Towards an Intelligent Tutor for Mathematical Proofs

Serge Autexier Dominik Dietrich
German Research Center for Artificial Intelligence (DFKI), Bremen, Germany
Serge.Autexier@dfki.de Dominik.Dietrich@dfki.de

Marvin Schiller
Brunel University, London, UK
Marvin.Schiller@brunel.ac.uk

Computer-supported learning is an increasingly important form of study since it allows for independent learning and individualized instruction. In this paper, we discuss a novel approach to developing an intelligent tutoring system for teaching textbook-style mathematical proofs. We characterize the particularities of the domain and discuss common ITS design models. Our approach is motivated by phenomena found in a corpus of tutorial dialogs that were collected in a Wizard-of-Oz experiment. We show how an intelligent tutor for textbook-style mathematical proofs can be built on top of an adapted assertion-level proof assistant by reusing representations and proof search strategies originally developed for automated and interactive theorem proving. The resulting prototype was successfully evaluated on a corpus of tutorial dialogs and yields good results.

1 Introduction

Computer-supported learning is an increasingly important form of study since it allows for independent learning and individualized instruction and has resulted in many tutoring systems for different domains. Mathematics is a key discipline in education and today, there exist strong systems to teach and tutor specific mathematical skills, such as mathematical computations, problem solving and geometry (see for instance, [49, 41, 46, 36, 14, 53, 33, 19] to name a few). However, teaching and tutoring support on the ability of how to do proofs is underdeveloped in state of the art e-learning systems. Notable exceptions are the tutoring systems for geometrical proofs [45, 42, 67], as well as for pure formal logic proofs (such as the CMU proof tutor [61], the NovaNet Proof Tutorial [13] or Proofweb [39]). The overall goal of the work presented in this paper is to provide e-learning support for classical textbook-style proofs, which is not fostered by the above approaches.

Following Van Lehn (see [69]), intelligent tutoring systems (ITSs) can be characterized as having both an outer loop and an inner loop. The outer loop selects a relevant task (exercise) for the student to complete. The inner loop iterates over individual problem-solving steps, evaluates the steps and provides feedback to the student. Typically, this is achieved by providing the following services: (S1) minimal feedback on a step, (S2) error specific feedback on an incorrect step, (S3) hints on the next step, (S4) assessment of knowledge, and (S5) review of the solution.

In this paper we focus on the inner loop and report on how we adapted the proof assistant system Ω*mega* to realize the services (S1), (S2), (S3) and a bit of (S5). The services are provided by two components, one for *step analysis*, and one for *step generation*. The step analyzer produces minimal feedback on a student's proof step, such as whether the step is correct or not, together with additional information that might be used to provide more sophisticated feedback, e.g., if a specific error type could be extracted. The step generator returns information about a step the student should do next, e.g., when a hint is requested by the student.

Individual aspects of this work have been published before and describe different stages of the development of the different components. For instance, parts of the step analyser described in [58] have never

P. Quaresma and R.-J. Back (Eds.); THedu'11
EPTCS 79, 2012, pp. 1–28, doi:10.4204/EPTCS.79.1

been described in connection with the other parts of the final step analyzer or the final hint generation module [29]. The contribution of this paper is to provide a coherent overview of all components from the initial requirements analysis, via the design phase to their final implementation and evaluation.

The paper is organized as follows: Section 2 discusses the aspects of the tutoring problem by analyzing the teaching domain and reports on a Wizard-of-Oz experiment which was part of the design phase of our ITS system. The collected data in the form of tutorial dialogs represents a kind of "gold standard" which we try to approximate. Informed by the analysis of the collected data, we briefly review the state of the art in designing tutoring systems and derive the architecture of our tutoring system. The methods and tools to analyze student's proof steps are presented in Section 3. Section 4 presents how hints with an increasing degree of explicitness are generated from the domain specific proof strategies specified by the domain expert. The intelligent tutor services have been evaluated on the corpus of tutorial dialogs about proofs as presented in Section 5. Section 6 presents an outlook of further qualitative service improvements that are within grasp. We review related work in Section 7 and conclude in Section 8.

2 Requirements Analysis and Functional Specification

The teaching goal in our domain is that students develop the skill to conduct and author mathematical proofs in textbook style. Our target students are in the final high-school years or undergraduate university students. Therefore, the assumptions on the students are that they have a knowledge of specific mathematics domains, but not necessarily of mathematical logic. Textbook-style proofs themselves consist of intermediate proof steps in a declarative style stating a subgoal or a derived fact. In that sense they are compatible to formal logic proofs. In contrast, the justifications in textbook style proofs are typically not exclusively references to basic formal logic rules. Rather, they are justified by references to hypotheses, definitions, lemmas or theorems—collectively called *assertions*—as well as specific proof strategies (e.g., well-founded induction) or symbolic computations (e.g., polynomial factorization)—collectively called *reasoning techniques*. It is common practice to reduce the size of a proof by leaving out references to assertions or reasoning techniques if they can easily be inferred by the reader.

Solution Space. For a given theorem, there is typically a space of possible proofs. This is for a variety of reasons: First, from a mathematical point of view, there often exist various distinct proofs for the same proof problem. For example, Ribenboim gives eleven proofs that there are infinitely many primes (see [55] for details). Especially, given a set of proofs it is difficult to be certain that no further mathematically sensible proof is possible, i.e., that a given set of proofs contains all mathematically sensible proofs.

Secondly, even for a specific proof, there are different ways to formulate the proof: permutability of proof steps is one reason which essentially leaves the proof structure invariant. Additionally, a proof step can be formulated in forward-style by deriving new facts or in backward-style by introducing new subgoals: the choice of forward-style vs. backward-style has a more severe impact on how the proof is structured. Typically, in textbook-style proofs most steps are in forward-style because it makes it easier to follow the proof. However, proof search often happens in backward-style, and the proofs are reformulated in forward-style afterwards (see [62] p. 13-15 for a discussion).

A given proof with a specific proof-style and a particular order of applying assertions is considered to be at the lowest level of mathematical proof, as all information is explicitly provided. However, this is often much too detailed and more high-level versions of the proof are also acceptable or preferred. For instance, one may allow for larger proof steps (w.r.t. step-size) and the omission of those justification details that are considered as trivial. The step-size of proof steps and the detailedness of their justifica-

tions directly correlates with the conciseness of a proof, which in turn is crucial in order to effectively communicate the idea of a proof. Which high-level proof is actually still acceptable depends, of course, on the expertise of the audience; for instance, proofs in introductory mathematical textbooks are much more detailed than in advanced mathematical textbooks. Thus, thirdly, the detailedness of proofs adds a further dimension along which a proof may vary.

Proof Analysis Criteria. When judging a proof there are local and global aspects: On a local scale, one may judge each proof step separately. The first and foremost criterion, of course, is whether the proof step is correct or not. For an incorrect proof step, it is important to know why it is wrong in order to be able to repair or remove it. On the other hand, a correct proof step is not necessarily relevant for a proof, such as tautological steps or redundant proof steps. Thus, a second criterion is the relevance of a proof step. Furthermore, conciseness is important to convey the idea of a proof. On the local scale of proof steps this boils down to judging if a proof step is of adequate size and whether the details of its justification are appropriate. Finally, for textbook proofs it matters whether a backward-style proof step should preferably be re-formulated into a forward step or vice versa. On a global scale, the conciseness of a proof in the sense of whether the proof is without detours, or if there exist alternative shorter proofs, are interesting aspects.

Proof Construction. The goal to write proofs in textbook style requires knowledge of how to find a proof in the first place. Both the construction of a proof as well as the ability to come up with a good presentation of the proof which conveys the proof idea well requires proof strategic knowledge, as well as knowledge about when which proof strategy is appropriate or inappropriate. A human teacher tries to develop that knowledge by training the students on proof problems requiring a specific strategy. When students get stuck in the proof, trying to help them by strategic advise is the primary choice instead of showing them the detailed next steps.

2.1 Wizard-of-Oz Experiment

Our approach to the intelligent tutoring of proofs is informed by experiments to study the specific requirements for such a system. In particular, we used the Wizard-of-Oz paradigm[1] to assess the requirements for the system's sub-components to fulfill the tasks of the inner loop that are specific to proof tutoring, as outlined in the previous section.

The data was collected in an experiment where thirty-seven students interacted with a mock-up of a natural language dialog tutoring system for mathematical proofs. The system was simulated via a specific software environment [17] and the help of four experienced human tutors. We obtained a corpus of tutorial dialogs [16] that allowed us to study the actions of students and tutors related to proof exercises illustrating the properties of binary relations. Student input consisted of natural language text and formulas, to investigate the prospect of natural-language understanding for mathematics within such a tutoring system. In addition to providing feedback to student's actions (proof steps, questions or comments), the tutors rated each proof step with respect to correctness, granularity (or proof step size) and relevance to the current task.

Figure 1 shows a fragment of a tutorial session in which the student was instructed to prove the theorem $(R \circ S)^{-1} = (S^{-1} \circ R^{-1})$, where R and S are relations, and $\circ$ and $^{-1}$ denote relation composition

[1] Wizard-of-Oz experiments [40] simulate a complex system via a partial/prototype implementation that is assisted by a human expert (the "wizard"). Such experiments provide valuable data to assist the design and to evaluate components of such a system in advance of its completion.

S8: let $(x,y) \in (R \circ S)^{-1}$ **T9:** correct **S10:** hence $(y,x) \in (S \circ R)$ **T11:** incorrect	**S8a:** we consider the subgoals $(R \circ S)^{-1} \subset S^{-1} \circ R^{-1}$ and $(R \circ S)^{-1} \supset S^{-1} \circ R^{-1}$	**S8b:** first, we consider the subgoal $(R \circ S)^{-1} \subset S^{-1} \circ R^{-1}$

Figure 1: Examples illustrating phenomena of the corpus

and relation inverse, respectively. In the examples, **S** refers to a student turn and **T** to a tutor turn. The approach taken by the student in the first example on the left of Figure 1 is to apply set extensionality and then to show that the subset relation holds in both directions. The student begins in utterance **S8** by directly introducing a pair (x,y) in the set $(R \circ S)^{-1}$. This is rated as correct by the tutor, who recognizes that the student wants to prove both directions separately and that the introduction of the pair (x,y) is useful due to the definition of subset. The student then states an incorrect formula in **S10**, which the tutor rates as incorrect.

Two alternative ways that the student could have started the same exercise, which we will use as running examples in this paper, are shown on the right in Figure 1. In **S8a** the student explicitly splits the proof into two subgoals with an application of set extensionality. In **S8b** the same rule is applied, but only one of the two resulting proof obligations is explicitly presented.

We analyzed those utterances from the corpus which contain contributions to the theorem proving task. We were able to identify five general phenomena which must be accounted for in order to correctly verify (or reject) the proof steps that students perform and to maintain correct consistent representations of the proofs they are building. These phenomena show that verification in this scenario is not simply a matter of logical correctness, but must also take into account the proof context, for instance.

