On the Way to Learning Deterministic Objects

Jindřich Bůcha

Institute of Information Theory and Automation, Pod vodárenskou věží 4, 182 08 Prague, Czech Republic
bucha@utia.cas.cz

Abstract. The paper deals with learning knowledge about objects, i.e. entities of the real environment. This is an important topic, often neglected by machine learning. The whole experimental approach is implemented via the integration of several areas of AI, namely machine learning, knowledge base management, reasoning, especially Prolog-like and analogical. More specifically, the approach is based on further generalization of already learned (generalized) rules, and on analogy. How to obtain suitable rules and how to organize learned knowledge into ontology is analyzed too. The ontology has special properties, like rules, interconnection of descriptions (concepts) via interfaces, use of views, and modularization.

1 Introduction

Look at

```
at(WK,g1).
```

It is a plain predicate from a chess domain saying “white king WK is at the field g1”. Such predicate is a basic building block of many knowledge bases. In machine learning (ML), much attention is devoted to the creation of predicates and their specification via rules, especially in inductive logic programming (ILP) [20]. It is known, how to build them. Much attention is devoted to their organization into knowledge bases (ontologies) [23]. However, nearly no attention is devoted to the identification of *objects*, like king and field, and creation of their descriptions. Yet still, the use of KB does assume knowledge of such objects. This paper persuades this *topic*, i.e. how to learn descriptions of deterministic objects.

“A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.” [19]. It is a good and well-accepted definition (see e.g. [9], [13], [14], [21]). However, Mitchell [19] found it useful “to reduce the problem of improving performance P at task T to the problem of learning some particular target function”. Most of the contemporary ML systems correspond to this reduced definition. Corresponding consequences reveal when we try to apply resulting knowledge: It is necessary to add other steps, e.g. 1) formulate problem, 2) determine representation, 3) collect training data, 4) evaluate the learned knowledge, and 5) field the knowledge base [13]. Especially steps 1) and 2) are practically *not automated at all*.

In contemporary ML, *stochastic approaches* prevail. However, the environment has sometimes a deterministic character too and corresponding deterministic approaches to ML can be simpler and more efficient. This can be illustrated on the task of parity recognition. The reported accuracy of this task solved by neural network increased with number of hidden neurons. For ten neurons it was 99.3% [26]. However, inductive logic programming can reach directly just 100% [32].

Kaelbling et al [11] in their study say: “We are interested in building systems that learn to interact with complex real world environments, by representing the dynamics of the world with models that allow strong generalization through representation in terms of objects. Humans speak (and apparently think) of the world as being made up of objects. There are chairs and apples and clouds and meetings … it is hard to imagine a truly intelligent agent that does not conceive of the world in terms of objects and their properties and relations to other objects.” [11] paper, which is an introductory study to the problematic, and Kaelbling course [10], show how fresh the problem of learning objects is. Besides the [11] paper there is only one concrete approach (as far as I know) directly oriented on discovering objects [12] [22].

I’m trying to responds to this situation. My general *goal* is to test and further precise the previously developed *framework* [6]. The framework is designed for complex learning in complex environments. This learning corresponds to the original not reduced definition of ML. Learning is done in the context of *problem solver* (PS). PS, in turn, works in the context of its environment. Our *environment is chess* and the task is to learn legal chess moves. It has an advantage that this is a moderately complex environment and that *we* (designers) *know some rules* governing the behavior of this environment and *we know some objects* in this environment. We can use this knowledge to influence the incremental design of PS. I have chosen this task even though Mitchell [19, p. 286] referring to [24] claims FOIL system “demonstrated to learn … to discriminate legal from illegal chess positions”. More precise description, motivating my choice, would be that FOIL correctly classified better then 95% of unseen cases in the chess endgame White King and Rook versus Black King [24, pp. 257-8]. Moreover, FOIL had knowledge about objects like White King, Rook, Black King, fields, and their relations. This is that kind of knowledge, which PS should try to discover.

To achieve our goal, learning must be *integrated* with other cognitive functions of PS, like reasoning and knowledge base management. In the framework, there are already integrated different areas of AI, like ML, Prolog-like clauses representation and deduction, ontological knowledge representation. It brings difficulties, e.g. terminological and technical. C’est la vie.

