Cognitive Tutoring in Mathematics based on Assertion Level Reasoning and Proof Strategies

Serge Autexier Dominik Dietrich Marvin Schiller
German Research Centre for Artificial Intelligence (DFKI) Brunel University
autexier@dfki.de dominik.dietrich@dfki.de Marvin.Schiller@brunel.ac.uk

To know how to do proofs is a skill that is essential for every mathematician and scientist. Hence, learning how to do proofs is a major part in the education of students of mathematics and modern science. Intelligent tutoring systems (ITS) are an attractive vehicle to make high-quality teaching and training environments available for a wide public. There are two main approaches: Model tracing tutors (MTTs), which are process-centric and try to fathom the process a student arrived at a solution, and constraint based tutors (CBTs), which are product-centric and are based on the idea that diagnostic information is not in the sequence of actions leading to the problem state, but solely in the problem state itself (see [11, 10] for a comparison). According to [10], the main advantage of a CBT is that its design requires less time and effort (as it does not need to solve the problem itself), while a MTT offers more specific advice giving capabilities (as the solution building process is explicit).

There exist already strong tools for teaching mathematical computations (such as [9]) or pure logic (such as the CMU proof tutor [14] or Proofweb [8]), however there are only a few attempts that support teaching of mathematics at a more abstract level, comparable to the type of mathematics taught in schools or in the first year at university. The main reasons we identified for this are:

Automated Proof Search Limitations: Domain reasoning quickly becomes too difficult for existing reasoners. However, MTT require a domain reasoner to determine how a student arrived at a certain step.

Solution Space Size: A theorem may have different kinds of proofs, each of which can be formulated in slightly different variants. Thus, the ITS technique of preauthoring solutions (including problem- and situation-specific hints) is unsuitable – if not unfeasible – for teaching mathematical theorem proving.

Proof Search vs. Proof Presentation: Finding a proof for a conjecture is different from presenting the found proof to the user: for the final presentation, backward steps are often converted to forward steps, several steps are combined to obtain a shorter proof, and the names of the employed proof techniques are omitted (see [15] p. 13-15 for a discussion). Therefore, existing proofs can only be used as target for tutoring to a very limited extent.

Proof Step Granularity: Human tutors may reject a proof step even though it is logically correct as it may lack other desirable properties, for instance if it is of inappropriate step size or irrelevant with respect to a specific proof strategy to be taught. However, classical reasoners usually provide large proof objects based on some particular logical calculus, such as resolution, which make it difficult to judge the general appropriateness of a proof step proposed by a learner in a tutorial context or to synthesize a hint if the student requests help.

Human-style vs. Machine-style Proofs: Proof techniques used in standard automated theorem proving (ATP) are different from techniques used by humans to solve problems – it is unreasonable to expect the solutions generated by standard ATP systems to reflect these techniques. Therefore, inspecting the proof object makes it difficult to analyze the solution with respect to other properties than correctness. Moreover, the complexity of the generated proof objects make it difficult to extract suitable hints at a strategic level.

Submitted to:
THEdu 2011

Irrelevant/Inexpedient Proof States: Classifying (incorrect) problem states by constraints as done in CBT seems to be difficult in the domain of proof tutoring.

Therefore, we advocate a dynamic approach to support the computer assisted teaching of mathematical proofs in the spirit of MTT. In previous work, we have shown that the so-called *assertion level* (see [7]) provides a suitable basis to verify underspecified and incomplete human-level proof steps in the domain of set theory based on a proof reconstruction approach (see [5]). In contrast to machine oriented calculi, each proof step at the assertion level corresponds to the application of a theorem, definition, or axiom. Our assertion level prover systematically converts declarative knowledge (formulas) to procedural knowledge (inferences), a process that usually needs to be done manually in the context of MTT. The contribution of this paper is to give a concise summary of our work in the context of computer theorem proving, highlighting the techniques from the theorem proving community that are used. These are: The different processing models of the prover, which actively processes declarative proof commands entered by a student in a tutoring setting, and lazily processes proof commands in a pure verification setting (Section 1); specification of declarative domain knowledge in theory languages, the use of a declarative proof language to encode student dialogue turns, specification of strategic problem solving knowledge as well as verification techniques in proof strategies/tactics (see [6, 2] for an overview) in Section 2; and representation of proof reconstructions in a hierarchical proof data structure to analyze the student's input and to offer hints at different levels of detail (see [1] for details) in Section 3.