Underspecification. Some subset of the complete description of a proof step is often left unstated. Utterance **S8** is an example of a number of different types of this underspecification which appear throughout the corpus. The proof step in **S8** includes the application of set extensionality, but the rule and its parameter are not stated explicitly. The student also does not specify that of the two subgoals introduced by set extensionality, he is now proving one particular subset direction, nor does he specify the number of steps needed to reach this proof state. Part of the task of analyzing such steps is to instantiate the missing information so that the formal proof object is complete.

Incomplete Information. Proof steps can, in addition to issues of underspecification, be missing information which is necessary for their verification by formal means. For instance, utterance **S8b** is a correct contribution to the proof, but the second subgoal is not stated. This second subgoal is however necessary to verify that proving the subset relation is part of justifying the equality of the sets, since one subgoal alone does not imply the set equality which is to be shown.

Ambiguity. Ambiguity pervades all levels of the analysis of the natural language and mathematical expressions that students use. Even in fully specified proof steps an element of ambiguity can remain. For example in any proof step which follows **S8a**, we cannot know which subgoal the student has decided to work on. Also, when students state formulas without indicating a proof step type, such as “hence” or “subgoal”, it is not clear whether the formula is a newly derived fact or a newly introduced subgoal. Again, this type of ambiguity can only be resolved in the context of the current proof, and when resolution is not possible, the ambiguity must be maintained by the system.

T:	[Show] $(R \cup S) \circ T = (R \circ T) \cup (S \circ T)$
S8:	$(a,b) \in (R \cup S)$, if $(a,b) \in R$ or $(a,b) \in S$
T:	Correct. [correct] [appropriate] [relevant]
S9:	$\exists x$, such that $(a,x) \in (R \cup S)$ and $(x,b) \in T$
T:	What does this follow from? [correct] [too coarse-grained] [relevant]
S10:	This follows from the definition of relation product for binary relations: $(a,x) \in R$ and $(x,b) \in S$, hence $R \circ S$
T:	More concretely: it follows from $(a,b) \in (R \cup S) \circ T$ [correct] [appropriate] [relevant]

Figure 2: Dialog fragment illustrating tutor's intervention for inappropriate granularity of **S9**

Proof Step Granularity. As outlined above, proofs can generally be constructed at various levels of detail. In a tutorial setting, however, the tutor needs to ascertain that the student develops the proof at an acceptable pace. For this task, classical reasoners are of little help, since they usually provide large proof objects based on some particular logical calculus, such as resolution, that operate at a different step size (granularity) than typical mathematical practice. Generally, typical proof steps may represent several steps in a more formal representation, such as the natural deduction calculus or proofs at the assertion level. This is even the case for proofs at the beginner level (cf. [15]).

In the Wizard-of-Oz studies, the tutors were found to react to deviations in step size, e.g. in the dialog fragment in Figure 2, where the student is skipping a sub-step the tutor expects to see. Generally, whether a proof is of acceptable step size (granularity) depends on the student's knowledge (which we represent via a simple overlay student model) and other factors.

Relevance of Proof Steps. The tutors in the experiments indicated when they believed that steps suggested by the student were not goal-directed. One problem that we address in this work is that the student may introduce hypotheses (e.g. "let $(x,y) \in (R \circ S)^{-1}$" in Figure 1), which may or may not be useful with respect to the current proof goal. We refer to this form of assessment as *relevance checking.*

2.2 Domain Modelling & Teaching Strategies

ITS are designed to be effective – i.e., to lead to increased knowledge and skill via engaging the student with the system. There are three main approaches in the literature on how to build an ITS: Model tracing tutors, constraint based tutors, and example tracing tutors:

Model Tracing Tutors (MTTs), such as the Andes physics tutor (see [71]), contain a *cognitive model* of the domain that the tutor uses to "trace" the student's input, i.e., to infer the process by which a student arrived at a solution. MTTs are based on the ACT-R theory of skill knowledge [3] that assumes that problem solving skills can be modeled by a set of *production rules*. Given a student input, a *model tracer* then uses these rules to find a *trace*, i.e., a sequence of rule applications that derive the student's input. If such a trace can be found, the student is assumed to have used the same reasoning as encoded in the rules to arrive at his input and the step is reported to be correct (cf. (S1)). Thus, the tutor can use the trace to analyze the cognitive process of the student. Alternative solutions are supported by providing rules that capture alternative solution approaches.

To be able to also trace common student errors, a MTT typically provides a set of *buggy rules* (see [20]) that model incorrect reasoning. If a trace contains one or several buggy rules, the step is assumed to be incorrect and error specific feedback can be given to the student (cf. (S2)).

In practice, it turns out that a model tracer will not always be able to trace all student inputs, for example, if a solution cannot be found due to the complexity of the search space, or because a faulty step is not captured by a buggy rule. In this case, it is either assumed that the student step is wrong or an undefined answer is returned by the analysis component.

MTTs can offer strategic and context sensitive problem-solving hints on demand by computing a solution for the current proof state using the expert module (cf. (S3)). By analyzing this solution, hint sequences can be computed that contain increasingly more information about the next step to be performed. The dynamic generation of the solution guarantees that the hint will be tailored to the specific situation in which the student got stuck.

Constraint Based Tutors (CBTs), such as the SQL tutor (see [64]), are based on Ohlsson's theory of learning from performance errors (see [54]) and use *constraints* to describe abstract features of correct solutions. There are two fundamental assumptions:

(i) Correct solutions are similar to each other in that they satisfy all the general principles of the domain.

(ii) Diagnostic information is not contained in the sequence of actions leading to the problem state, but solely in the problem state itself.

Constraints describe equivalent student states and consist of three components: a *relevance condition*, a *satisfaction condition*, and a *feedback message*. The relevance condition describes the abstract properties of the class of solution states that is represented by the constraint. The satisfaction condition contains additional checks a state of this class must satisfy in order to be correct. The feedback message contains the feedback that is given to the student when the satisfaction condition is not satisfied, i.e., when an error is detected (cf. (S2)). If the student enters a situation where the tutor has no knowledge of, i.e., no relevance condition evaluates to true, the CBM remains silent (cf. S(1)). Typically, CBTs provide two kinds of constraints: syntactic constraints and semantic constraints. Syntactic constraints check whether the input is well-formed, whereas semantic constraints compare the input with an optimal solution provided by the tutor. Semantic constraints also check for alternative solutions by capturing alternative subexpressions of a solution.

As CBTs are not equipped with an expert system that solves problems, they cannot automatically complete solutions for a given problem state. Therefore, hints can only be given by comparing the current solution state with an ideal solution, trying to detect missing features. This entails the risk that the hints that are given are overly general or misleading if the student follows a solution different to the ideal solution that is given by the tutor (cf. (S3)).

Example Tracing Tutors (ETTs), such as the stoichiometry tutor (see [48]), interpret a student's solution step with respect to a predefined *solution graph* that represents a generalized solution, which is often also called *behavior graph* (cf. [52]). A behavior graph is a directed, acyclic graph, whose nodes represent problem solving states and whose edges represent problem solving actions. Several outgoing edges represent different ways of solving the problem represented by the state corresponding to the node. Misconceptions and common errors can be included within the graph using so-called *failure links* that indicate typical failures. This way, ETTs can give specific feedback to both correct and incorrect steps (cf. (S1, S2)).

Initially, the current student's state is the root node of the graph, which is the only node that is marked as visited. Given a student's input, the input is matched against all outgoing edges of the current node. If the input matches a regular link leading to a yet unvisited node, the step is classified to be correct, if it matches a failure link or no link at all, it is classified to be incorrect. If a step matches

Property	MTT	CBT	EBT
feedback correct step	yes	no	yes
error specific feedback	buggy rules	satisfaction condition	error paths
hints	context sensitive and strategic	only missing elements	strategic
authoring	production rules and strategies	constraints and ideal solution	all solution paths
runtime costs	high	low	low
step size diagnosis	yes	no	half
alternative solutions	yes	yes	yes
diagnosis no match	error	correct	error
theory	ACT-R	Ohlsson's theory of learning	-

Table 1: Comparison of MTT, CBT, and EBT

multiple regular links, all successor nodes represent *possible interpretations* of the student's step, which are then maintained in parallel. Behavior graphs can be extended to *generalized behavior graphs*, e.g., by defining groups of unordered steps (so the student can change the order of the steps), or by generalizing the matching condition for a link.

Behavior graphs are also used to provide hints as to what a student might do next (cf. (S3)). This is done by identifying an unvisited link in the behavior graph, and then displaying a hint message associated with that link. ETT have the advantage that they do not require to model domain knowledge in form of production rules or constraints and are therefore considerably cheaper to develop. However, they work only for solutions that were foreseen by the author of the exercise.

Summary. Table 1 summarizes the properties of the different approaches to design an ITS.

2.3 Functional Specification for the Proof Tutoring System

For our Wizard-of-Oz experiments we deliberately did not impose any restrictions on the language used to write proof steps, and the input varied from pure formulas to pure natural language and all forms of mixed natural language and formulas (see the example fragment shown in Figure 1). Processing that input poses challenging problems for natural language understanding. In order to have a clear separation of concern, we devised a clear, formal interface language for the kernel module, which serves as target for the natural language analysis component(s) that still need to be developed (see [74] for recent work on that topic). This also has the advantage that different natural language analysis components for different languages can be used on top of the kernel module.

Our interface language for the kernel module is a declarative proof language (see for example [65, 72, 9, 26, 66]) that has been modified to support the elision of information that is typically required to facilitate the verification process. This is because declarative proofs that can be processed by current proof assistants are usually much more detailed than corresponding textbook proofs that we want to teach. For example, we do not enforce the student to give justification hints or restrict the student to a specific granularity. By allowing arbitrarily large gaps between the commands, one arrives at the notion of a *proof plan* [28] or *proof sketch* [73]. Following van Lehn's requirement that an ITS should allow a

proof	::= **proof** *steps* **qed**	*assume*	::= **assume** *form*+ *from steps* **thus** *form*
steps	::= (*ostep*;*steps*) | *cstep*	*fact*	::= *sform* | *by from*
ostep	::= *set* | *assume* | *fact* | *goal*	*goals*	::= **subgoals** (*goal*)$^+$ *by*
cstep	::= *trivial* | *goals* | *cases* | ε	*cases*	::= **cases** (*form* { *proof* })$^+$ *by from*
by	::= **by** *name*? | *proof*	*goal*	::= **subgoal** *form* (**using** *form* (**and** *form*)$^+$)? *by*
from	::= **from** (*label* (, *label*)*)?	*set*	::= **set** *var*=*form* (**,** *var*=*form*)*
sform	::= *form* | **.** *binop form*	*trivial*	::= **trivial** *by from*

Figure 3: Ω*mega* proof script language

student to stepwise construct a solution, we obtain as individual building blocks single declarative proof commands. This approach has the following properties:

1. Proof commands are the primary solution steps a student can enter.
2. The sum of all proof commands gives a complete solution in the style of a textbook proof.
3. A proof command is justified in the context of the previously given steps.
4. Proof steps might be partial, incomplete or underspecified.