The rest of the paper is organized in two following sections. In the next section 2, a basic problem solver set-up is given. In the first subsection of the section 3, our approach to analogy and its relation to the creation of object description are described. Then, in the next subsection, learning suitable rules is described. In the last subsection of this section, the management of knowledge base is described. The *object* is a central concept of our framework. However, I am not going to give a precise specification of this concept. I am able to give (a lot of) examples of objects only (see above). The framework is implemented, mainly as a Smalltalk-based [30] system; used examples are (mostly) produced by this system. Its design is supported by Rational Rose [25] and Rational SoDA [29] tools. Its testing is based on SUnit [1],

[30]. Its rule based representation and deduction is based on SOUL [18]. Its visualization and user interface is partially based on WinBoard [17]. As the paper covers several areas of AI its extent is limited (see [7], [8] for more details).

2 Basic Set-up

The basic set-up consists of environment and problem solver (PS). On a conceptual level, it is described in [5], [6]. The *environment* is a simulated chess environment. It is governed by its set of rules. It allows legal moves only. *PS* should solve problems, i.e. its goal is to comply with that what the environment allows, with the rules of the environment. At the beginning, PS does not know these rules. It can only recognize that it has a *problem*, when it does not respect them. PS is connected with its environment by its inputs and outputs. PS should provide outputs, which do not cause the problem. It can use its inputs for this, i.e. it can use inputs and knowledge of a function, f: X→Y, where X and Y are spaces of inputs' and outputs' values. Practically no knowledge is pre-built into PS and only a trivial *interface* between PS and its environment is chosen.

This trivial interface does not mean a simple one. The *inputs* consist of 773 binary *attributes*, *outputs* of 5 integer and binary attributes. For each side, figure, and field of the chess environment, there is an input attribute (2x6x8x8=768 attributes) specifying whether a corresponding figure is at a corresponding field. As PS has no knowledge about sides, fields, and figures, it also has no knowledge about relations of these attributes to these objects. There are also other 5 input attributes specifying whether there is a problem identified by the *teacher*, p1, by board, p2, as a move problem, p3; p4 specifies if it is PS turn to move, p5 if there is a problem at all. These input attributes fully characterize the state of the environment (for our purposes). Using p5, moves can be evaluated and positive (good) and negative (wrong) *examples* of PS behavior can be gathered. This means that the evaluation is done by PS and is based on values provided by environment. Our teacher is a part of the environment and can influence the evaluation only indirectly and partially. o1, o2, o3, o4, o5 output attributes specify from which column, o1, and row, o2, to move and to which column, o3, and row, o4, to move, o5 specifies whether PS makes a move at all. The *performance* P of PS is probability of successful-good move.

This trivial interface is intentional. It is something like a situation of a new-born baby. We do know there are, in this environment, objects like fields, figures, columns, rows, and there are rules governing their behavior. For example, there is a rule like

```
at(o5,?X):-at(p4,?X)                                    (1)
```

where at(o5,1), means PS should move, at(o5,0) means PS should not move. I use a Prolog-like notation. There are clauses like at(o5,1), Prolog terms like o5 and 1. Some of these Prolog terms, e.g. o5, correspond to PS attributes, others to their values. This *very-little-knowledge* (assumption) aims at two things. First, to provide an opportunity to learn descriptions of these objects and rules, and second, to *analyze and govern learning processes*, which are discovering them.

As PS has initially practically no knowledge, there is a low probability of a good move, low performance P. It depends on specific situation, but it is roughly from 0.00012 to 0.05. This could lead to many negative examples as compared to positive ones, and consequently to low efficiency of learning. To overcome this, PS can follow *suggestions* of the teacher.

PS *knowledge base* (KB) is a net of object descriptions. They are related like concepts in an ontology, e.g. by generalization, association, and aggregation relations. There are three extensions as compared to the classical ontology structure [23]. First, descriptions can contain *rules* to describe behavior of corresponding objects. Rules are already present in ontologies like CLASSIC [4] and Loom [16]. What differentiates these ontologies from our one is that we do not assume consistency of rules, but allow *inconsistency*. This corresponds to a human approach to the real environment. Second, they can also contain inputs and outputs *interfaces* to relate them to other descriptions. Third, descriptions can be composed from other descriptions, *views*, which partially describe corresponding object. At the beginning of PS work, KB is nearly empty. It contains only descriptions corresponding to the attribute values. It also contains trivial descriptions of the whole environment, called *model*, and of the whole PS, called *plan*; here, a trivial description means a description without rules.

