Conceptualizing Procedural Knowledge Targeted at Students with Different Skill Levels

Martin Možina, Matej Guid, Aleksander Sadikov, Vida Groznik, Jana Krivec, and Ivan Bratko
Contact email: martin.mozina@fri.uni-lj.si
University of Ljubljana, Faculty of Computer and Information Science, Slovenia

Abstract. Conceptualizing procedural knowledge is one of the most challenging tasks of building systems for intelligent tutoring. We present a novel algorithm that enables teachers to accomplish this task (semi)automatically. Furthermore, it is desired to adapt the level of conceptualization to the skill level of particular students. We argue that our algorithm facilitates such adaptation in a straightforward fashion. We demonstrate this feature of the algorithm with a case study.

1 Introduction

Domain conceptualization lies at the very core of building an intelligent tutoring system (ITS) [5, 7]. This involves the structuring of the domain and creating a vocabulary or ontology of key concepts. Domain conceptualization enables the implementation of a sophisticated expert module that is capable of sensible interaction with the student, can diagnose student's errors, and can generate situation-dependent explanation for the student. Procedural knowledge, which generally speaking is the knowledge exercised in the performance at some task, is – in contrast to declarative knowledge – usually more implicit and not easily articulated by the individual. Due to its tacit nature this kind of knowledge is often very hard to conceptualize. Moreover, it is even harder to conduct successful conceptualization of procedural knowledge at different levels with respect to types and amount of skills that final users possess.

Different students are at different skill levels. To enable a more successful and/or interesting learning experience, it is imperative that tutoring systems exhibit different conceptualizations or different levels of conceptualization. Take, for example, an exercise from high school physics. The aim is to solve for the acceleration of an object on a slope given its mass, the force of friction, and the angle of the slope. The role of the ITS is to help the student solve the problem by guiding him or her through the solving process with hints. These hints can be of various levels of complexity, and should be adjusted for the particular student's skill level. Such hints are actually intermediate goals, represented as concepts. For our problem, using the second Newton's law is an appropriate hint for more advanced students, while for others a more direct hint is needed (like split the force of gravity to obtain its part in the direction of the slope). An extreme case would be to guide the student step by step, which requires that all possible nuances of discussed physical problems are stored in a database along with their hints/concepts (e.g., calculate the force of gravity, split the force of gravity into two parts, calculate its part in the direction of the slope by using sinus of the angle, calculate the force acting on the object in the direction of the slope, and finally calculate the acceleration of the object). The other extreme is to just supply the student with all the basic physics/math formulae, and let him find the solution by searching for the appropriate combination of steps.

In this paper, we describe an algorithm for (semi) automated domain conceptualization of procedural knowledge that should enable teachers to more easily express their intended (for students) view of domain representation. A very important characteristic of the algorithm is its inherent ability to enable teachers to produce different levels of conceptualization in a very straightforward way, tuning only two easy-to-understand parameters.

2 Conceptualization of Symbolic Domains

In this paper, we will consider symbolic problem solving domains where problem solving is based on reasoning with symbolic descriptions (like in physics, mathematics, or games like chess). A particular domain is defined with a basic domain theory (*e.g.* rules of chess) and a solution to be achieved (*e.g.* checkmate the opponent in chess). The task is to find a sequence of steps (or actions) that bring us from the beginning state of the problem (definition of the problem) to the goal state (the solution).

The basic domain theory (or basic declarative knowledge of the domain) is usually simple and easy to remember and is, in principle, sufficient for solving problems (e.g. knowing rules of chess could in theory enable optimal play). However, finding a solution using only declarative knowledge would require far too extensive searching. A human student is incapable of searching very deep, therefore we need to teach him also the procedural knowledge – how to solve problems. The "complete" procedural knowledge would be a function mapping from each problem state to a corresponding action that leads to the solution of the problem. In chess such complete knowledge (called a "tablebase") is computed for some endgames. A tablebase effectively specifies best moves for all possible positions. Such tablebases logically follow from the rules of the game and can be viewed as a compilation of the rules into an extensive form. Tablebases can be used easily because they only requires trivial amount of search. But now the problem is the space complexity – it is impossible for humans to memorize such tablebases that typically contain millions of positions.