We use Ω*mega*'s declarative proof script language presented in Figure 3 (see also [8]) as input language and allow underspecified proof scripts that are obtained by omitting "*by*" and "*from*" as well as the "**thus** *form*" in *assume*-proof steps. We also added them as special closing proof steps (*cstep*) at the end of *steps* following an introductory **assume**. Similarly, we relax the required **qed** at the end of a **proof**.

To develop the intelligent tutor we follow the MTT approach: first, the size of the solution space of textbook-style proofs with adaptive degree of detailedness ruled out the EBT approach. Since the solutions are proof sketches, the constraints that would have to be formulated when following the CBT approach would have to be constraints on proof sketches. The formalism required to formulate such constraints comes close to what is expressed in the tactic and declarative proof script language of the proof assistant system. The difference is that constraints are descriptive while information contained in tactics is constructive, which makes them attractive to generate (strategic) hints. Finally, the solution objects, i.e. (sketches of) declarative proofs, are themselves abstract structurings of parts of the cognitive model representing how the student solves the problem (proving the theorem). Filling the remaining gaps in proof sketches allows to obtain a very detailed cognitive model.

The functional specification of the ITS for proofs is shown in Figure 4: The student's inputs are analyzed by a natural language processing module, which provides either a declarative proof command in the declarative proof script language as output, or the information that a hint has been requested. The ITS maintains the focus on the current open goal and interprets the inputs in that context. Subsequently, the proof analyzer tries to automatically reconstruct missing proof steps and derive missing justifications. If that succeeds, the proof step is further analyzed with respect to granularity and relevance by the granularity analyzer. If the reconstruction fails, the proof assistant tries to find a reconstruction using in addition buggy rules specified by the tutor. In either case, the result is a feature vector composed of local proof analysis criteria *soundness*, *granularity*, and *relevance* (see Section 2). Due to the characteristics of the solution space, several alternative proof reconstructions may be possible for the same proof step sketch. In order to not rule out any of the possible solutions the student may follow, the ITS must trace all possible reconstructions simultaneously. All this is described in Section 3.

In order to generate hints, the ITS shall be able to provide hints with increasing degree of explicitness. It shall use automated theorem proving to try to complete the current partial proof (branch) to a complete

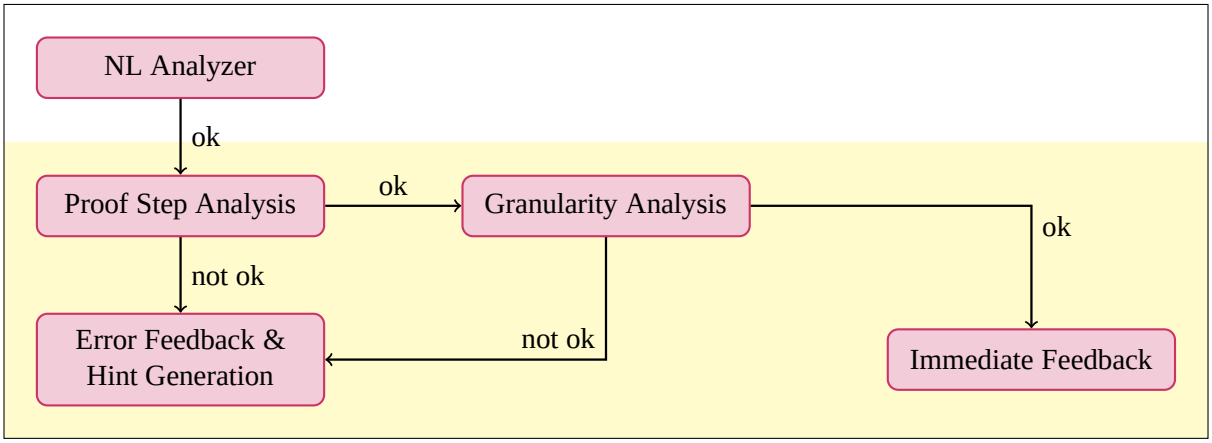

Figure 4: Workflow

proof and use the information used for proof search to generate the hints. Proof search is conducted by specific tactics, which the teacher can provide for a specific mathematical domain. The tactics encode strategic knowledge and the hierarchy of tactics used to complete a proof is recorded in order to be exploited to generate hints with increasing degree of explicitness. This is described in Section 4.

3 Step Analysis

Human one-on-one tutoring is thought to be effective due to its very interactive nature and frequent (step-by-step) feedback [50]. It was found that using step-by-step feedback in an ITS translates to significant learning gains (cf. [25]). A recent meta-analysis [70] determines that tutoring systems with step-based feedback are almost as effective as human tutoring, and more effective than systems that are answer-based (i.e. they provide feedback at the level of the solution). Interestingly, systems that use even finer levels of feedback (so-called sub-step feedback) are found to be (only) similarly effective to traditional step-based systems.

Implementing the concept of step-wise tutoring requires that, whenever the student performs a proof step, the step is evaluated in the current context by a step analyzer. In our approach, a three-dimensional *feedback vector* is computed with an entry for a proof step's soundness, granularity, and relevance, respectively. Computing the feedback vector is a two-staged process: First, a *reconstruction algorithm* is started that tries to relate the proof step given by the student to the current proof situation. Afterwards, the derivation obtained from the reconstruction algorithm is analyzed to compute the granularity and relevance measure.

When and how the computed feedback is given to the student is determined by the *feedback policy*. Our default feedback policy is to give feedback on the correctness of a proof step immediately, whereas feedback on the granularity and relevance is only given when the student violates the condition that is demanded by the tutor.

3.1 Proof Step Reconstruction

Didactic considerations require theorem provers to support *actual mathematical practice*, in addition to providing powerful automation in a selected mathematical domain. Since the development of classi-

cal automated search based theorem provers and the corresponding investigations of logical calculi are mainly driven by correctness, completeness and efficiency issues, these theorem provers operate not on a "human-oriented level", but almost on the "machine code" of some particular logical calculus, such as resolution. Hence, they can generally not be used as a model tracer. While there exist techniques to convert (completed) resolution proofs or matrix proofs into natural deduction proofs, (see for example [4, 51]), it turns out that performing the proof search directly at a more abstract level is beneficial for the runtime of the reconstruction. Moreover, it provides the possibility to run in a "discovery" mode without explicitly having to state an isolated proof obligation, which is often very difficult in a tutorial context due to underspecification or incomplete information.

Abstract Reasoning: The Assertion Level. To come close to the style of proofs as done by humans, Huang [37, 38] introduced the *assertion-level*, where individual proof steps are justified by axioms, definitions, or theorems, or even above at the so-called *proof level*, such as "by analogy". The idea of the assertion-level is, for instance, that given the facts $U \subset V$ and $V \subset W$ we can prove $U \subset W$ directly using the assertion:

$$\subset_{Trans}: \forall U.\forall V.\forall W.U \subset V \wedge V \subset W \Rightarrow U \subset W$$

An assertion level step usually subsumes several deduction steps in a standard calculus, say the classical sequent calculus [32]. Therefore, traditional theorem provers can only achieve such conclusions after a number of proof steps. To use an assertion in the classical sequent calculus, it must be present in the antecedent of the sequent and be processed by means of decomposition rules, usually leading to new branches in the derivation tree. Some of these branches are subsequently closed by means of the axiom rule which correspond to "using" that assertion on known facts or goals.

The technique to obtain such inferences automatically from assertions follows the introduction and elimination rules of a natural deduction (ND) calculus [32] and can be found in [29].

The Reconstruction Algorithm. The proof step reconstruction algorithm is based on two main ideas (see [30] for details): (i) Represent the possible states the student might be in as so-called *mental proof state* (MPS). (ii) Given a new proof step and a MPS, perform a depth-limited BFS at the *assertion level*, trying to derive one/several successor states that are consistent with the student's utterance, where the consistency is proof command specific. The depth limiter imposes an upper bound on the number of assertion level inferences that are assumed to be contained implicitly in the student's input.[2] Whether this limit is sufficient depends on (i) the step size of the available proof mechanism and (ii) the experience of the student, as we discuss in Section 3.2. We have determined such a bound empirically for the corpus of students' proof steps from the Wizard-of-Oz experiments, as discussed in Section 5. The bound is needed to guarantee termination of the reconstruction algorithm, which might otherwise not terminate.

Figure 5 shows an example reconstruction of a complete dialog taken from the corpus. In the figure, the shaded formulas correspond to the steps entered by the student. The white formulas correspond to assertions the student has left out and which were filled in by the reconstruction module.

A MPS is represented as a set of sequents that are the subproblems to be solved, together with a global substitution which instantiates meta-variables. One of these sequents is always marked and represents the sequent the student is working on. Always keeping track of the student's subgoals facilitates task sensitive feedback.

[2] Note that the correspondence of student steps to calculus steps may vary for each calculus.

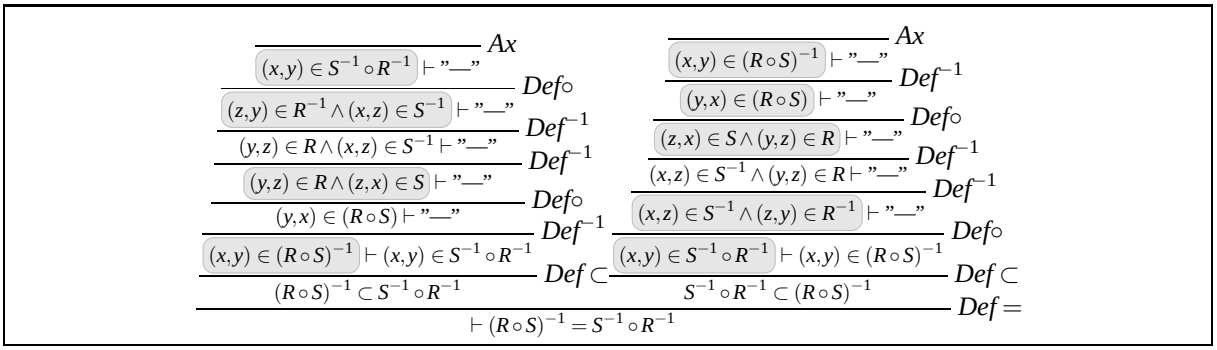

Figure 5: Annotated Ω*mega* assertion level proof

Initially, the MPS is unique and consists of the exercise given to the student as single sequent, together with the empty substitution. During the search, an invariant is that a MPS always represents a valid proof state. By expanding a given proof state only by valid actions, it is guaranteed that only reachable and consistent proof states are generated. Let us stress again that due to ambiguity and underspecification several consistent successor states are possible (as in the case of statement **S8b** shown in Figure 1 where the next subgoal the student will work on is underspecified). There can also be several reconstructions for a given proof step. Therefore, the verification algorithm works on a list of MPS rather than on a single one.