1 Proof Command Processing Models

In a tutoring context, the student stepwise enters proof steps to solve a given tutorial exercise in the domain of set theory. Proof steps are formulated as proof commands in a declarative proof language. However, the processing of a proof command by the proof assistant within the setting of tutorial dialogues differs from the processing in a pure verification setting with respect to the following points:

- In a pure verification setting, it is sufficient to find some verification for a proof command. The verification itself is usually not of interest and needs not to be further processed. In contrast, in a tutorial setting we need to consider several, if not all, possible verifications of the given proof command (and relate them to the knowledge of the student) and need to further analyze the verification to avoid the student to rely on the power of the underlying theorem prover to solve the exercise.
- In a pure verification setting, we can assume the user to be an expert in the problem domain as well as in the field of formal reasoning. This has several implications on the processing model: (i) inputs can be expected to be correct and just need to be checked, (ii) proof commands can lazily be verified until a (sub)proof is completed, (iii) justification hints are given that indicate how to verify a given proof command, (iv) feedback is limited to "checkable" or "not checkable". In contrast, in a tutorial setting, we must assume the user to be neither a domain expert nor an expert in formal reasoning. The underlying mechanisms need to be hidden from the user, direct and comprehensive feedback has to be provided at each step. Therefore, it is for example a requirement to anticipate why an assumption is made, in contrast to a lazy checking once the conclusion has been obtained.
- In a pure verification setting, we can assume the user to indicate when the proof of a subgoal is finished (as usually done by so-called *proof step markers* in the proof language). However, in the tutorial setting this information is implicit. Similarly, we must be able to perform backward steps where some of the new proof obligations have not yet been shown.

```
strategy work-backward
  repeat
  use select * from definitions as
backward

strategy close-by-logic
  repeat first deepaxiom, or-l
```

```
strategy work-forward
  repeat
  use select * from definitions as forward

strategy close-by-definition
  try work-backward then try work-forward
then close-by-definition
```

Figure 1: Formalization of a simple proof strategy

2 Representation of Proof Strategies

A *proof strategy* represents some mathematical technique that happens to be typical for a given problem. For example, there are strategies which perform proof by induction, proof by contradiction, solve equations, or unfold definitions. To achieve a goal, a strategy performs a heuristically guided search using a dynamic set of assertions, as well as other strategies. Proof strategies are encoded in a strategy language (see [6, 2] for an overview) and generate hierarchical *strategic proof plans* (see [1] for details), where each justification in a plan corresponds to a strategy or an assertion application. The hierarchies arise from invocations of strategies or assertion applications from inside strategies. Intuitively, a hierarchical proof plan at a high level sketches how the overall problem was structured into subproblems at a meta level. At the lowest level, a concrete proof with concrete assertion steps is given.

A simple proof strategy that is proposed in [15] is the "Forward-Backward Method", which consists of first applying definitions to the conclusion of a goal to simplify it, and then using forward reasoning on the assumptions to derive new facts and finally close the goal. We have formalized this method within our strategy language within the strategy "Close-by-Definition" which relies on three sub-strategies, "Work-Forward", "Work-Backward" and "Close-by-Logic". "Work-Forward" works on the assumptions of the current task and mainly expands definitions. "Work-Backward" tries to simplify the current goal by applying definitions. "Close-by-Logic" applies logical reasoning, such as performing case splits, to close the task it was applied to. Figure 1 shows the formalization of the methods.

3 Adaptive Generation of Hints

The basic idea to dynamically generate context sensitive hints for a specific proof task is simple: Complete the proof for the current proof task using a proof strategy (such as close by definition) and analyze the solution to extract and generate a suitable hint. Thereby, make use of the proof hierarchies to increase the detailedness of hints on demand. More precisely, the generation of a hint works as follows: (i) selecting a certain level of hierarchy in the reconstruction, (ii) selecting a sequence of steps starting from the task with which the completion of the proof was invoked, (iii) extracting information from this sequence and converting it to a concrete hint. A given hint can be refined by either switching to a more detailed level of hierarchy, increasing the length from which the hint was generated, or by increasing the information which was extracted from the selected sequence. Let us stress here that this simple approach is only made possible by our abstract proof presentation at the assertion level and would be much more complicated if not impossible within natural deduction or in a resolution calculus.

