulation of blood between two of the major arteries near the heart. Before birth, the two major arteries, i.e., the aorta and the pulmonary artery, are normally connected by a blood vessel called the *ductus arteriosus*, which is an essential part of the fetal circulation. After birth, the vessel is supposed to close within a few days as part of the normal changes occurring in the

tus arteriosus), sepsis (generalized reaction on infection leading to multiorgan failure), and Ureaplasma lung infection (acquired during pregnancy or birth) may exacerbate the course of respiratory failure. Each of these conditions can be treated as an unrelated disease requiring a separate treatment. But they co-exist in a patient very often, so in a single patient we may deal with their combination, for example, RDS + PDA + sepsis. In the holistic therapeutic approach, it is important to synchronize the treatment of the co-existing abnormalities, which can finally lead to cure from the respiratory failure.

The respiratory failure dominates in clinical course of prematurity but is not the only factor limiting the success of treatment. Effective care of the prematurely born infant should include all co-existing abnormalities such as infections, both congenital and acquired, water-electrolyte and acid-base imbalance, circulatory, kidney, and other problems. All these factors are related and they influence one another. The care of the prematurely born infants in their first days of life requires continuous analysis of plenty of the parameters including vital sings and the results of the additional tests. These parameters can be divided into stationary (*e.g.*, gestational age, birth weight, Apgar score) and continuous changing in time. The continuous values can be examined on the discrete (*e.g.*, blood gases) or the continuous basis, *e.g.*, with the monitoring devices (oxygen hemoglobin saturation, hear rate, blood pressure, temperature, and lung mechanics). The neonatal care includes assessment of imagine techniques results (ultrasound of the brain, echocardiography, chest X-ray). The global analysis should also include current methods of treatment applied in the particular patients. They may have qualitative (*e.g.*, administration of medication) or quantitative (*e.g.*, respiratory settings) characteristics.

Everyday analysis of numerous parameters requires great theoretical knowledge and practical experience. It is worth mentioning that this analysis should be quick and precise. Assessment of the patient's state is performed very often under rush and stress conditions.

A very important element of this analysis is an appropriate assessment of the risk of death of the small patient caused by the respiratory failure during next hours or days. The appropriate assessment of this risk leads to the decision of a particular method and level of treatment. The life of a sick child depends on this quick and correct decision. It should be emphasized that the correct assessment of the risk of death depends not only on analysis of the current clinical status, lab tests, and imagine techniques results but also on the dynamics observed lately and the character of changes (*e.g.*, progression of the blood gases indices of respiratory failure). The additional risk parameters such as birth weight are also important.

Computer techniques can be very useful in the face of difficulties in an effective data analysis. They may provide a support for the physician in everyday

infant's circulation. In some infants, however, the ductus arteriosus remains open (patent). If an infant has a PDA, but has an otherwise normal heart, the PDA may shrink and go away completely. If a PDA does not shrink, or is due to causes other than prematurity, surgery may be needed. This surgery is called *ligation* and involves placing a suture around the ductus to close it (see, e.g, [328] for more details).

diagnostic-therapeutic process both as a collecting, storing and patient's data presenting tools (*e.g.*, Neonatal Information System [329]) and as a tool of quick, automatic and intelligent analysis of this data. This approach might enable a computer presentation of some information based on the observed patterns which might be helpful in planning of the treatment. An example is the tool detecting patterns of changes in the newborn clinical status which lead to death with high probability. This kind of patterns is called the risk patterns (see Section 6.23). In this approach, a given patient is treated as an investigated complex dynamical system, whilst diseases of this patient (RDS, PDA, sepsis, Ureaplasma, and the respiratory failure) are treated as complex objects changing and interacting over time (see Section 6.22).

Index of Main Symbols

Symbol	Description	Page
CDS	A complex dynamic system	478
RSES	The Rough Set Exploration System	502
$\mathbf{A} = (U, A)$	An information system	503
V_a	A value set of an attribute a	503
$\mathbf{A} = (U, A, d)$	A decision table	503
$IND_{\mathbf{A}}(B)$	A B-indiscernibility relation in a decision table $\mathbf{A}$	504
$[u]_{IND_{\mathbf{A}}(B)}$	An equivalence class of a relation $IND_{\mathbf{A}}(B)$, such that an object u belongs to this class	504
$RED_{\mathbf{A}}(A)$	A set of all reducts in an information system $\mathbf{A}$	504
$M(\mathbf{A})$	A discernibility matrix of an information system $\mathbf{A}$	504
$f_{\mathbf{A}}$	A discernibility function for an information system $\mathbf{A}$	504
$\underline{B}X$, $\overline{B}X$	B-lower and B-upper approximations of a set X in an information system $\mathbf{A}$	505
$BN_B(X)$	B-boundary of a set X (boundary region)	505
$POS_B(d)$	A B-positive region of a decision table $\mathbf{A} = (U, A, d)$	505
$RED(\mathbf{A}, d)$	A set of all relative reducts of a decision table $\mathbf{A}$	506
$Pred(r)$	A predecessor of a rule r	508
$Succ(r)$	A successor of a rule r	508
$Match_{\mathbf{A}}(r)$	A number of objects matched by a decision rule r	508
$Supp_{\mathbf{A}}(r)$	A number of objects supporting a decision rule r	508
$\mu_{\mathbf{A}}(r)$	A coefficient of consistency of a rule r	508