While the reconstruction algorithm might look similar to the processing model of proof commands in a pure verification setting, there are the following subtle differences:

- 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 need to relate them to the student's knowledge 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.

In order to illustrate how the verification algorithm works, we will step through the verification of utterance **S8** from Figure 1, beginning with the initial MPS and finishing with the MPS extended by the proof step. The initial MPS is $\{\langle \underline{\vdash (R \circ S)^{-1} = S^{-1} \circ R^{-1}}; \emptyset \rangle\}$ and the proof step to be verified is **let** $(x,y) \in (R \circ S)^{-1}$.

Having expanded the current proof state (step (i), shown in Figure 6), we apply a **let**-proof step specific filter to find the set of newly-created sequents which are consistent with the given proof step. Of the sequents in the tree, only the node containing the sequent T_k passes, since the formula in the proof step appears on the left-hand side of the sequent. Now that we have found the consistent successor sequents, we must complete these sequents to MPSs. Because the decomposition of the sequent T_0 introduced a subgoal split, the sequent T_j must be proved in addition to T_k. The resulting MPS is therefore $\{\langle \underline{T_k}, T_j; \emptyset \rangle\}$, that is, T_k is now the current sequent, and T_j is still to be proved. Finally, we prune the nodes which were rejected by the filter.

Relevance Checking. Using the proof step reconstruction mechanism for each proof step allows our approach to perform a form of relevance checking when a hypothesis is introduced. A hypothesis introduced by the student is matched against a proof search in Ω*mega*, and considered relevant only if it can be unified with a step that is part of one of the partial solutions that are discovered via strategic proof search.

In practice, it turns out that it is very important for a tutoring system to enable a broad range of people to create content for the system in form of exercises and domain expertise. One of the main advantages of our approach is to use existing mature representation and search technology that has been developed over the last decades in the context of ITP/ATP. New domains can easily be added by users either by relying on already existing specifications of formalized mathematics, or by writing new specifications from scratch. That is, the only information the author has to provide is a problem description and the knowledge needed to solve the problem. As inferences are automatically synthesized from theorems and definitions, it is sufficient to provide this knowledge in a declarative form. For simple domains, this is already sufficient and there is a high chance that modifications of existing proofs or even new proofs are recognized by the tutor. For more complex domains in which the reasoning is more complicated, the author also has to provide strategic information on how to solve a problem.

3.2 Granularity Analysis

In addition to verifying the correctness of proof steps generated by the student, and to detect steps that are logically incorrect, we use proof reconstructions to judge about another qualitative aspect of proof steps: granularity. By assessing the step size (in the context of the ongoing proof and a student model), a tutoring system for proofs can react if the student's solution lacks necessary detail, or, to the contrary, the student is progressing at smaller steps than expected, and adapt feedback and hints accordingly. Having a metric for step size also allows the system to generate and present hierarchical proofs (or steps to be used as hints) at specific levels of granularity.

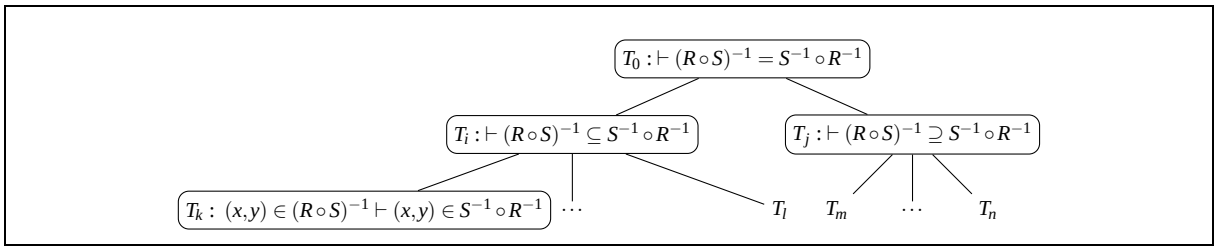

Figure 6: The expanded proof state after step (i) of verification (abbreviated).

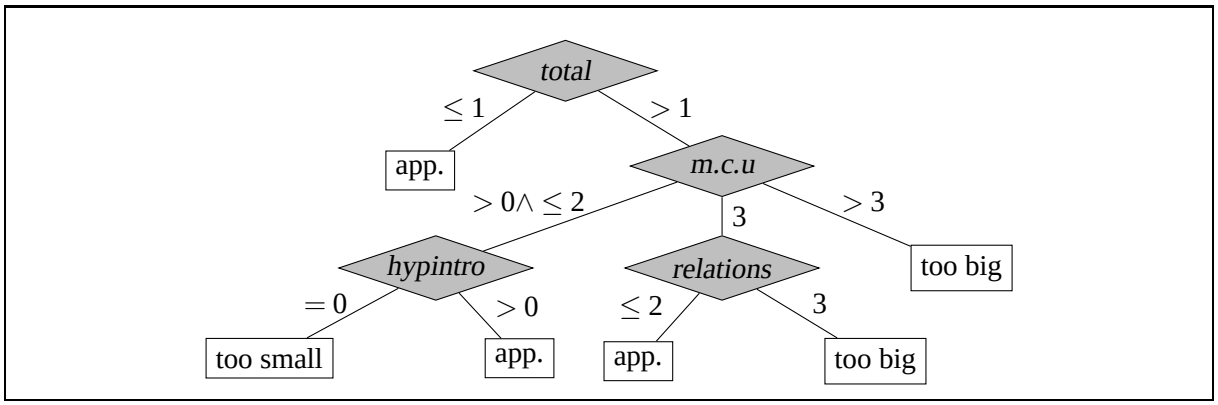

Figure 7: Example granularity classifier

We have devised a framework to analyze the step size of proof steps [58], where a proof step can refer to either the single application of an inference rule, or consist of several (tacit) intermediate inference applications provided by the reconstruction algorithm. Granularity judgments for such a (single or aggregate) step are considered as a classification task. Proof steps are characterized according to a catalog of criteria (cf. [58]) that are thought to be indicative of granularity, and classified as appropriate, too small, or to big according to a classifier.

Granularity criteria, which are the basis for the classification task, take into account the current proof context, the content of the student model, and the verbal explanations given by the student. We currently use an overlay student model which for each assertion-level inference rule maintains an assumption whether it is mastered by the student or not, based on the student's actions.[3] Analyzing a proof step with respect to the catalog of criteria yields a vector that is encoded numerically. For example, the criterion "unmastered concepts" is assigned the count of mathematical concepts employed as inferences in the (simple or aggregate) step which are supposed to be not yet mastered by the student. Classifiers can be represented in the form of decision trees, where decision nodes represent granularity criteria, and leaves record the granularity verdict. In our evaluation discussed in Section 5, we have also considered other forms of classifiers, such as rule based classifiers and classifiers learned by the support vector machine approach. An example for a simple decision tree classifier for granularity is presented in Figure 7. According to this particular classifier, for example, a proof step that consists of two inference applications at the assertion level (total=2), which represent two different concepts both of which have previously been mastered according to the current state of the student model (m.c.u.=2) and none of which introduces a new hypothesis into the proof (hypintro=0) is classified as "too small". Note that this particular decision tree only uses a small number of criteria. Granularity classifiers can be written by hand, but an interesting question is what kind of judgments human experts actually make when assessing proof steps. In order to assess what granularity criteria are relevant for human tutors, and whether corresponding classifiers can be learned from samples of proof steps annotated with granularity judgments, we conducted an experiment presented in Section 5.

[3] There are more sophisticated techniques (e.g. Bayesian networks) for estimating the student's knowledge that can be used instead. However, student modelling as such was not the focus of this research.

4 Next Step Generation for Hinting

At any time of a tutorial session, a student might get stuck and request help on what step to perform next. Help can also be given without an explicit request, for example after repeated student errors, or a long period of silence. Thus, one important design decision for a tutoring system is *when* to give a hint, i.e., to provide a *hinting policy*. For our tutoring system, we use the simplest possible hinting policy, namely to give a hint only if it is explicitly requested by the student. As discussed in [2], this might not be optimal, as students might abuse this functionality or refuse to ask the system for a hint; nevertheless it is the strategy that is used in most tutoring systems.

It has been shown that human tutors use hint sequences that start with abstract hints and refine the hint on demand. The principle of progressively providing more concrete hints if required has been applied to a number of tutoring systems, including the Carnegie Proof Lab [60]. The goal is to leave the student to perform the actual concrete steps that the hint has requested, leading to better knowledge construction. If the student is still stuck, subsequent hints should refer to smaller subtasks of the proof, becoming increasingly close to the fully-specified assertion level step.

To find a relevant, context-sensitive hint, we follow the typical approach of MTTs and invoke the domain reasoner to find a solution for the current proof state. This solution is then analyzed to extract a hint. In our approach, the provision of increasingly concrete hints is supported by a problem solver that generates a *hierarchical solution* (see [7] for details) based on proof strategies, where each (sub)invocation of a strategy introduces a new hierarchy in the computed solution. Intuitively, a high level in the hierarchy sketches how the overall problem was structured into subproblems. At the lowest level, a concrete proof with concrete assertion steps is given. Consequently, we can synthesize both *strategic hints* as well as information about the *concrete next step* to be performed. Of course, the quality of the hints directly depends on the quality of the proof strategies, i.e., how the knowledge is encoded by the author of the exercise.

We first describe in Section 4.1 how an author can encode proof strategies and how these strategies are used to generate a hierarchical proof object, and in Section 4.2 how the computed solution is used to synthesize a hint.

4.1 Authoring 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 (strategic) goal, a strategy performs a heuristically guided search using a dynamic set of assertions, as well as other strategies.

In contrast to other approaches that require to encode the knowledge in the underlying programming language of the system, we encode proof strategies in a separate strategy language (see [31, 8] for an overview).

A simple proof strategy that is proposed in [62] is the “Forward-Backward Method”, which combines the two well-known problem solving strategies: *forward chaining* and *backward chaining*: The method starts with backward chaining by matching the current proof goal with the conclusions of theorems and definitions and adding their premises as new goals to be proved. The backward chaining phase continues until all conclusions have been solved or until no further definition or theorem can be applied. Subsequently, a forward chaining phase is started. Forward chaining starts from available assumptions and given facts and continuously applies definitions and theorems forwards by instantiating all their premises and adding their conclusions to the current proof state.

```
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-logic
```

Figure 8: Formalization of a simple proof strategy

We have formalized this method as a strategy "Close-by-Definition". Each phase is realized by a sub-strategy: "Work-Backward" applies definitions in backward direction, as indicated by the keyword **backward**, that is, expands definitions that occur in the goal. "Work-Forward" applies all definitions in forward direction, as indicated by the keyword **forward** (see Figure 8), that is, expands concepts that occur on the left-hand side of the sequent. It is the responsibility of the author of the strategy to guarantee termination.