3 Learning

The previously described set-up allows the gathering of *history* and consequently the preparation of evaluated *examples* of PS behavior. Then we can use known heuristic methods [19], [15] allowing to find description of function f: X→Y, if we have a set of examples E, (x,y) є E, x є X, y є Y. We can *learn* description of function f. Function f can be described using rules, e.g. to distinguish positive and negative moves

```
at(p,0):-at(p4,0),at(p3,0),at(WPe4,0),at(o5,0)           (2)
at(p,1):-at(p4,1),at(p3,0),at(WPe3,1),at(WPe4,0)
at(p,0):-at(p4,1),at(p3,1),at(WPe3,1),at(WPe4,0),
at(o5,1)
...
```

where `at(p,0)`, resp. `at(p,1)` mean positive resp. negative consequences of combinations of inputs and outputs; `WPe3` and `WPe4` are names of input attributes used for visualization purposes. Problem[1] is that the space F of possible functions f є F is usually huge. The cardinality of F,

$$|F|=|Y|^{|X|-|E|}\ , \qquad (3)$$

[1] It is a problem of PS design. It should be distinguished from the problem of PS.

grows super-exponentially with number of input attributes [15]. This is not a new problem. It is necessary to constraint this space somehow. My approach to this problem is to use every suitable possibility (heuristics) to reduce this complexity.

3.1 Analogy and objects

The *basic idea* how to learn objects is following: Let PS follows a sequence of recommended similar moves, like 1. e3 ..., 2. e4 ..., 3. e5 ..., ... If we segment history into appropriate parts, we can learn corresponding descriptions. These descriptions are called *views*. From these views we can get a set of similar rules like

```
at(o3,5,o4,3):-at(WPe3,0)
at(o3,5,o4,4):-at(WPe4,0)
at(o3,5,o4,5):-at(WPe5,0)
...
```

and by *analogy* we can derive that this can be described as

```
at(o3,5,o4,?A1):-at(?A2,0),r(?A1,?A2)
r(3,WPe3). r(4,Wpe4). r(5,Wpe5). ...
```

and the sets of basic objects, i.e. variable value ranges

```
?A1~{3,4,5,...},?A2~{WPe3,Wpe4,Wpe5,...}
```

correspond to some possibly new objects. In fact, object {`3,4,5,...`} is not a new one; it corresponds to the values of o3 attribute. However, {`WPe3,Wpe4, Wpe5,...`} is new, it corresponds to something like a seed of e-column.

It showed that the task is more complicated. Learned 7, 2, and 7 rules look like

```
at(o3,4):-at(p3,0),at(WPe3,1),at(WPe4,0),at(o5,0)         (4)
at(o3,5):-at(p3,1),at(WPe3,1),at(WPe4,0),at(o5,1)
...

at(o3,4):-at(o5,1)                                         (5)
at(o3,3):-at(o5,0)

at(o3,4):-at(WPe4,1),at(WPe5,0),at(WPe6,0),at(o5,0)       (6)
at(o3,3):-at(WPe4,0),at(WPe5,0),           at(o5,0)
...
```

They are not very similar. The measure of *similarity* is chosen to be inversely proportional to the number of differences in compared descriptions. There are several possible reasons for their dissimilarity: A) Conditions for learning are changing as learned knowledge is immediately applied. B) If PS has no knowledge to use, it selects a move randomly. C) PS has no knowledge to segment examples properly. D) There are (two) different groups of attributes; I would call them local and global ones. Global ones are changing more often. Global ones are e.g. p3, o5, local e.g. WPe3. There are dependencies among global attributes and it is necessary to discover and use them first. E) The rules are relatively complex. When applying analogy in this situation we obtain rules like

```
at(?A1,?X):-at(?A2,?Y),not(?X,?Y),r(?A1,?A2)
```

with variable value ranges

```
?A1~{p5n,p4n,p3n,p5n,BPf2n,BPg5n,p1n,p3n,BPg6n,WPd2n,
p5n,WPe4n,p4n,p3n},
```

```
?A2~{p3, p5, o5, p5, BPf3, o5,   p3n,p4, BPg7, WPd3,
p1, o5,   o5, p3}
```

where name like p5n refers to next version of p5. They are obviously useless.