$RUL(\mathbf{A})$	A set of all optimal basic decision rules of an information system $\mathbf{A}$, i.e., a set of all rules with minimal numbers of descriptors of an information system $\mathbf{A}$	508
$\mathbf{A}(T)$	A sub-table of a decision table $\mathbf{A}$ containing all objects matching a template T	512
$\mathbf{A}(\neg T)$	A sub-table of a decision table $\mathbf{A}$ containing all objects not matching a template T	512
μ_C	A rough membership function for a concept C	514
$AS = (U, I, \nu)$	An approximation space	515
$LOW\,(AS, X)$	A lower approximation of a set X with respect to an approximation space AS	516
$UPP\,(AS, X)$	An upper approximation of a set X with respect to an approximation space AS	516
μ_C^E	A stratifying classifier of a concept C which classifies each tested object into one of the layers labeled with labels from a set E	526
$\mathcal{L}$	A language to define features of complex objects (a set of formulas over a given finite alphabet)	547
$\models_{\mathcal{L}}$	A satisfiability relation of a language $\mathcal{L}$	547
$\mathcal{P} = (X, \mathcal{L}, \models)$	A property system	547
$GDL(\mathbf{A})$	A generalized descriptor language of an information system $\mathbf{A}$	549
$DISM_{\mathbf{A}}$	A dissimilarity function of pairs of objects in an information system $\mathbf{A}$	550
$NL(\mathbf{A})$	A neighborhood language for an information system $\mathbf{A}$	551
$\mu_{DISM_{\mathbf{A}}}$	A dissimilarity classifier for an information system $\mathbf{A}$	552
TIS	A total information system	553
$PEC(\mu_C^E)$	A language of patterns extracted from a classifier μ_C^E	575
$\mathbf{A}_C$	A concept approximation table using stratifying classifiers for a concept C	577
$\mathbf{LT}_C$	A layer table for a concept C	581
AR-scheme	An approximate reasoning scheme	588
$\mathbf{T}$	A temporal information system	596
$ETW(\mathbf{T})$	A language for extracting time windows from system $\mathbf{T}$	600
$TW(\mathbf{T})$	A family of all time windows from a c-temporal information system $\mathbf{T}$	601
$TW(\mathbf{T}, s)$	A family of all time windows from a c-temporal information system $\mathbf{T}$ with length equals to s	601

$FTW(\mathbf{T})$	A language for definnig features of time windows of c-temporal information system $\mathbf{T}$	603
$\overline{\mathbf{T}} = (\overline{U}, \overline{A})$	An information system of time windows (TW-information system)	605
$ECTW(\mathbf{T})$	A language for extracting clusters of time windows from a system $\mathbf{T}$ ($ECTW$-language)	610
$FCTW(\overline{\mathbf{T}})$	A information system of time windows for a system $\mathbf{T}$ ($FCTW$-language)	612
$\overline{\overline{\mathbf{T}}} = (\overline{\overline{U}}, \overline{\overline{A}})$	An information system of clusters of time windows (CTW-information system)	613
$\overline{\overline{\mathbf{T}}}_C = (\overline{\overline{U}}, \overline{\overline{A}} \cup \{d_C\})$	A temporal concept table for a concept C	614
$G = (V, E)$	A behavioral graph	617
$PATH(G, l)$	A family of all temporal paths with a length l in a behavioral graph G	617
SA	A sweeping algorithm around a complex object	620
$ESTW(\mathbf{T})$	A language for extracting sequences of time windows from system $\mathbf{T}$ ($ESTW$-language)	625
$SEQ(\mathbf{T})$	A family of all sequences of time windows of a given c-temporal information system $\mathbf{T}$	626
$SEQ(\mathbf{T}, s)$	A family of all sequences of time windows of a length s of a given c-temporal information system $\mathbf{T}$	626
$SEQ(\mathbf{T}, l, s)$	A family of all sequences of time windows of a length s of a given c-temporal information system $\mathbf{T}$ that they are windows of a length l	626
$Subsequence(S, i, j)$	A sub-sequence of a sequence of time windows S, where i and j are indexes of time windows, such that $i < j$	626
$Partition(W, k)$	A k-partition of a time window W	627
$FTP(\mathbf{G})$	A language for defining features of temporal paths of behavioral graph $\mathbf{G}$	629
$\overline{\mathbf{G}} = (\overline{U}, \overline{A})$	An information system of temporal paths (TP-information system)	630
$DPATH(\mathbf{G}, s)$	A set of all temporal paths observed in data of duration s	630
$ECTP(\mathbf{G})$	A language for extracting clusters of temporal paths from a behavioral graph $\mathbf{G}$ ($ECTP$-language)	633
$FCTP(\mathbf{G})$	A language for definnig features of clusters of temporal paths for a behavioral graph $\mathbf{G}$ ($FCTP$-language)	636
$\overline{\overline{\mathbf{G}}} = (\overline{\overline{U}}, \overline{\overline{A}})$	An information system of clusters of temporal paths (CTP-information system)	637