As additional logical steps are commonly needed to close a proof task, we provide a third strategy "Close-by-Logic", which performs case-splits and applies the axiom rule to close sequents. Finally, these strategies are assembled to the overall strategy "Close-by-Definition", which calls the strategies in a specific order. The strategy keyword **try** ensures that a strategy application can also be skipped, e.g., when the student has already expanded the goal completely.

4.2 Hinting

Once a solution of the current proof state has been computed in the form of a hierarchical proof, the proof hierarchies can be used to synthesize hints of increasing specificity. This is done as follows: (i) selecting a certain level of hierarchy in the reconstruction, (ii) selecting a successor state at the selected hierarchy (iii) extracting information from the selected state and converting it to a concrete hint. A given hint can be refined by either switching to a more detailed level in the hierarchy, or by increasing the information which was extracted from the selected successor state.

How should the next proof step be communicated to the user? A general issue in ITS design is how much scaffolding is to be provided to the learner, which is known as the "assistance dilemma" – both too much and too little assistance hamper learning [41]. So-called *Socratic* teaching strategies (cf. [57]) have been demonstrated to be superior to *didactic* teaching strategies, i.e. hinting based on direct instruction, especially regarding their long-term effects [57, 23, 5]. Socratic teaching is motivated by the idea that learners a priori posses the necessary prerequisites to acquire new knowledge from existing knowledge (cf. [75]). The role of the tutor thus is to moderate this process by asking knowledge-eliciting questions. Such a teaching strategy for the domain of mathematical proofs has been developed and automated by Tsovaltzi [68] and provides the background for our work. By applying the Socratic teaching strategy, the next proof step is not given directly to the student. Instead, a question is formulated that encourages the student to think for himself and construct his own solution to the task at hand.

Similar to Andes (see [34]), we use templates to generate hints in natural language. Variables in the templates are filled with the concrete objects that are available in the actual proof situation. We have designed a general ordered set of templates that can be used for arbitrary assertion level steps; moreover, we attach an ordered set of templates to each strategy. The hints can be classified as follows:

(i) A strategic hint that describes what to apply from a strategic viewpoint, for example: "What asser-

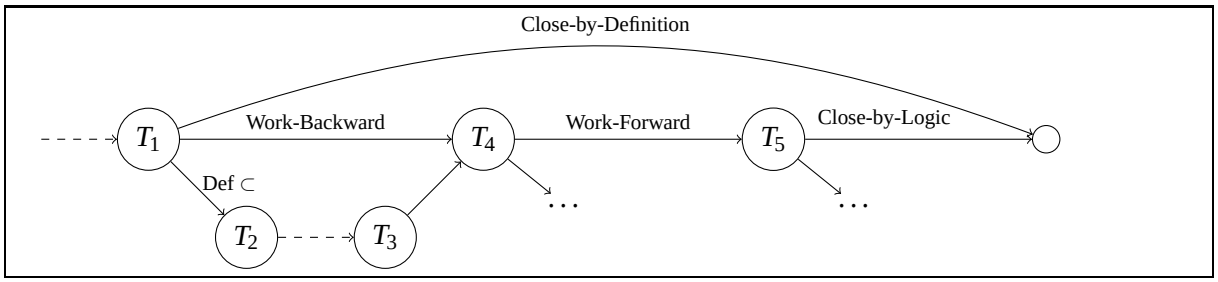

Figure 9: Hierarchical proof plan completing the proof of task T_1

tion can be applied backward to the goal?"

(ii) A hint on an inference to be applied by pointing to involved variables, for example: "Can you say anything about the sets A and B?"

(iii) A hint on an inference to be applied without stating the premises and conclusion that are involved, for example: "How can you show that two sets are equal?"

(iv) A hint that points to the premises of an inference that is applied backwards without naming the inference, for example: "If you want to show that $A \cap B = B \cap A$, what should be true about these sets?"

(v) A hint that points to a subgoal but does not say how the subgoal is achieved, for example: "How can you show that $A \cap B \subset B \cap A$?"

(vi) A hint that points to the conclusion of an inference that is applied forwards without naming the inference, for example: "What can you conclude if you know that $x \in A$ and $x \in B$?"

(vii) A hint that describes the complete application of an inference, that is, the name of the inference, together with the instantiated premises and conclusion.

(viii) A hint that points to an inference application together with restating the assertion to be applied.

Let us illustrate our approach by means of an example. Consider the exercise $(R \circ S)^{-1} = S^{-1} \circ R^{-1}$. Suppose that the student starts the proof by equality, 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 9. 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.

As mentioned above, 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 $(R \circ S)^{-1} \subset S^{-1} \circ R^{-1}$"
- "By the application of Def $\subset$ we obtain the new goal $(x,y) \in (R \circ S)^{-1} \Rightarrow (x,y) \in S^{-1} \circ R^{-1}$"

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". The ability to provide hints that address several dimensions at which scaffolding can be provided in proof tutoring is beneficial for providing targeted, adaptive feedback.

5 Evaluation

The presented techniques have been successfully evaluated using experiments and experimental data from the Wizard-of-Oz experiments described in Section 2.1. In particular, we examined in how far the proof reconstruction mechanism successfully models the proofs from the students (who were unconstrained in their solution attempts). We performed additional experiments to study granularity judgments by human tutors, and in how far these judgments can be learned via machine learning techniques and represented as classifiers within the tutoring system.

Proof Reconstruction & Assessment. In the Wizard-of-Oz experiments introduced in Section 2.1, tutors were asked to indicate explicitly whether steps are correct or incorrect. We investigated in how far the steps judged as correct by the tutors are reconstructed by our approach – to provide appropriate feedback on correctness, but also to serve as the basis for further analysis, e.g. granularity analysis. Since the algorithm uses breadth-first-search, an important question is what search depth is necessary to verify the proof steps from the students.

In our analysis, we used 144 proof steps from the Wizard-of-Oz experiment which deal with an exercise about binary relations, namely to show that $(R \circ S)^{-1} = R^{-1} \circ S^{-1}$ (where the operators $\circ$ and $^{-1}$ denote relation composition and inverse). Of these steps, 116 were judged as correct and 28 as incorrect. Table 2 shows the proportion of steps that were correctly accepted and rejected, and the proportion of steps that were not verified or wrongly accepted, using a depth-limit of four steps for BFS. Apart from three steps, all proof steps are correctly verified or rejected (which corresponds to an overall accuracy of 98%). Since the relatively small depth limit of four steps is sufficient for accurate verification in our sample, we conclude that our approach to proof reconstruction is feasible (within the given domain of proofs).

Granularity. We have used proof reconstructions at the assertion level as the basis for granularity analysis (as outlined in Section 3.2) within a framework for judging granularity (presented in [58]). To investigate the prospect of learning granularity classifiers from the granularity judgments by expert tutors, we asked four expert tutors to contribute to a corpus of granularity judgments. This corpus was used to analyze agreement between tutors, to synthesize granularity classifiers via machine learning techniques, and to determine what granularity criteria are most useful for judging granularity (as presented in more

	Verified	Rejected
Step correct	**113 (97%)**	3 (3%)
Step incorrect	0	**28 (100%)**

Table 2: Number of steps that were verified or rejected via proof reconstruction. Bold numbers indicate accurate classification, relative to the true class of the steps (correct/incorrect).

detail in [58]). A first exploratory analysis revealed that most of the proof steps that were presented to the tutors (which were constructed from one or a few inference steps at the assertion level) were considered to be of appropriate granularity. This is in particular the case for proof steps that correspond to one single inference at the assertion level, which were considered of appropriate size in 93%, 69%, 83% and 96% of the cases by the four tutors. When the bias towards the "appropriate" class is accounted for, agreement between tutors is moderate. The individual sub-corpora of granularity judgments by the four tutors also differed in how far they were found to be amenable to the automated learning of classifiers. We used several algorithms (decision tree learning, decision rule learning and support vector machines) offered by the data mining tool Weka[4], with different degrees of success for the four sub-corpora (cf. [58]). It should be noted that the experiment investigated the naturalistic judgments of tutors without enforcing "consistency" in the judgments, therefore some disagreement was to be expected. Overall, the experiments indicated that the proof steps at the assertion level are a good basis for granularity analysis, since they are close to what human judges (in the setting of our experiments) consider as appropriate step size. Furthermore, counting the number of assertion level steps in the reconstruction of a student's proof step was determined as the most indicative criterion when only judgments are considered where three out of the four judges agree. The experiment illustrates the prospects of learning classifiers from expert tutors, but it also points at the differences between experts in judging granularity.

Discussion. The evaluation illustrates the use of assertion level reasoning for assessing proof steps as they occur in relatively unconstrained tutoring dialogs that we collected in the Wizard-of-Oz experiments. Furthermore, we have shown that these reconstructions are a useful basis for analyzing further aspects of proof steps, such as granularity.

6 Outlook

The previous sections describe how the proof assistant Ω*mega* was utilized as a domain reasoner providing feedback on proof steps and generating hints. We believe that using a proof assistant offers a lot more of not yet exploited potential to improve the quality of ITSs. In this section we present some of these ideas that are the basis for future work.

Further Qualitative Proof Step Assessment Criteria. Reconstruction of a proof step is usually a task that is *local* to an individual student session and can be seen as the process of generating an individual solution graph for the given subject. To make the tutoring system more efficient, it is possible to cache solution graphs from previous tutoring sessions and to combine them to an overall solution graph. This has two major advantages: (i) The instructor gets a compact overview over typical approaches taken by

[4] http://www.cs.waikato.ac.nz/ml/weka/

all of his students, (ii) the instructor can review and refine hints that were given by the system, (iii) each student that comes up with a new solution becomes an author, (iv) whether a proof is sensible or whether a (irrelevant) proof step is nevertheless sensible typically are properties that need the comparison with other solutions. The hope is that eventually a fixed point will be reached, containing a fully specified trace over all possibilities.

Another criterion to analyze a proof step for is whether it is consistent with an overall strategy. For instance, if the teaching goal also consists in teaching a specific proof strategy, then the proof assistant must interpret a student's proof step not only with respect to a current open goal, but also with respect to an upper strategy. In case a strategy consists of different sequential sub-strategies, such as, for instance, the "Forward-Backward Method" from Section 4.1, this also requires to be able to find out when one strategy is finished and the next one starts. How to achieve this tracking of strategy execution is an open problem, but a solution could be to hook the user input into the tactic execution mechanism.