This means that on the way to learning objects we must first understand (learn) dependencies among (global) attributes, i.e. find ways how to create suitable rules describing these dependencies and use these rules to simplify views and, analogies.

3.2 Rules

There are several possibilities how to learn suitable rules: 1) orientate on changes, 2) eliminate redundant attributes, 3) use standard information gain [19], 4) reduce the number of used attributes, 5) use simplifications, 6) maintain dependency on released output attributes, 7) remove cyclic dependencies, 8) use recognized inconsistency, 9) use plan and model, 10) use learned descriptions immediately, 11) variabilize rules, 12) introduce variables into simple rules. I'll elaborate some.

Orientation on changes. The greatest influence on |F| has the exponent from (3). So, we try to reduce |X|. The first reduction possibility is an orientation on changes. The chess environment is relatively stable, e.g. in our experiments about 0.2% of input attributes changes their value in one move. PS focuses its attention on changing attributes only. It brings enormous reduction of cardinality of F.

Elimination of redundant attributes. The next reduction also focuses on |X|. It is done by eliminating redundant attributes form X. We consider attribute x_i redundant if

$$f(x_1, \dots x_{i-1}, x_i, x_{i+1}, \dots x_n) = f(x_1, \dots x_{i-1}, x_{i+1}, \dots x_n) \; . \tag{7}$$

Using the modified approach of [31], we could eliminate all redundant attributes. It is true that due to a restricted number of used learning examples some other in-fact non-redundant attributes can be considered redundant also. This could lead to a local extreme (from the viewpoint of simplicity of learned description) and to situation when really redundant attributes are not eliminated. However, this can be at least partially refined by the other PS possibilities.

Reduction of number of used attributes. The next reduction again focuses on |X|. It is possible to try to reduce the number of used attributes. We try to find a dependency among the smallest set of attributes. This set is determined as a set of attributes changed between (two) consecutive moves.

Simplifications. The next reduction focuses on |Y|. The simplification is a function derived from the original one by removing output attribute(s), i.e. from

$$y=(y_1, \ldots y_m)=f(x) , \qquad (\mathbf{8})$$

$$y_j=f_j(x) \qquad (\mathbf{9})$$

can be derived. (This possibility is used by [22]. They call it a functional transformation.) Examples of such learned descriptions are

```
at(o5,?X):-at(p4,?X)                                    (10)

at(p4n,?X):-at(o5,?Y),not(?X,?Y)                        (11)

at(p3n,1):-at(p4,1),at(o5,0)                            (12)
at(p3n,0):-at(p4,1),at(o5,1)
at(p3n,1):-at(p4,0),at(o5,1)
```

The example of learned not simplified description, so called equivalent description, is

```
at(p,?X):-at(p5n,?X)                                    (13)
```

This “describes the whole model”, i.e. how the most important attribute of the model, p, depends on the remaining attributes. (How to learn the dependency of p5n on other attributes still remains to be solved - learned.)
Maintenance of dependency. Simplification can depend on removed output attributes. It is possible to keep this dependency by including these output attributes among potential input attributes. The whole hierarchy of dependent descriptions may be created.

3.3 Ontology

The exploitation of possibilities listed in the previous subsections leads and, at the same time, is driven by an organization of knowledge descriptions into some kind of ontology. What is the aim of this *ontology*? It is given by the necessity to describe relations among descriptions. Here, we describe a part of our ontology, this one, supporting the creation of analogies (and learning objects). It is illustrated by Fig. 1.

Above, we introduced plan, model, view, and simplification. They are organized in the following way. Plan and model are similar structures (similar in a general sense, not in the sense introduced above). They differ in that the inputs and outputs of plan are identical to the inputs and outputs of PS. The inputs of the model consist of the inputs and outputs of PS. The outputs of the model consist of next inputs of PS. It means that model determines what happens if, in some environment state given by PS inputs, PS outputs are applied. It determines the next environment state. This is given by next inputs of PS. Plan or model form a *context*. During learning, the contexts are iteratively filled with new descriptions.