There is a way, however, that enables humans to solve problems in such chess endgames quite comfortably. The key is that humans use some intermediate representation of the problem that lies between the rules of the game (or the corresponding tablebase) and solutions. We call such an intermediate representation a "conceptualized domain". Powerful conceptualizations are sufficiently "small" that they can be memorized by a human, and they contain concepts that enable fast derivation of solutions. Such a domain conceptualization enables effective reasoning about problems and solutions. [6]

In this paper, we propose a goal-oriented conceptualization of domains and explore how to (semi) automatically construct such a conceptualization that can be effectively used in teaching problem-solving. We will assume that the optimal solution is computable and annotate each problem state with a distance-to-goal value, that is, the minimal number of

steps required to reach a solution[1]. To enable automated learning, each state will be described with a vector of attributes that correspond to some known domain concepts.

3 Goal – Oriented Rules

A goal-oriented rule has the following structure:

IF preconditions THEN goal (depth)

The rule's *preconditions* and *goal* are both expressed in terms of attributes used for describing states. The term *preconditions* is a conjunction of simple conditions that specify the required value of an attribute. For example, in chess the *kdist* attribute stands for the distance between kings, and preconditions could contain *kdist=3* or a threshold on an attribute value, e.g. *kdist>3*. Similarly, a *goal* is a conjunction of subgoals, where a subgoal can specify the desired value of an attribute (e.g. *kdist=3*) or any of the four possible qualitative changes of an attribute given the initial value: {*decrease, increase, not decrease, not increase*} or its optimization: *minimize, maximize*. For example, a subgoal can be "*decrease kdist*" (decrease distance between kings). The *depth* property of a rule specifies the maximum allowed number of steps in achieving the *goal*.

The complete conceptualization of procedural knowledge is a decision list of ordered goal-oriented rules. In an ordered set of rules, the first rule that "triggers" is applied. Note that there is an important difference between semantics of goal-oriented rules and classical if-then rules. An if-then rule triggers for a particular state if the preconditions are true, while a goal-based rule triggers only if the goal is actually achievable (we can execute the goal in the selected state). In other words, even if all preconditions are true for a position, it is not necessary that this rule will *cover* the position. A goal-oriented rule R covers a state *s* if: (a) the preconditions of rule *R* are true for *s*, and (b) goal of *R* is achievable in *s*. For illustration, consider the following example rule:

IF edist > 1 THEN decrease kdist

The correct interpretation of this rule is: "if black king's *distance from the edge is larger than 1* and *a decrease in distance between kings is possible*, then reach this goal: *decrease the distance between kings*."

3.1 Evaluation of Goals and Rules

If a goal is achievable, we would like to know how good it is in a given state. We evaluate the goal by its worst possible realization in terms of distance-to-goal of the final state in the search tree. Formally, a goal's quality *q(g,s)* in state *s* is defined as the difference between starting distance-to-goal and distance-to-goal in the worst realization of the goal: $q(g,s)=dtg(s_{worst})-dtg(s)$. We say that a goal is *good* for a state *s* if its worst realization reduces the distance to solution, i.e. if $q(g,s)<0$; otherwise the goal is *bad*.

[1] If optimal solution cannot be obtained, distance-to-goal values can be replaced by heuristic estimations.

The quality of a rule *R* is directly related to the quality of its goal on states covered by the rule. Let *p* be the number of covered examples where the goal is *good* and *n* number of all covered examples. Then, the quality is computed using the Laplacian rule of succession: $q(R) = (p+1) / (n+2)$.

3.2 Reaching different levels of conceptualization

There are two potential parameters that could be used by a teacher to tailor the conceptualization of procedural knowledge to students with different skills. First, the *preconditions* can have a different number of conditions. If the rules have many conditions, then we need more rules to conceptualize the complete domain, since each rule would cover a smaller part of the feature space. On the other hand, less conditions would lead to less rules, however with goals that are more general (require more search). Secondly, a high value of the *depth* property would again lead to more general rules (in extreme case to only one rule: IF true THEN find goal state), while a smaller value for search depth would have exactly the opposite consequence. For example, setting the *depth* value to a minimum (1) is equivalent to learning simple rules predicting an action to be executed. As both parameters can be used to realize the same effect, we shall use only *depth* in the learning algorithm.