Integration of Model Generators. A variety of tools has been developed for finding finite models of first order logic (FOL) formulas, such as Paradox [24] and Mace [47], to name a few. Given for example the theory of groups and the assertion that all groups are commutative, they are able to produce a counter-model of the assertion, i.e., a non-commutative group. This is done by providing an interpretation for the involved function symbols which makes the assertion false, which can be understood as a group table. Similarly, counter-models can be generated for other theories, such as the theory of binary relations. Note that in contrast to the verification of a proof step, the counter-model provides the information that the given step is wrong. Proof step reconstruction could be made more efficient by systematically checking for counter-models during the reconstruction process. Moreover, in case no reconstruction was possible, model-finders could be employed to generate a counter-model for the proof step and the error-feedback routine could employ the counter-model to provide hints or explain why a proof step is invalid. The challenge here consists of adequately verbalizing the found counter-model.

Reviewing of Solution (service S5, p. 1). Mathematical proofs in textbook-style mainly contain forward proof steps. However, to find a proof often a backward-style is used. Checking if a specific proof step entered by the student is in forward-style or in backward-style is easy. Depending on the didactic strategy and possibly depending on the skills of a student, the ITS has the choice of enforcing forward-style proof steps immediately, or to let a student having difficulties with proving in the first place write a proof in any style. In the second case, the tutor can then review the solution with the student, for instance, by indicating backward proof steps and subsequently asking to transform them into a forward-style proof. Or simply by showing the student his own proof in forward-style, which is easily possible in Ω*mega* already now.

Assessment of Student Knowledge (service S4, p. 1). The student's knowledge is incorporated in the granularity classifier by exploiting the information which concepts are mastered by the student. During the proof development that information is updated, for instance, by adding initially unmastered concepts to the mastered concepts once they have been used correctly a number of times in a proof step. Since our approach uses a full-fledged proof assistant system for the analysis of the student's input, a precise and detailed assessment of the student's actions is provided, which is considered beneficial for student modeling. In this context, the benefit of using state-of-the-art techniques in student modeling for diagnosing student knowledge (using, for example, statistical inference) could be explored. Such a modeling

of student knowledge can enable the system to adjust instructions and exercises to the particular strengths and difficulties of individual students.

Furthermore, an interesting feature that an ITS as outlined in this paper could offer is a walk-through of the student's proof solution, where the proof steps that were rejected by the system are presented along with the reasons for rejection, as well as the accepted proof steps and their relation to the final proof.

Automatic hint generation for logic proof tutoring using historical data. So far the hint generation uses the recorded hierarchy of strategies used to find a proof. Of course, there are choices which hierarchy-level to consider and which form of hint (Socratic vs. didactic, next step vs. strategic) to deliver in a specific situation. Following ideas from [12] the tutor system itself could record the hints given away in specific proof situations and try to assess how useful they were. From that historical tutoring data it could come up with a classifier deciding which form of hint to use in which situation.

7 Related Work

We discuss our work in connection with related approaches that (i) focus on domain reasoning techniques for tutoring proofs in logics and mathematics (AProS and the Carnegie Proof Lab [60], the EPGY Proving Environment [63], Tutch [1] and approaches using hierarchical proofs), and (ii) hint generation (Carnegie Proof Lab, Andes [34] and the NovaNet Proof Tutorial [13]).

APROS. The AProS project (see [60] for an overview) provides an integrated environment for strategic proof search and tutoring in natural deduction calculi for predicate logic. It consists of four modules: the *proof generator* which implements strategic proof search based on the intercalation calculus, the *proof lab*, which builds the interface to the students, the *proof tutor*, which generates hints for students that are stuck, and a web-based course containing additional learning material. Proofs are represented in a Fitch-style diagram and constructed by adding/removing steps to the diagram. This means that the student enters stepwise the proof at the calculus level, and that the performed steps can therefore immediately be checked. If a student requests a hint, the proof generator initiates the construction of a complete proof, which the tutor analyzes to extract a hint. The first hint provided at any point in the proof is a general strategic one, and subsequent hints provide more concrete advice as to how to proceed. The last hint in the sequence recommends that the student take a particular step in the proof construction.

Compared with AProS, which focuses on the teaching of one particular logical calculus (without equality), the main difference with our approach is that we focus on the teaching of more abstract assertion level proofs, which are rather independent of a particular logical calculus. Proofs are essentially constructed in a declarative proof language in the form of proof sketches. If the information provided by the student is complete and correct, the verification is just a simple checking, as in the case of AProS. However, within our setting, this is not the typical situation. Rather, it is common that the information provided by the student is incomplete, as humans typically omit information they consider unimportant or trivial. Therefore, reconstruction of the missing information is necessary, as well as an analysis of the complexity of this information.

The generation of hints is similar to our approach in the sense that (i) it is dynamic and based on a completion of the student's proof attempt, and (ii) that it can be provided at several levels of granularity. However, we do not focus on a particular calculus, neither on a fixed proof strategy. This necessitates the sophisticated techniques presented in Section 3.2 to instantiate the system with the required problem

solving knowledge in the form of assertions and proof strategies, and to model the student's knowledge via a student model, which is used for granularity analysis.

EPGY. The Epgy theorem proving environment aims to support "*standard mathematical practice*" both in how the final proofs look as well as the techniques students use to produce them (see [63] p. 227). To verify the proof steps entered by a student, Epgy relies on the CAS Maple and the ATP Otter. The system is domain independent in the sense that the course authors can specify the theory in which a particular proof exercise takes place. Proof construction works by selecting predefined rules and strategies from a menu, such as definition expansion or proof by contradiction, or by entering formulas. For computational transformations, a so-called derivation system is provided. Once a statement is entered, the student selects a set of justifications that he thinks is sufficient to verify the new statement. The assumptions together with the goal and implicit hidden assumptions are then sent to Otter with a time limit of four to five seconds to verify the proof step.

Compared to our approach, the main similarities are that the system aims at teaching ordinary mathematical practice independent of a particular calculus. Moreover, it uses a theorem prover as domain reasoner to dynamically verify statements entered by a student. The authors acknowledge that the use of a classical ATP to verify proof steps has the following drawbacks ([63] p. 253-254): (i) "One weakness of the Theorem Proving Environment is that, like most computer-based learning tools, it does not easily assess the elegance and efficiency of the student's work". (ii) "In the current version of the Theorem Proving Environment, students are not given any information as to why an inference has been rejected. Students are told generally that an inference may be rejected because it represents too big a step of logic, because the justifications are insufficient to imply the goal, or because the goal is simply unverifiable in the current setting. From our standpoint, Otter's output is typically not enough to decide which is the reason of failure".

In contrast, our approach relies on using the assertion level as a basis to verify statements uttered by a student. This results in an abstract proof object, which can further be analyzed, for example with respect to granularity, as demonstrated in Section 3.2, or to extract hints on how to proceed if the student gets stuck. In particular, the problem whether specified assertions were used in the derivation can trivially be solved. We believe that limiting the runtime of Otter does not reveal any information about the complexity of a particular proof step. While it would also be possible to analyze the resulting proof object, we believe that it is not at an appropriate level of granularity and does not reflect a human-style of proof construction. Even for natural deduction calculi, an investigation [59] into the correspondence between human proofs and their counterparts in natural deduction points out a mismatch with respect to their granularity.

Moreover, our approach is more flexible with respect to the following aspects: (i) Due to the use of a proof language, the student is more flexible in entering the solution. (ii) It is compatible with the buggy rule approach. Note that this is not the case for classical automated reasoners, in which an inconsistent theory makes everything provable. In contrast, our approach allows for a full control over buggy rules, such as limiting their application to a single step. (iii) Our approach supports incomplete information such as a missing subgoal. Note that by leaving out such a subgoal, the resulting proof obligation becomes unverifiable and can therefore not be supported by a classical ATP. Finally, our approach is extensible and supports the specification of domain-dependent proof strategies, as well as checking whether a particular step can be checked by a specified proof strategy. This is not possible in the work cited above.

TUTCH. Tutch (see [1]) is a proof checker that was originally designed for natural deduction proofs in propositional logic. However, it was later extended to also feature constructive first order logic to support human oriented proof steps. To that end, a simple proof language that allows steps at the assertion level was developed, as well as proof strategies that allow for an efficient proof checking for proofs within that language. This is similar to our approach, which also relies on a proof language as well as a dynamic reconstruction of the proof steps. Because of these similarities, we focus on the details of the proof language and the strategies to verify the proof steps.

Hierarchical Proofs. Hierarchical proofs have been advocated by several people, such as Lamport [44] in the context of informal proofs. A similar idea is proposed by Back and colleagues for calculational proofs [11, 10]. In the context of HOL, Grundy and Langbacka [35] developed an algorithm to present hierarchical proofs in a browsable format. Another possibility for hierarchical proof construction is provided by a method called *window inference* [56]. Window inference allows the user to focus on a particular sub-formula of the proof state, transforming it and thereby making use of its context, as well as opening subwindows, resulting in a hierarchical structure.

Our approach is based on previous work by Cheikhrouhou and Sorge who developed the hierarchical proof data structure PDS in an earlier version of the Ω*mega* system, intended to support hierarchical proof presentation and proof search [22]. In particular, the PDS supports so-called "island proofs", i.e. proofs that contain gaps which can be filled via refinement operations. The same idea has been picked up by Denney, who developed the notion of hiproof [27]. Most recently, a tactic language for hiproofs has been proposed in [6].

MathsTiles. MathsTiles [19] are a flexible language to be used as an interface for an intelligent book on mathematics. In particular, proof exercises are offered where proofs formulated by the student are checked via Isabelle/HOL. The tiles correspond to graphical elements that can be arranged and recombined within a mathematical document, and which are used to represent formulas. This allows for flexibility while writing the proofs (e.g. in the order in which proof lines are written). However, the approach in [19] checks proofs linearly. In this context, the question of proof granularity is discussed. The authors state that the students should not use the prover to solve the exercises for them, and therefore the student is limited to using only Isabelle's simplifier (simp). Since rules can be added or removed from the simplifier, this makes the approach configurable. In contrast to our approach, however, such rule sets within the mathematical domain represent only an implicit model of granularity which does not take into account dynamic information such as the proof context and the student's knowledge.

Andes. Andes [34] is an ITS for teaching Newtonian physics. Like in our approach, student and tutor solve problems collaboratively, which is called *coached problem solving*. Andes includes a problem solver, which is run on the problem description to generate a solution graph. The system uses abstract plans, such that the resulting solution graph represents a hierarchical dependency network. Similar to our approach, Andes uses templates to generate hints based on the solution graph [34]. There may be several paths to a solution. Andes uses a Bayesian network for plan recognition to determine on which solution path the student might be on for giving an appropriate hint. As a justification, the authors mention their own informal studies where they found that human tutors rarely ask students about their goals before giving a hint. This can also be considered a motivation for our approach, where we keep track of several possible proof reconstructions simultaneously. Similarly to our approach again, the hints that are generated by Andes range from general to specific.