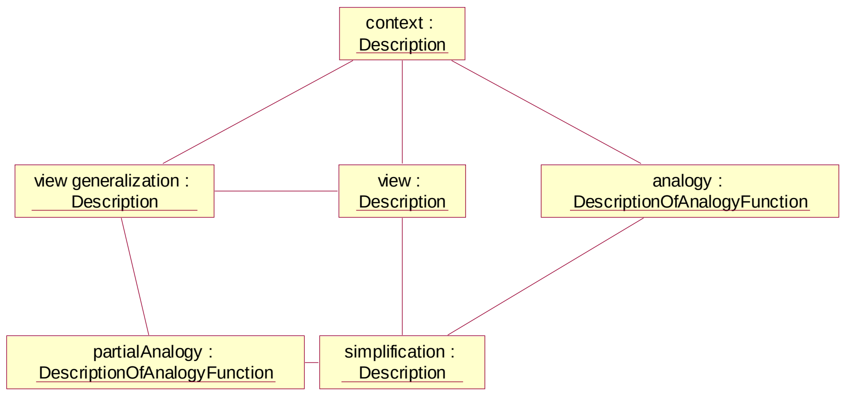
context : Description
view generalization : Description
view : Description
analogy : DescriptionOfAnalogyFunction
partialAnalogy : DescriptionOfAnalogyFunction
simplification : Description

Fig. 1. Part of ontology used for analogies. Boxes correspond to objects, now in a software sense. The first part of name is an object name, the second is a corresponding class name. Box connections show relations between objects. In UML terms [3], this is a type of object diagram.

On the first level, the context is described by or consists of particular *views*. (This will be refined later.) Each view is in turn described either directly by so called full description (not in the Figure) or by a collection of *simplifications*. In this knowledge structure, analogies are searched by searching for similar descriptions in all views. There are compared only these descriptions which have the same structure, i.e. same number of rules and same number of predicates in rule's conditions. If similar descriptions are found, the search continues as the searching mechanisms attempts to find all similar descriptions corresponding to the same *analogy*. It is also checked if the description does not belong to an already found analogy. The newly created analogy is connected to the context.

This kind of search is used also to search for generally valid descriptions, i.e. descriptions valid in more views. In that search directly the sameness is required. If some same descriptions are found then their generality is examined. It happens that in some view, this just found description is not learnable, yet it is consistent with the view's examples. In such case, this description is considered valid also for this view. If this description is valid for all views of the context, it is connected directly to this context as analogy and at the same time, the original description is removed from its original position. View histories contain changes only. To be able to examine the generality, views must be connected by *anchors* (not in the Figure), which store relative differences between corresponding view histories. When a new view is processed, it is only checked if corresponding analogical description is valid for the view's examples. This is the way in which descriptions (10-13) were found. If a new description is not valid for all views of the context, a new view is created as view generalization, this description is connected to this view as a partial analogy, this new view generalization is connected with original views, and again the original descriptions are removed from their original positions.

4 Conclusion

It is not a problem to generate as many rules as you like, if you have enough examples. This is our case. The problem is to generate - learn useful rules. This paper describes an experimental *approach* to one significant and unsolved task of machine learning, namely that of learning objects. Learning objects is not an easy task. Our *contribution* is a contribution to the understanding of this task and its solution. The understanding was gained via gradual solution of the task and revealed some suitable steps towards the solution itself. Learning useful rules seems to be one of such steps. Specifically, this approach has resulted in *learning several useful rules*, (10-13).

The core idea of our approach is the integration of more AI concepts – functions into one system. It is not new. In AI, this is slowly, painfully, but steadily pushed ahead. In the introduction, we mentioned the steps, which are necessary to achieve full learning. They exemplify the integration. These steps are (of course) implemented in our framework. They are there in simplified versions. The implementation is not simple, contrariwise, it is reasonably complex. For this reason it is based on a principle: “make it work, as simple as possible”. Thence the breadth given by the attempt to integrate necessarily brings some shallowness. However, I see it as a necessary way ahead.

There is a huge space of possible further developments and improvements. They can range from small and miner improvements to important and great ones. Among small, we can name e.g. a delayed use of learned descriptions, which could bring more stability and similarity into knowledge base and consequently improvement in discovering analogies. Among greater, we can mention the introduction of self-control or the extension of range of learning methods – representations of descriptions. However, these are only the implementations of the other parts of our general framework.

Acknowledgements

This work has been partially supported by the grant GA ČR No. 102/99/1564. I would also like to thank to anonymous reviewers.

