

Verification of Proof Steps for Tutoring Mathematical Proofs¹

Dominik DIETRICH^a and Mark BUCKLEY^b

^a *Dept. of Computer Science*

^b *Dept. of Computational Linguistics*

Saarland University, Postfach 151150, 66041 Saarbrücken, Germany

dietrich@ags.uni-sb.de buckley@coli.uni-sb.de

1. Motivation

The feedback given by human tutors is strongly based on their evaluation of the correctness of student’s contributions [7]. A typical approach to verification of contributions in ITSs is model tracing, in which contributions are matched against a precomputed solution graph [1,3]. However verifying steps in mathematical theorem proving poses particular problems because there are possibly infinitely many acceptable proofs, steps may be only partially ordered, and the student should be free to build any correct solution. Tracing contributions against static solution graphs is therefore overly restrictive for exercises in typically sized mathematical theories.

We propose a flexible domain-independent method of verifying proof steps of varying lengths on-the-fly for a mathematics tutoring system. Our approach can verify steps in unforeseen proofs and incrementally builds a solution state for each possible interpretation of underspecified or ambiguous steps. We utilise an existing mathematical reasoner Ω MEGA [8] and existing mathematical knowledge. Our approach to the verification of proof steps is motivated by our corpus of Wizard-of-Oz tutorial dialogues between students and experienced mathematics teachers [4]. In a proof of the theorem $(R \cup S) \circ T = (R \circ T) \cup (S \circ T)$, the student’s first proof step “Let $(x, y) \in (R \cup S) \circ T$ ” consists of two definitions: set extensionality and subset. The tutor responds with “Correct!”. This shows the tutor must know the correctness of steps to apply pedagogical strategies and that proof steps can contain applications of many mathematical definitions.

2. The Ω MEGA Prover

The development of the proof assistant system Ω MEGA is one of the major attempts to build an all-encompassing assistance tool for the working mathematician or for the formal work of a software engineer. It is a representative of systems in the paradigm of *proof planning* and combines interactive and automated proof construction for domains with rich and well-structured mathematical knowledge.

¹Supported by the Collaborative Research Centre on *Resource-Adaptive Cognitive Processes*, SFB 378.

In Ω MEGA, proofs are represented at the *tasklayer*, which is an instance of the proof data structure (PDS) [2]. Each proof attempt is represented by an agenda, which maintains a set of subproblems to be solved to finish the proof of the overall conjecture. The nodes of the PDS are annotated with *tasks*, which are Gentzen-style multi-conclusion sequents augmented by a means to define multiple foci of attention on subformulas. Proof search is realised by reducing a task to a possibly empty set of subtasks by transformation rules which can be applied to subformulas. The basic transformation rules are *inferences*, representing the application of theorems, definitions, and axioms.

3. Representing and Updating the Student’s Cognitive Proof State

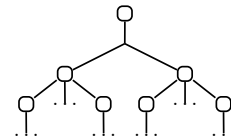
We now show how the Ω MEGA-prover can be used to represent possible cognitive states the student might be in, and to judge about the correctness of a proof step of the student.

Representing the Possible Cognitive Proof States The PDS is used to simultaneously represent all of the possible cognitive proof states the student might be in, where each agenda corresponds to one possible cognitive proof state. In addition, the subproblem the student is currently working on is marked. Given a problem and a theory, an *initial PDS* is constructed, containing the problem the student has to solve. The *initial mental state* is represented by the agenda containing the problem.

Updating the Cognitive Proof States Given a set of possible cognitive proof states and a preprocessed utterance, we must determine all possible successor states which are consistent with the utterance. For the sake of simplicity we only show how to determine the successor states of a single cognitive proof state. Combining the result for each state gives the complete set of successor states. If no agenda can provide a consistent successor state then we conclude that the step is incorrect.

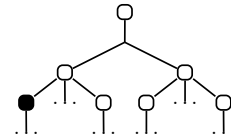
Given a classified utterance its successor agendas are determined in four steps: (i) Performing a depth limited BFS search, resulting in a set of successor nodes, (ii) selecting consistent successor nodes and creating partial agendas out of them, (iii) completing the partial agendas, (iv) cleaning the PDS. The overall result is a confirmation of whether the step could be verified, with the side-effect that the PDS has been updated to contain exactly the possible cognitive proof states resulting from the performance of the step.