4 The Goal-Oriented Rule Learning Algorithm

The task of learning goal-oriented rules can be stated as: given a set of problem solving states each labeled with a distance-to-goal, learn an ordered set of goal-oriented rules. Since the problem solving states act as learning examples, we will use the latter term in the description of the algorithm. As mentioned above, each learning example is described with a set of attributes.

The pseudo code of our goal-oriented rule learning method is shown in Algorithm 1. It accepts two parameters; *ES* are the learning examples and *depth* is the maximum allowed search depth for achieving goals. The learning loop starts by selecting a seed example, which is afterwards used in the following calls to procedures DiscoverGoals and LearnRule. The DiscoverGoals procedure finds *good* goals for the seed example and then LearnRule induces one rule for each possible goal and returns the most appropriate rule[2]. The idea of seed examples and learning rules from them was adopted from the AQ series of rule-learners developed by Michalski [3], and is especially useful here, since discovering a goal is a time consuming step. A learned rule is afterwards added to the list of all rules *allRules* and all examples covered by this rule are removed from the learning examples. The loop is stopped when all learning examples have been covered.

---

[2] The selection of the most appropriate rule will be explained later within the explanation of *LearnRule* procedure.

Algorithm 1: Pseudo code of the goal-oriented rule learning method.

GOAL-ORIENTED RULE LEARNING (Examples *ES*,*depth*)
1. **Let** *allRules* be an empty list.
2. **While** *ES* is not empty:
3. **Let** *seedExample* be FindBestSeed(*ES*, *ruleList*)
4. **Let** *goals* be DiscoverGoals(*ES*, *seedExample*, *ruleList, depth*)
5. **If** *goals* is empty:
6. Remove *seedExample* from *ES* and return to step 3.
7. **Let** *rule* be LearnRule(*ES, goals, ruleList*)
8. **Add** *rule* to *allRules*.
9. **Remove** examples from *ES* covered by *rule*.
10. **Return** *allRules*.

The FindBestSeed procedure selects as the seed example the one with the lowest distance-to-goal in the remaining examples. Since we speculate that important goals come into place in later stages of the solution – states with lower distance-to-goal, we prefer to learn those goals first. The DiscoverGoals procedure searches for best goals in a given example. It starts with an empty goal and sequentially adds subgoals until we find a *good* goal. If there are several good goals having the same number of subgoals, then the method returns all good goals. The LearnRule procedure first creates for each provided goal a new data set containing all examples from *ES*, where this goal is achievable. Each example in the new data set is labeled as either a *good* goal or as a *bad* goal. Afterwards, LearnRule procedure learns a single rule from each data set and selects the best among them. Its preconditions separate examples where a goal is *good* from those where it is not. We use the CN2 algorithm [1] to learn one rule only, where its conditions must cover the *seedExample*.

5 Argument Based Machine Learning

Due to high complexity of problem solving domains, the automatic learning procedure in the previous section usually extracts only partially correct knowledge. Here we will describe an approach based on argument based machine learning (ABML) [4] where teachers can use their domain knowledge to help the learning algorithm. ABML is machine learning extended with certain concepts from argumentation. Arguments in ABML are a way to enable domain experts (in our case teachers) to provide their prior knowledge about a specific learning example that seems relevant for this case.

An argument is provided by a teacher and is always given to a single example. We call such examples *argumented examples*. The structure of an argument depends on the learning problem. In goal-oriented rule learning, we defined the structure of an argument as: "*argGoal* because *argConditions*", which means that the goal *argGoal* is *good* in the selected example, because the conditions *argConditions* hold. For example, a concrete argument given by a teacher to a specific chess position could be: "In this position, a good goal is to decrease the distance between kings, because black king is sufficiently constrained". As it will be demonstrated in the case study, an argument often contains concepts that are not yet present in the attributes. These concepts represent new needed

declarative knowledge and should thus be added as new attributes in the domain. It is important to note that the teacher is able to control the level of conceptualization with his selection of these very attributes. He can choose to explain the example either with more high-level or low-level concepts.