To illustrate how such a strategy can be used to generate a context-sensitive hint, consider the exercise $(R \circ S)^{-1} = S^{-1} \circ R^{-1}$. Suppose that the student starts the proof by applying the definition of set equality,

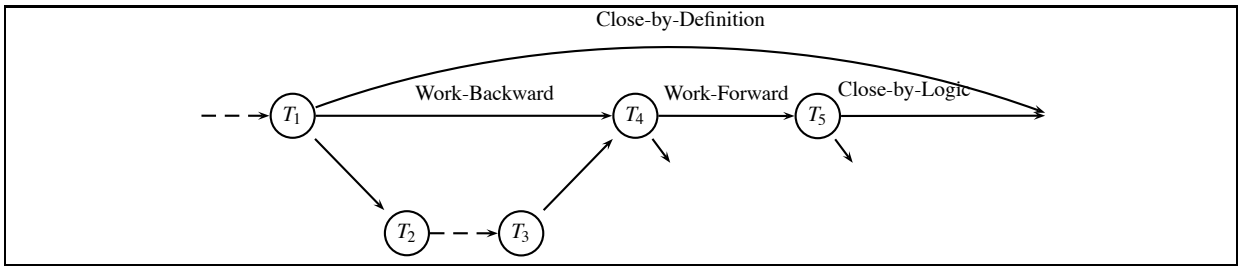

Figure 2: Hierarchical proof plan completing the proof of task T_1

yielding two subtasks

$$T_1 : \vdash (R \circ S)^{-1} \subset S^{-1} \circ R^{-1} \tag{1}$$

$$T_1' : \vdash S^{-1} \circ R^{-1} \subset (R \circ S)^{-1} \tag{2}$$

and requests a hint for the task T_1. A possible completion of the proof, encoded in the strategy "Close-by-Definition", consists of expanding all definitions and then using logical reasoning to complete the proof. The resulting hierarchical proof object is shown schematically in Figure 2. The task T_1 has three outgoing edges, the topmost two corresponding to a strategy application and the lower-most one corresponding to an assertion application, respectively. Internally the edges are ordered with respect to their granularity, according to the hierarchy of nested strategy applications that generated them. In the example the most abstract outgoing edge of T_1 is the edge labelled with "Close-by-Definition", followed by the edge labelled with "Work-Backward", both representing strategy applications. The edge with the most fine-grained granularity is the edge labelled with "Def $\subset$" and represents an inference application.

By selecting the edges "Work-Backward", "Work-Forward", and "Close-by-Logic", we obtain a flat graph connecting the nodes T_1, T_4, and T_5. A more detailed proof-view can be obtained by selecting the edge "Def $\subset$" instead of "Work-Backward". In this case the previous single step leading from T_1 to T_4 is replaced by the subgraph traversing T_2 and T_3. Each selection can be used to generate several hints. Suppose for example that we select the lowest level of granularity, and the first proof state to extract a hint. This already allows the generation of three hints, such as

- "Try to apply Def $\subset$"
- "Try to apply Def $\subset$ on $(R \cup S) \circ T \subset (T^{-1} \circ S^{-1})^{-1} \cup (T^{-1} \circ R^{-1})^{-1}$"
- "By the application of Def $\subset$ we obtain the new goal $(x,y) \in (R \cup S) \circ T \Rightarrow (x,y) \in (T^{-1} \circ S^{-1})^{-1} \cup (T^{-1} \circ R^{-1})^{-1}$"

Selecting a more abstract level would result in hints like "Try to work backward from the goal", or "Try to apply definitions on the goal and assumptions". Within our approach, the analysis functions to extract hints and their verbalization are hard-coded within the programming language.

4 Evaluation

The presented techniques have been successfully evaluated in experiments with computer-based tutoring of proofs in naive set theory [3]. In [12], we have shown that reconstructions at the assertion level can be used for a proof step analysis that goes beyond the correctness of proof steps, such as the analysis of proof granularity, i.e. the question whether a given proof step is of appropriate size. Moreover, in [13]

we have advocated that assertion-level proof steps provide a good level of abstraction to approximate and judge the proof steps done by students in the naive set theory domain.