NovaNet Proof Tutorial. The NovaNet Proof Tutorial [13] is a learning tool for logic proofs. Students write proofs line by line which are verified by the system. To generate hints for the next step, a path through the space of the previously explored actions is sought by optimizing a Markov decision process. This substitutes the use of (strategic) proof search by data-mining a large number of example solutions. The process can be tuned to extract an expert, a typical, or least error-prone solution. For the suggested steps, four levels of hint are generated and presented to the student in sequence; (i) the goal is indicated, (ii) what rule is to be applied, (iii) the statement the rule can be applied to, (iv) both the rule and the statements (bottom-out-hint). These variants of hints are a subset of the hinting categories provided by our approach as presented in Section 4.2, and do not involve proof hierarchies.

8 Conclusion

In this paper we presented a coherent overview of the design of and methods for an ITS to teach students to write mathematical proofs in textbook style. Based on a detailed analysis of the teaching domain, the paper argues in favor of adopting the model-tracing tutor style to support the inner loop of such an ITS. The tutor was then built on top of the slightly adapted proof assistant Ω*mega* to provide step analysis and hint generation. Feedback is provided on each proof step entered by the student in form of a vector composed of the criteria soundness, relevance and granularity. Hints are provided with increasing degree if explicitness.

The following features of Ω*mega* were particularly important to realize the system: First, Ω*mega*'s declarative proof script language allowed to define a clean interface proof sketch language to separate natural language analysis from the pure step analysis and hint generation tasks. Second, the assertion-level proof calculus allowed for a depth limited search to reconstruct missing information in proof steps. We established that information obtained by proof reconstruction is a good basis for classifier-based granularity analysis. In particular, this allows the system to judge granularity based on criteria such as the number of assertions used by the student (which was found to be a useful criterion for modelling human tutors' judgments), the number of which are mastered or unmastered by the student, etc. We furthermore investigated the use of machine learning techniques to instantiate the classification module via corpora of example classifications from human experts. Third, Ω*mega*'s strategy language combining declarative and procedural LCF-style tactics served as an authoring language for domain specific strategic proof procedures. They can be used both to generate a proof from scratch as well as to complete the partial proof of the student. Recording the hierarchy of strategies completing a proof provided an excellent basis for generating hints.

The resulting prototype tutoring system has been evaluated on a corpus of tutorial dialogues to analyze the student inputs and yields good results. Having a proof assistant system as domain reasoner bears a lot of potential in order to improve the quality of proof step analysis, feedback, and reviewing of solutions. Further work is also devoted to design the interface towards the student, which maps the student input to the intermediate formal proof step format.

References

[1] Andreas Abel, Bor-Yuh Evan Chang & Frank Pfenning (2001): *Human-Readable Machine-Verifiable Proofs for Teaching Constructive Logic*. In Uwe Egly, A. Fiedler, Helmut Horacek & Stephan Schmitt, editors: *Proceedings of the Workshop on Proof Transformations, Proof Presentations and Complexity of Proofs (PTP'01)*, Universitá degli studi di Siena.

[2] Vincent Aleven & Kenneth Koedinger (2000): *Limitations of Student Control: Do Students Know when They Need Help?* In Gilles Gauthier, Claude Frasson & Kurt VanLehn, editors: *Intelligent Tutoring Systems, Lecture Notes in Computer Science* 1839, Springer Berlin / Heidelberg, pp. 292–303. Available at http://dx.doi.org/10.1007/3-540-45108-0_33.

[3] John Robert Anderson (1993): *Rules of the mind*. Lawrence Erlbaum Associates, Hillsdale, NJ.

[4] Peter B. Andrews (1980): *Transforming matings into natural deduction proofs*. In: *Proceedings of the 5th Conference on Automated Deduction (CADE)*, LNCS 87, Springer, pp. 375–393. Available at http://dx.doi.org/10.1007/3-540-10009-1_22.

[5] Kevin D. Ashley, Ravi Desai & John M. Levine (2002): *Teaching Case-Based Argumentation Concepts using Dialectic Arguments vs. Didactic Explanations*. In: *In Proceedings of the Intelligent Tutoring Systems Conference*, Springer, pp. 585–595. Available at http://dx.doi.org/10.1007/3-540-47987-2_60.

[6] David Aspinall, Ewen Denney & Christoph Lüth (2010): *Tactics for Hierarchical Proofs*. *Journal Mathematics in Computer Science* 3, pp. 309–330. Available at http://dx.doi.org/10.1007/s11786-010-0025-6.

[7] 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 Kohlhase [43], pp. 126–142. Available at http://dx.doi.org/10.1007/11618027_9.

[8] 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.

[9] Serge Autexier & Armin Fiedler (2006): *Textbook Proofs Meet Formal Logic - The Problem of Underspecification and Granularity*. In Kohlhase [43], pp. 96–110. Available at http://dx.doi.org/10.1007/11618027_7.

[10] Ralph-Johan Back (2010): *Structured derivations: a unified proof style for teaching mathematics*. *Formal Asp. Comput.* 22(5), pp. 629–661. Available at http://dx.doi.org/10.1007/s00165-009-0136-5.

[11] Ralph-Johan Back, Jim Grundy & Joakim von Wright (1997): *Structured Calculational Proof*. *Formal Asp. Comput.* 9(5-6), pp. 469–483. Available at http://dx.doi.org/10.1007/BF01211456.

[12] Tiffany Barnes & John C. Stamper (2008): *Toward Automatic Hint Generation for Logic Proof Tutoring Using Historical Student Data*. In Beverly Park Woolf, Esma Aïmeur, Roger Nkambou & Susanne P. Lajoie, editors: *Intelligent Tutoring Systems, 9th International Conference, ITS 2008, Montreal, Canada, June 23-27, 2008, Proceedings*, *Lecture Notes in Computer Science* 5091, Springer, pp. 373–382. Available at http://dx.doi.org/10.1007/978-3-540-69132-7_41.

[13] Tiffany Barnes & John C. Stamper (2010): *Automatic Hint Generation for Logic Proof Tutoring Using Historical Data*. *Educational Technology & Society* 13(1), pp. 3–12.

[14] Michael Beeson (1992): *Mathpert: Computer Support for Learning Algebra, Trig, and Calculus.* In Andrei Voronkov, editor: *LPAR*, *Lecture Notes in Computer Science* 624, Springer, pp. 454–456. Available at http://dx.doi.org/10.1007/BFb0013085.

[15] Christoph Benzmüller, Dominik Dietrich, Marvin Schiller & Serge Autexier (2007): *Deep Inference for Automated Proof Tutoring*. In Joachim Hertzberg, Michael Beetz & Roman Englert, editors: *KI 2007: Advances in Artificial Intelligence. 30th Annual German Conference on AI, LNAI* 4667, Springer, pp. 435–439. Available at http://dx.doi.org/10.1007/978-3-540-74565-5_34.

[16] Christoph Benzmüller, Helmut Horacek, Henri Lesourd, Ivana Kruijff-Korbayová, 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.

[17] Christoph Benzmüller, Helmut Horacek, Henri Lesourd, Ivana Kruijff-Korbayová, Marvin Schiller & Magdalena Wolska (2006): *DiaWOz-II - A tool for wizard-of-oz experiments in mathematics*. In Christian

Freksa, Michael Kohlhase & Kerstin Schill, editors: *KI 2006: Advances in Artificial Intelligence. 29th Annual German Conference on AI, LNAI* 4314, Springer. Available at http://dx.doi.org/10.1007/978-3-540-69912-5_13.

[18] Yves Bertot, Gilles Dowek, André Hirschowitz, C. Paulin & Laurent Théry, editors (1999): *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Nice, France, September, 1999, Proceedings*. *Lecture Notes in Computer Science* 1690, Springer. Available at http://dx.doi.org/10.1007/3-540-48256-3.

[19] William Billingsley & Peter Robinson (2007): *Student Proof Exercises Using MathsTiles and Isabelle/HOL in an Intelligent Book*. *J. Autom. Reasoning* 39(2), pp. 181–218. Available at http://dx.doi.org/10.1007/s10817-007-9072-3.

[20] John Seely Brown & Kurt VanLehn (1980): *Repair theory: A generative theory of bugs in procedural skills*. In: *Cognitive Science*, 4, pp. 379–426. Available at http://dx.doi.org/10.1207/s15516709cog0404_3.

[21] Stefano A. Cerri, Guy Gouardères & Fábio Paraguaçu, editors (2002): *Intelligent Tutoring Systems, 6th International Conference, ITS 2002, Biarritz, France and San Sebastian, Spain, June 2-7, 2002, Proceedings*. *Lecture Notes in Computer Science* 2363, Springer. Available at http://dx.doi.org/10.1007/3-540-47987-2.

[22] Lassaad Cheikhrouhou & Volker Sorge (2000): *PDS – A Three-Dimensional Data Structure for Proof Plans*. In: *Proceedings of the International Conference on Artificial and Computational Intelligence For Decision, Control and Automation In Engineering and Industrial Applications (ACIDCA) 2000.*

[23] Michelene T. H. Chi, Nicholas De Leeuw, Mei hung Chiu & Christian Lavancher (1994): *Eliciting self-explanations improves understanding*. *Cognitive Science* 18, pp. 439–477. Available at http://dx.doi.org/10.1207/s15516709cog1803_3.

[24] Koen Claessen & Niklas Srensson (2003): *New Techniques that Improve MACE-style Finite Model Finding*. In: *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications.*

[25] Albert Corbett (2001): *Cognitive Computer Tutors: Solving the Two-Sigma Problem*. In Mathias Bauer, Piotr Gmytrasiewicz & Julita Vassileva, editors: *User Modeling 2001, Lecture Notes in Computer Science* 2109, Springer Berlin / Heidelberg, pp. 137–147. Available at http://dx.doi.org/10.1007/3-540-44566-8_14.

[26] Pierre Corbineau (2007): *A Declarative Language for the Coq Proof Assistant*. In Marino Miculan, Ivan Scagnetto & Furio Honsell, editors: *Types for Proofs and Programs, International Conference, TYPES 2007, Cividale del Friuli, Italy, May 2-5, 2007, Revised Selected Papers, Lecture Notes in Computer Science* 4941, Springer, pp. 69–84. Available at http://dx.doi.org/10.1007/978-3-540-68103-8_5.

[27] Ewen Denney, John Power & Konstantinos Tourlas (2006): *Hiproofs: A Hierarchical Notion of Proof Tree*. In: *Proc. of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXI), Birmingham, UK, May 18-21, 2005*, *ENTCS* 155, Elsevier, pp. 341–359. Available at http://dx.doi.org/10.1016/j.entcs.2005.11.063.