An ABML method should induce such a hypothesis that considers given arguments. To this end, we implemented the following changes in the algorithm:

- If *seedExample* has an argument, then DiscoverGoal starts with *argGoal*.
- If *seedExample* does not have an argument, then DiscoverGoal first tries to apply any of the goals from arguments given to other examples. If none of the goals is good, then it searches for a goal in the same way as the original algorithm.
- If *seedState* has an argument, the LearnRule procedure uses ABCN2 [4] instead of the classical CN2.

The ABCN2 method used in the LearnRule procedure is an argument-based extension of the CN2 algorithm [1] that learns a set of unordered probabilistic rules from argumented examples. In ABCN2, the theory (a set of rules) is said to explain the examples using given arguments, when there exists at least one rule for each argumented example that contains at least one positive argument in the condition part. Since the method here learns only a single rule at a time, the requirement is that the final rule needs to contain *argConditions* in its preconditions.

5.1 ABML Refinement Loop: Selection of Critical Examples

In argument-based approach, teachers provide their domain knowledge in the form of arguments for some of the learning examples. Asking them to give arguments to the whole learning set is not likely to be feasible, because it would require too much time and effort. It is also not necessary. The following loop describes the skeleton of the procedure that picks out critical examples that should improve the quality of learned rules the most:

A. Learn a set of goal-oriented rules.
B. Find the most critical example and present it to the teacher.
C. The teacher explains the example; the explanation is encoded in arguments and attached to the learning example.
D. Return to step A if critical state was found.

With respect to the second step (B) in the loop, the most critical example is the example with lowest distance-to-goal where current rules suggest a *bad* goal. The explanation of a critical example (step C) contains the following six steps:

1. The teacher is first asked whether the goal suggested by the current model is admissible from the teaching perspective or not. If it is not, we proceed to step 2, otherwise we provide the expert with another critical example. Note that by definition rules assign a *bad* goal to a critical example, but the goal can still be admissible from the teacher's point of view.

2. The teacher provides an argument “goal because reasons”. This argument is then added to the critical example.
3. Method classifies the goal in the argument into *bad*, *good*, or *unachievable*. If the goal is *unachievable*, the teacher is asked to change the argument (back to step 1). If the goal is *good*, we proceed to step 4. If the goal is *bad*, then the method shows the teacher the critical sequence of actions that achieves the goal and at the same time increases distance-to-goal. The teacher can accept the sequence of (continue to step 4), or returns to step 2 and improves the initial goal.
4. The learning algorithm learns a rule “IF preconditions THEN goal”, where the critical example is used as a *seedExample*. Then we identify counter examples, which are examples covered by this rule, where the learned goal is classified as *bad*. If there are no such positions, we move to step 6. Otherwise, the counter example *e* with highest *q(g,e)* is denoted as the *key counter example*. This example is presented to the teacher.
5. If a key counter example is found, the teacher can act in three possible ways: (a) if he thinks that the goal in the key counter example is admissible, then we automatically accept also all other counter examples and move to step 6, (b) if the goal in the key counter example is not acceptable, we return to step 2 where the teacher can improve the initial argument, (c) if the goal in the key counter example is not acceptable, but the teacher cannot improve the initial argument, then he can request the next counter example. Step 4 is repeated.
6. We show the learned rule to the teacher. If the rule seems good, we conclude the explanation process of the example. If there is a condition in preconditions of the rule that teacher finds irrelevant or even wrong, then we can prevent adding this condition to the preconditions of the rule, when this example is used as a *seedState*. Return to Step 2.