References

1. Beck K.: Simple Smalltalk Testing: With Patterns. http://www.xprogramming.com/
2. Benjamins V. R. (Ed.): Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends, Stockholm (1999)
3. Booch G., Rumbaugh J., and Jacobson I.: The Unified Modeling Language User Guide. Addison-Wesley, Reading (1999)
4. Brachman R. J., McGuinness D. L., Patel-Schneider P. F., Borgida A.: „Reducing“ CLASSIC to practice: Knowledge representation theory meets reality. Artificial Intelligence 114 (1999) 203-237

5. Bůcha J.: Problem Solving with Knowledge Represented by Probabilistic Automata Network, Ph.D. Thesis. Czechoslovakian Academy of Science, Prague, (1978) (in Czech)
6. Bůcha J.: On Efficiency of Learning: A Framework and Justification. In: Staab S., Maedche A., Nedellec C., Wiemer-Hastings P. (eds.): Proceedings of the First Workshop on Ontology Learning. ECAI, Berlin (2000), 51-53
7. Bůcha J.: Experimental Learning System, Software Documentation, Version 1, Research Report 2062. UTIA AV CR, Prague (2002), 345 p.
8. Bůcha J.: On the Way to Learning Deterministic Objects, http://www.utia.cas.cz (2003)
9. Flach P. A.: On the state of the art in machine learning: A personal review. Artificial Intelligence 131 (2001) 199–222
10. Kaelbling L. P.: Learning in Worlds with Objects, Course. http://www.ai.mit.edu/ (2002)
11. Kaelbling L. P., Oates T., Hernandez N. G., and Finney S.: "Learning in Worlds with Objects" AAAI Stanford Spring Symposium on Learning Grounded Representations (2001)
12. Kuipers B.: The Spatial Semantic Hierarchy. Artificial Intelligence J. 119 (2000) 191-233
13. Langley P. and Simon H. A.: Applications of Machine Learning and Rule Induction. Communications of the ACM 38 (1995):55-64
14. Langley P.: Preface: The Maturing Science of Machine Learning. In Proceedings of the Seventeenth International Conference on Machine Learning (2000)
15. Langley P.: Elements of Machine Learning. Morgan Kaufman Pub., San Francisco, (1997)
16. Loom Tutorial, Loom Release 2.1, http://www.isi.edu/isd/LOOM/ (1995)
17. Mann T.: WinBoard: Chessboard for Windows. http://www.tim-mann.org/winboard/ (2001)
18. De Meuter W., Brichau J. and Mens K.: SOUL Manual. http://prog.vub.ac.be/research/DMP/soul/SOULManual.pdf (2002)
19. Mitchell T. M.: Machine Learning. McGraw Hill, N.Y. (1997)
20. Muggleton S.: Inductive Logic Programming: Issues, results and the challenge of Learning Language in Logic. Artificial Intelligence 114 (1999) 283-296
21. Nilsson N. J.: Introduction to Machine Learning. http://robotics.stanford.edu/ (1996)
22. Pierce D. and Kuipers B.: Map learning with uninterpreted sensors and effectors. Artificial Intelligence Journal 92 (1997) 169-229
23. Perez A. G., Benjamins V. R.: Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods. In Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends (1999)
24. Quinlan J. R.: Learning Logical Definitions from Relations. Machine Learning 5 (1990) 239-266
25. Rational Rose, Using Rose. Rational Software Corporation, USA (2000)
26. Sarle W. S.: Re: [ML] Parity Problem, "Machine Learning" machine-learning@egroups.com, 29 Nov (2000)
27. Shen WM. and WM. Leng WM.: A Metapattern-Based Automated Discovery Loop for Integrated Data Mining. IEEE Transactions on Data and Knowledge Engineering 8 (1996) 898-910
28. Shen WM.: Functional Transformations in AI discovery systems. Artificial Intelligence 41 (1990) 257-272
29. Using Rational SoDA for Word, Rational Software Corporation (2000)
30. VisualWorks, Application Developer’s Guide. Cicom Systems, Inc., Cincinnati (2002)
31. Zupan B, Bohanec M, Demsar J, Bratko I.: Learning by discovering concept hierarchies. Artificial Intelligence 109 (1999) 211-242
32. Železný F.: personal communication (2000)