References

[1] Serge Autexier, Christoph Benzmüller, Dominik Dietrich, Andreas Meier & Claus-Peter Wirth (2006): *A Generic Modular Data Structure for Proof Attempts Alternating on Ideas and Granularity*. In: *Proc. of MKM*, Springer, Bremen, pp. 126–142.

[2] Serge Autexier & Dominik Dietrich (2010): *A Tactic Language for Declarative Proofs*. In Matt Kaufmann & Lawrence C. Paulson, editors: *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, *Lecture Notes in Computer Science* 6172, Springer, pp. 99–114. Available at `http://dx.doi.org/10.1007/978-3-642-14052-5_9`.

[3] Christoph Benzmüller, Helmut Horacek, Henri Lesourd, Ivana Kruijff-Korbayova, Marvin Schiller & Magdalena Wolska (2006): *A corpus of tutorial dialogs on theorem proving; the influence of the presentation of the study-material*. In: *Proceedings of International Conference on Language Resources and Evaluation (LREC 2006)*, ELDA, Genova, Italy, pp. 1766–1769.

[4] Peter Brusilovsky, Albert T. Corbett & Fiorella de Rosis, editors (2003): *User Modeling 2003, 9th International Conference, UM 2003, Johnstown, PA, USA, June 22-26, 2003, Proceedings*. *Lecture Notes in Computer Science* 2702, Springer.

[5] Dominik Dietrich & Mark Buckley (2008): *Verification of Human-level Proof Steps in Mathematics Education*. *Teaching Mathematics and Computer Science* 6(2), pp. 345–362.

[6] Dominik Dietrich & Ewaryst Schulz (2009): *Integrating Structured Queries into a Tactic Language*. *JAL - Special issue on Programming Languages and Mechanized Mathematics Systems* .

[7] Xiaorong Huang (1994): *Reconstructing Proofs at the Assertion Level*. In Alan Bundy, editor: *Proc. 12th CADE*, Springer-Verlag, pp. 738–752. Available at `citeseer.ist.psu.edu/huang94reconstructing.html`.

[8] Cezary Kaliszyk, Freek Wiedijk, Maxim Hendriks & Femke van Raamsdonk (2007): *Teaching logic using a state-of-the-art proof assistant*. In H. Geuvers & P. Courtieu, editors: *Proc. of the International Workshop on Proof Assistants and Types in Education*, pp. 33–48.

[9] Erica Melis, Eric Andrès, Jochen Büdenberger, Adrian Frischauf, George Goguadze, Paul Libbrecht, Martin Pollet & Carsten Ullrich (2001): *ActiveMath: A Generic and Adaptive Web-Based Learning Environment*. *Artifical Intelligence in Education* 12(4).

[10] Antonija Mitrovic, Kenneth R. Koedinger & Brent Martin (2003): *A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling*. In Brusilovsky et al. [4], pp. 313–322. Available at `http://link.springer.de/link/service/series/0558/bibs/2702/27020313.htm`.

[11] Antonija Mitrovic & Stellan Ohlsson (2006): *A Critique of Kodaganallur, Weitz and Rosenthal, "A Comparison of Model-Tracing and Constraint-Based Intelligent Tutoring Paradigms"*. *I. J. Artificial Intelligence in Education* 16(3), pp. 277–289. Available at `http://iospress.metapress.com/content/1rlwf2k7qdr1ag4l/`.

[12] Marvin Schiller (2010): *Granularity Analysis for Tutoring Mathematical Proofs*. Ph.D. thesis, Saarland University.

[13] Marvin Schiller & Christoph Benzmüller (2010): *Human-Oriented Proof Techniques are Relevant for Proof Tutoring*. Extended Abstract for MIPS 2010 workshop (affiliated with CICM 2010). CNAM, Paris, France.

[14] Wilfried Sieg & Richard Scheines (1994): *Computer Environments for Proof Construction*. *Interactive Learning Environments* 4(2), pp. 159–169.

[15] Daniel Solow (2005): *How to read and do proofs*. John Wiley and Sons.