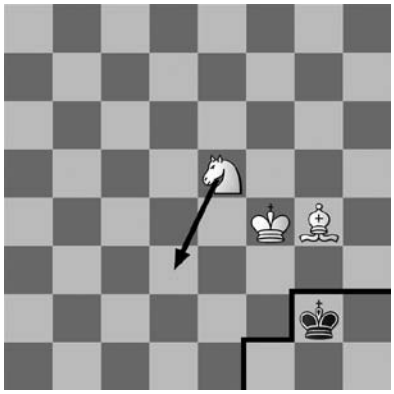
6 Case Study

As a case study, we considered the conceptualization of procedural knowledge required to deliver checkmate in the KBNK chess endgame. KBNK (king, bishop, and knight vs. a lone king) is regarded as the most difficult of the elementary chess endgames. Most books mention only a basic strategy, however, it is hardly enough for successfully checkmating the opponent.

We used our algorithm to derive the instructions in the form of goals for delivering checkmate from any given KBNK position. These instructions were semi-automatically derived using an interactive procedure between a chess teacher (a FIDE master of chess) and the computer, which also had chess tablebases (perfect information in terms of distance to mate in any position), at its disposal. The result of this procedure was an ordered set of eleven rules, which was presented in more detail in [2].

The aim was to tailor the instruction level for students at club level. Our chess teacher evaluated that the skill level of targeted students should be sufficient for them to be able

to calculate chess variations at least three moves (6 plies) ahead. This depth of lookahead was therefore set for the depth parameter of our algorithm.



Computer: "I suggest the following goal: the distance between black king and the edge of the board should decrease. However, it does not seem to work well in this position. *What goal would you suggest for white in this position? What are the reasons for this goal to apply in this position?"*

The teacher gave the following answer: "Pushing black king to the edge of the board is fine. However, I find the following goal to be more instructive for the student: *Build a barrier and squeeze the defending king into the corner. Currently such barrier is not yet established.* The move expected from the student is 1.Ne5-d3 achieving the goal."

Figure 1: Interaction between computer and teacher: explanation of a critical example.

In this section, we will focus on some important points behind the interactive procedure between the teacher and the machine-learning algorithm, with respect to the appropriateness of our approach for conceptualizing procedural knowledge in a given task.

The teacher and the machine learning algorithm improve the model iteratively. A typical interaction between the method and the expert is shown in Fig. 1. Our algorithm is capable of inducing goals that are both achievable and successful in terms of progressing towards delivering checkmate by itself. However, in the critical example in Fig. 2 the automatically induced goal ("distance of black king from the edge of the board should decrease") was characterized by the algorithm as *bad* (*e.g.* after 1. Bg4-f3 Kg2-h3 the goal is achieved, but distance-to-mate increases). The teacher was asked whether the goal suggested was admissible or else to provide an argument explaining this position (step 1 of the algorithm for explaining a critical example).

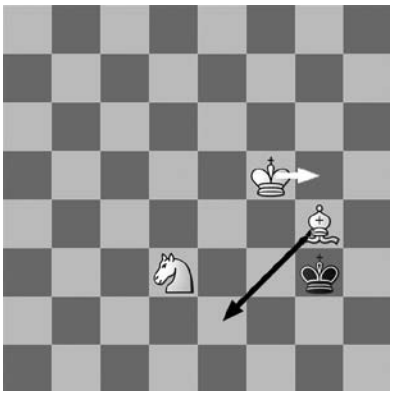
As a result of the new argument given to this position (step 2) a new attribute, `king_area`, was introduced. New attributes allow teachers to introduce important elements of knowledge to be acquired by students. In the present case, the student is advised to build a barrier that holds the defending king in an area beside the corner. When such barrier is built, the student should aim to squeeze the constrained area in order to further restrain the defending king. The comment from the teacher was translated into the following argument in computer-understandable form:

```
decrease king_area AND minimize king_area BECAUSE king_area is high
```

Important to note here is that "king_area" is a new concept introduced by the teacher which he used for explaining his view. The teacher is free to use as high-level or low-level concepts as he wishes his students to be presented with. In short, this is another way for teachers to tailor the level of conceptualization.

Appropriate translation of the teacher's comment into the language understandable to the algorithm is mandatory. For example, this goal was classified as *good* (step 3), as the student has to squeeze the area available to the defending king as much as possible. Without this particular subgoal, another move would be acceptable apart from the one that the teacher mentioned: 1.Ne5-c4, which is worse, since it provides more freedom to the defending king. Note that such optimizations are often very logical to humans, however, not to computers, which demand precise specifications.