$\mathbf{G}_R = (V_R, E_R)$	A behavioral graph representing changes of relations between parts of structured objects	640
$\mathbf{G}_{\bowtie} = (U_{\bowtie}, A_{\bowtie})$	An information system of clusters of temporal paths for structured objects ($SCTP$-information system)	640
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	A behavioral graph for a structured object	644
$PATH(\mathcal{G}, l)$	A family of all temporal paths with length l in a behavioral graph $\mathcal{G}$	644
$\mathbf{PG} = (S, A, E)$	A planning graph	663
$PATH(\mathbf{PG})$	A family of all paths in a planning graph $\mathbf{PG}$	663
$PATH(\mathbf{PG}, k)$	A family of all paths with length k in a planning graph $\mathbf{PG}$	663
$Subpath(p, i, j)$	A sub-path of a path p, where i and j are indexes of nodes, such that $i < j$	663
$PLAN(\mathbf{PG})$	A family of all plans in a planning graph $\mathbf{PG}$	663
$PLAN(\mathbf{PG}, k)$	A family of all plans with length k in a planning graph $\mathbf{PG}$	663
$Subplan(p, i, j)$	A sub-plan of a plan p, where i and j are indexes of states, such that $i < j$	663
$FPPG(\mathbf{PG})$	A language for defining features of paths in planning graphs ($FPPG$-language)	668
$TW(\mathbf{T}, p)$	A set of all time windows of a given plan p in system $\mathbf{T}$	673
$DPLAN(\mathbf{T}, \mathbf{PG}, k)$	A set of all plans, occurring in system $\mathbf{T}$ of a length k	673
$DPLAN(\mathbf{T}, \mathbf{PG}, s, k)$	A set of all plans, occurring in system $\mathbf{T}$ and ending with state s and of a length k	673
$\mathbf{RT}(s, k)$	A resolving table for a state s from a planning graph constructed using a length of path k	673
MAP1	The low-intensity mechanical ventilation	675
MAP2	The middling-intensity mechanical ventilation	675
MAP3	The high-intensity mechanical ventilation	675
CPAP	The continuous positive airway pressure	675
$\mu_{\mathbf{RT}(s,k)}$	A resolving classifier constructed for a table $\mathbf{RT}(s, k)$	676
$PairList(\mu_{\mathbf{RC}(s,k)}(p))$	A list of pairs $(action, state)$ which a classifier $\mu_{\mathbf{RC}(s,k)}$ returns to a path p	676
EFS	The expert forward search algorithm	678
$EEFS$	The exhaustive expert forward search algorithm	679
$FEEFS$	The full exhaustive expert forward search algorithm	680
$DISM_{\mathbf{PG}}$	A function of dissimilarity between states from a planning graph $\mathbf{PG}$	683

$\mathbf{DIT_{PG}}$	A table of dissimilarity between states from a planning graph **PG**	683
$\mu_{DIT(\mathbf{PG})}$	A classifier of dissimilarity between states from a planning graph **PG**	683
Reconstruction	The algorithm of partial reconstruction of a plan for unstructured objects	686
$\mathcal{PG} = (\mathcal{S}, \mathcal{A}, \mathcal{E})$	A planning graph for structured objects	688
PGF	A family of planning graphs	690
$GPLAN(\mathbf{PGF}, k)$	A set of all g-plans with length k for a family of planing graphs **PGF**	690
$DGPLAN(\mathbf{T}, \mathbf{PGF}, k)$	A set of all g-plans occurring in a system **T** with length k and constructed for a family of planing graphs **PGF**	690
$\mathbf{ET}(\mathbf{T}, \mathbf{PGF}, k)$	An elimination table of g-plans from a set $GPLAN(\mathbf{PGF}, k)$	691
$\mathbf{ERUL}(\mathbf{T}, \mathbf{PGF}, k)$	A set of all elimination rules for an elimination table $\mathbf{ET}(\mathbf{T}, \mathbf{PGF}, k)$	693
$\mu_{\mathbf{ET}(\mathbf{T},\mathbf{PGF},k)}$	An elimination classifier based on a set $\mathbf{ERUL}(\mathbf{T}, \mathbf{PGF}, k)$	694
$\mathbf{MAT}(\mathbf{T}, \mathbf{PGF}, k)$	A meta action table of g-plans for structured objects	695
$\mu_{\mathbf{MAT}(\mathbf{T},\mathbf{PGF},k)}$	A meta action classifier constructed for a table $\mathbf{MAT}(\mathbf{T}, \mathbf{PGF}, k)$	695
SearchGPlanMA	The algorithm of searching of g-plan for a meta action	699
SearchGPlan	The algorithm of searching of g-plan for states	700
EEFSS	The exhaustive expert forward search algorithm for a structured object	701
GReconstruction	The algorithm of partially reconstruction of a g-plan	706
MReconstruction	The algorithm of reconstruction of a plan for structured objects	707
RDS	The respiratory distress syndrome	744
PDA	Patent ductus arteriosus	744