Step (i) The node representing the problem the student is currently working on is expanded using a depth limited BFS search. The depth limiter specifies how many calculus steps the student is allowed to perform implicitly. Intuitively we can think of this bound as reflecting the experience of the student, and should be based on a student model. This bound is needed to guarantee termination of the verification algorithm, which might otherwise not terminate for a faulty step even if we assume a complete calculus. The correspondence of student steps to calculus steps may vary for each calculus. Tests show that 3 is a good choice for our domain.



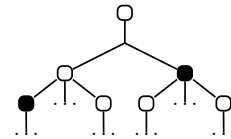
In our example the initial agenda contains the task $\vdash (R \cup S) \circ T = (R \circ T) \cup (S \circ T)$. The search tree is shown schematically on the right. To verify the student’s proof step the node containing this task is expanded using the axioms from the theory, whereby each task is successively reduced to a list of subtasks.

Step (ii) We suppose that for each class of proof step there is a filter which chooses exactly those successor states consistent with the student's step. For example, a task is consistent with respect to a step "let f " if it contains the (sub)formula f as an assumption, and the free variables of f have been introduced as new eigenvariables. To further restrict the consistent successor nodes only those with minimal distance to the expansion node are stored.



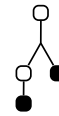
In the example the filter returns exactly one consistent successor node at depth 2, namely the node associated with the task $(x, y) \in (R \cup S) \circ T \vdash (x, y) \in (R \circ T) \cup (S \circ T)$. Indeed, this task justifies the introduction of $(x, y) \in (R \cup S) \circ T$. A new partial agenda is created in which the task is the new selected subgoal.

Step (iii) The agendas obtained from step (ii) can be partial, i.e. not all subgoals that must be solved are in the agenda. Such situations occur if the prover reduces a task to multiple subtasks but the filter does not select a node from each branch. In this case the user has not modified any of the missing subtasks.



Hence it is reasonable to extend the partial agenda by the tasks introduced by the reduction. In the example above we extend the agenda as shown on the right.

Step (iv) Usually there will be many nodes which are generated by the search but are rejected by the filter. All those nodes are removed from the PDS. In our example this results in the PDS as shown. The new cognitive state of the student could be uniquely identified and thus the proof step was correct.



Conclusion and Related Work Using proof search to verify steps within the current proof situation removes the need for precomputed solution graphs, and our approach is thus more flexible than approaches which check contributions by matching them to a precomputed solution graph [3,5,6,9]. It has been implemented in the Ω MEGA system. Future work will include investigating the filters which are applied to proof states and verifying the goal-directedness of steps.

References

- [1] V. Alven, K. Koedinger, and K. Cross. Tutoring Answer Explanation Fosters Learning with Understanding. In *Proc. of AIED*, pages 199–206, Amsterdam, 1999. IOS.
- [2] S. Autexier, C. Benzmüller, D. Dietrich, A. Meier, and C.-P. Wirth. A generic modular data structure for proof attempts alternating on ideas and granularity. In *Proc. of MKM*, Bremen, 2006. Springer.
- [3] Cristina Conati, Abigail Gertner, and Kurt Vanlehn. Using Bayesian Networks to Manage Uncertainty in Student Modeling. *User Modeling and User-Adapted Interaction*, 12(4):371–417, 2002.
- [4] C. Benzmüller *et al.* A corpus of tutorial dialogs on theorem proving; the influence of the presentation of the study-material. In *Proc. of LREC*, Genoa, 2006. ELDA.
- [5] Arthur C. Graesser, Katja Wiemer-Hastings, Peter Wiemer-Hastings, and Roger Kreuz. Autotutor: A simulation of a human tutor. *Cognitive Systems Research*, 1:35–51, 1999.
- [6] N. Heffernan and K. Koedinger. Building a 3rd generation ITS for symbolization: Adding a Tutorial Model with Multiple Tutorial Strategies. In *Proc. of the ITS Workshop in Algebra Learning*, 2000.
- [7] Gregory D. Hume, Joel A. Michael, Rovick A. Allen, and Martha W. Evens. Hinting as a tactic in one-on-one tutoring. *Journal of the Learning Sciences*, 5(1):23–47, 1996.
- [8] Jörg Siekmann, Christoph Benzmüller, and Serge Autexier. Computer Supported Mathematics with Ω MEGA. *Journal of Applied Logic*, 4(4):533–559, 2006.
- [9] C. Zinn. Supporting Tutorial Feedback to Student Help Requests and Errors in Symbolic Differentiation. In M. Ikeda, K. D. Ashley, and T.-W. Chan, editors, *Proc. of ITS*, pages 349–359. Springer, 2006.