Often, counter examples are detected by the method, and presented to the teacher. Counter examples are positions where the goal can be achieved, but the resulting play nevertheless leads to an increased distance to mate. Among such positions, the one with the highest distance to mate is chosen as the key counter example (step 4) Figure 2 illustrates.



Computer: “Would you admonish a student if he or she played 1.Bg4-e2 in this position?”

The teacher would not admonish a student, he found this move to be perfectly acceptable. Despite of its non-optimality: from the tablebase point of view 1.Kf5-g5 is a better move - 1.Bg4-e2 (the worst possible execution of the goal of squeezing the area available to the black king) achieves mate in 10 moves whereas after 1.Kf5-g5 only 8 moves are necessary.

Figure 2: Interaction between computer and teacher: explanation of the key counter example.

Allowing non-perfect execution of tasks enables acquiring procedural knowledge in a more human-like manner. In chess, human players typically do not play optimally in the sense of shortest win against best defense; they choose a longer path to win by systematically achieving intermediate goals. The teacher would not admonish students if resulting play in counter examples leads to overall progress towards achieving the final goal (step 5a). The way of squeezing the area available to the black king in the counter example in Fig. 2, although not optimal, was judged to lead to such progress.

The expert may also find the execution of the goal in a counter example to be unacceptable. In this case, he or she may add, modify, and/or remove any of the preconditions and subgoals in the triggered rule, according to his knowledge about the domain. Automatic detection of relevant counter examples represents an important mean of preventing mistakes or possible misconceptions in the process of conceptualization of knowledge, and contributes effectively to obtaining models that are in accordance with the teacher's knowledge.

Let us briefly report on the success of the learning. The compiled rules were aimed to be used as textbook instructions for teaching the KBNK endgame. As such they were presented to three chess teachers (among them a selector of Slovenian women's squad and a selector of Slovenian youth squad) to evaluate their appropriateness for teaching

chess-players. They all agreed on the usefulness of the presented concepts and found the derived strategy suitable for educational purposes at the level targeted for. Among the reasons to support this assessment was that the instructions “clearly demonstrate the intermediate subgoals of delivering checkmate.” [2]

7 Conclusions

We presented a novel algorithm for (semi)automated conceptualization of procedural knowledge based on goal-oriented rules in symbolic domains. We argued that apart from easier conceptualization the algorithm also facilitates the teacher to regulate the level of conceptualization as dictated by the level of skills of students targeted at. This is accomplished in a very natural way by: (a) setting a single parameter of the algorithm, and (b) by allowing the teacher to use/add his own concepts as he sees fit when explaining the more difficult examples.

The main target of the future work is a more thorough evaluation, both on chess and other domains. The effect of the search depth parameter should be assessed more precisely.

References

[1] Boswell, R., Clark, P. Rule induction with CN2: Some recent improvements. *Proceedings of the Sixth European Working Session on Learning*, 1991, pp.151-163, Porto, Portugal.

[2] Guid, M., Možina, M., Sadikov A., Bratko, I. Deriving Concepts and Strategies from Chess Tablebases. *Advances in Computers and Games conference (ACG 12),* Pamplona, Spain, May 11-13, 2009.

[3] Michalski, R. S., A Theory and Methodology of Inductive Learning. In R. Michalski, J. Carbonnel, T.Mitchell (editors): *Machine Learning: An Artificial Intelligence Approach*, 1983, pp. 83-134, Kaufman, Tioga, Palo Alto, CA.

[4] Možina, M., Žabkar, J., Bratko, I. Argument based machine learning. *Artificial Intelligence*, 2007, 171(10-15), pp. 922-937.

[5] Murray, T. Authoring Intelligent Tutoring Systems: An analysis of the state of the art. *Int. Journal of Artificial Intelligence in Education* , 1999, *10*, pp. 98-129.

[6] Tadepalli, P. Learning to Solve Problems from Exercises. *Computational Intelligence*, 2008, 24(4), pp. 257-291.

[7] Woolf, B. Building intelligent interactive tutors, 2008, Morgan Kaufman.