[28] Louise A. Dennis, Mateja Jamnik & Martin Pollet (2006): *On the Comparison of Proof Planning Systems: lambdaCLAM, OMEGA and IsaPlanner*. *Electr. Notes Theor. Comput. Sci.* 151(1), pp. 93–110. Available at http://dx.doi.org/10.1016/j.entcs.2005.11.025.

[29] Dominik Dietrich (2010): *Assertion Level Proof Planning with Compiled Strategies*. Ph.D. thesis, Saarland University.

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

[31] Dominik Dietrich & Ewaryst Schulz (2009): *Integrating Structured Queries into a Tactic Language*. *JAL - Special issue on Programming Languages and Mechanized Mathematics Systems* Available at http://dx.doi.org/10.1007/s10817-009-9138-5.

[32] Gerhard Gentzen (1969): *The Collected Papers of Gerhard Gentzen (1934-1938)*. Edited by Szabo, M. E., North Holland, Amsterdam.

[33] GeoGebra (2008): *Dynamische Mathematiksoftware*. http://www.geogebra.org. Accessed December 5th, 2011.

[34] Abigail S. Gertner, Cristina Conati & Kurt Vanlehn (1998): *Procedural help in Andes: Generating hints using a Bayesian network student*. In: *Proceedings of the 15th National Conference on Artificial Intelligence*, AAAI Press, pp. 106–111. Available at `http://www.aaai.org/Conferences/AAAI/aaai98.php`.

[35] Jim Grundy & Thomas Långbacka (1997): *Recording HOL Proofs in a Structured Browsable Format*. In Michael Johnson, editor: *Algebraic Methodology and Software Technology, 6th International Conference, AMAST '97, Sydney, Australia, December 13-17, 1997, Proceedings*, *Lecture Notes in Computer Science* 1349, Springer, pp. 567–571. Available at `http://dx.doi.org/10.1007/BFb0000500`.

[36] Neil T. Heffernan & Ethan A. Croteau (2004): *Web-Based Evaluations Showing Differential Learning for Tutorial Strategies Employed by the Ms. Lindquist Tutor.* In James C. Lester, Rosa Maria Vicari & Fbio Paraguau, editors: *Intelligent Tutoring Systems, Lecture Notes in Computer Science* 3220, Springer, pp. 491–500. Available at `http://dx.doi.org/10.1007/978-3-540-30139-4_46`.

[37] Xiaorong Huang (1994): *Reconstructing Proofs at the Assertion Level*. In Alan Bundy, editor: *Proc. 12th CADE*, Springer-Verlag, pp. 738–752.

[38] Xiaorong Huang (1996): *Human Oriented Proof Presentation: A Reconstructive Approach*. *DISKI* 112, Infix, Sankt Augustin, Germany.

[39] 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.

[40] John F. Kelley (1984): *An iterative design methodology for user-friendly natural language office information applications*. *ACM Trans. Inf. Syst.* 2(1), pp. 26–41. Available at `http://dx.doi.org/10.1145/357417.357420`.

[41] Kenneth R. Koedinger & Vincent Aleven (2007): *Exploring the Assistance Dilemma in Experiments with Cognitive Tutors*. *Educational Psychology Review* 19, pp. 239–264. Available at `http://dx.doi.org/10.1007/s10648-007-9049-0`.

[42] Kenneth R. Koedinger & John R. Anderson (1993): *Reifying implicit planning in geometry: Guidelines for model-based intelligent tutoring system design*. In S. P. Lajoie & S. J. Derry, editors: *Computers as cognitive tools*, Erlbaum, Hillsdale, NJ, pp. 15–46.

[43] Michael Kohlhase, editor (2006): *Mathematical Knowledge Management, 4th International Conference, MKM 2005, Bremen, Germany, July 15-17, 2005, Revised Selected Papers*. *Lecture Notes in Computer Science* 3863, Springer. Available at `http://dx.doi.org/10.1007/11618027`.

[44] Leslie Lamport (1995): *How to write a proof*. *American Mathematical Monthly* 102(7), pp. 600–608. Available at `http://dx.doi.org/10.2307/2974556`.

[45] Chee-Kit Looi, Gordon I. McCalla, Bert Bredeweg & Joost Breuker, editors (2005): *Artificial Intelligence in Education - Supporting Learning through Intelligent and Socially Informed Technology, Proceedings of the 12th International Conference on Artificial Intelligence in Education, AIED 2005, July 18-22, 2005, Amsterdam, The Netherlands*. *Frontiers in Artificial Intelligence and Applications* 125, IOS Press.

[46] Noboru Matsuda & Kurt VanLehn (2005): *Advanced Geometry Tutor: An intelligent tutor that teaches proof-writing with construction*. In Looi et al. [45], pp. 443–450.

[47] William McCune (2003): *Mace4 Reference Manual and Guide*. *CoRR* cs.SC/0310055. Available at `http://arxiv.org/abs/cs.SC/0310055`.

[48] Bruce M. McLaren, Sung-Joo Lim & Kenneth R. Koedinger (2008): *When and how often should worked examples be given to students? New results and a summary of the current state of research*. In: *Proceedings of the 30th Annual Conference of the Cognitive Science Society*, Citeseer, pp. 2176–2181.

[49] 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).

[50] Douglas C. Merrill, Brian J. Reiser, Michael Ranney & J. Gregory Trafton (1992): *Effective Tutoring Techniques: A Comparison of Human Tutors and Intelligent Tutoring Systems*. *The Journal of the Learning Sciences* 2(3), pp. 277–305. Available at http://dx.doi.org/10.1207/s15327809jls0203_2.

[51] Dale A. Miller (1984): *Expansion Tree Proofs and Their Conversion to Natural Deduction Proofs*. In Robert E. Shostak, editor: *7th International Conference on Automated Deduction, Napa, California, USA, May 14-16, 1984, Proceedings*, *Lecture Notes in Computer Science* 170, Springer, pp. 375–393. Available at http://dx.doi.org/10.1007/BFb0047132.

[52] Allen. Newell & Herbert A. Simon (1972): *Human Problem Solving*. Prentice Hall, Englewood Cliffs.

[53] Jean-Franois Nicaud, Denis Bouhineau & Thomas Huguet (2002): *The Aplusix-Editor: A New Kind of Software for the Learning of Algebra*. In Cerri et al. [21], pp. 178–187. Available at http://dx.doi.org/10.1007/3-540-47987-2_22.

[54] Stellan Ohlsson (1996): *Learning from performance errors*. *Psychological Review* 103, pp. 241–262. Available at http://dx.doi.org/10.1037/0033-295X.103.2.241.

[55] Paulo Ribenboim (1996): *The New Book of Prime Number Records*. Springer.

[56] Peter J. Robinson & John Staples (1993): *Formalizing a Hierarchical Structure of Practical Mathematical Reasoning*. *J. Log. Comput.* 3(1), pp. 47–61. Available at http://dx.doi.org/10.1093/logcom/3.1.47.

[57] Carolyn Penstein Rosé, Johanna D. Moore, Kurt Vanlehn & David Allbritton (2001): *A comparative evaluation of socratic versus didactic tutoring*. In: *Society, University of Edinburgh*, pp. 869–874.

[58] Marvin Schiller (2011): *Granularity Analysis for Tutoring Mathematical Proofs*. AKA Verlag, Heidelberg.

[59] Marvin Schiller, Christoph Benzmüller & Ann Van de Veire (2006): *Judging Granularity for Automated Mathematics Teaching*. In: *LPAR 2006 Short Papers Proceedings*, Phnom Pehn, Cambodia.

[60] Wilfried Sieg (2007): *The AProS Project: Strategic Thinking & Computational Logic*. *Logic Journal of the IGPL* 15(4), pp. 359–368. Available at http://dx.doi.org/10.1093/jigpal/jzm026.

[61] Wilfried Sieg & Richard Scheines (1994): *Computer Environments for Proof Construction*. *Interactive Learning Environments* 4(2), pp. 159–169. Available at http://dx.doi.org/10.1080/1049482940040203.

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

[63] Richard Sommer & Gregory Nuckols (2004): *A Proof Environment for Teaching Mathematics*. *Journal of Automated Reasoning* 32(3), pp. 227–258. Available at http://dx.doi.org/10.1007/s10817-004-5097-z.

[64] Pramuditha Suraweera & Antonija Mitrovic (2002): *KERMIT: A Constraint-Based Tutor for Database Modeling*. In Cerri et al. [21], pp. 377–387. Available at http://dx.doi.org/10.1007/3-540-47987-2_41.

[65] Don Syme (1999): *Three Tactic Theorem Proving*. In Bertot et al. [18], pp. 203–220. Available at http://dx.doi.org/10.1007/3-540-48256-3_14.

[66] Donald Syme (1997): *DECLARE: a prototype declarative proof system for higher order logic*. Technical Report UCAM-CL-TR-416, University of Cambridge.

[67] Jana Trgalova & Hamid Chaachoua (2009): *Automatic analysis of proof in a computer-based environment*. In F.-L. Lin, F.-J. Hsieh, G. Hanna & M. de Villiers, editors: *Proof and Proving in Mathematics Education, ICMI'19: Proceedings of the 19th International Conference on Mathematics Instruction, Taipei, Taiwan, May 10-15, 2009*, 1, pp. 226–231.

[68] Dimitra Tsovaltzi (2010): *MENON - Automating a Socratic Teaching Model for Mathematical Proofs*. Phd thesis, Universität des Saarlandes, Saarbrücken, Germany.

[69] Kurt VanLehn (2006): *The Behavior of Tutoring Systems*. *I. J. Artificial Intelligence in Education* 16(3), pp. 227–265. Available at http://iospress.metapress.com/content/al6r85mm7c6qf7dr/.

[70] Kurt VanLehn (2011): *The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems*. *Educational Psychologist* 46(4), p. 197221. Available at http://dx.doi.org/10.1080/00461520.2011.611369.

[71] Kurt VanLehn, Collin Lynch, Kay G. Schulze, Joel A. Shapiro, Robert Shelby, Linwood Taylor, Donald Treacy, Anders Weinstein & Mary Wintersgill (2005): *The Andes Physics Tutoring System: Lessons Learned*. *I. J. Artificial Intelligence in Education* 15(3), pp. 147–204. Available at http://iospress.metapress.com/content/4qh80ubfdft0g4yr/.

[72] Markus Wenzel (1999): *Isar - A Generic Interpretative Approach to Readable Formal Proof Documents*. In Bertot et al. [18], pp. 167–184. Available at http://dx.doi.org/10.1007/3-540-48256-3_12.

[73] Freek Wiedijk (2004): *Formal Proof Sketches*. In Mario Coppo Stefano Berardi & Ferruccio Damiani, editors: *Types for Proofs and Programs: Third International Workshop TYPES 2003*, *LNCS* 3085, Springer, Torino, pp. 378–393. Available at http://dx.doi.org/10.1007/978-3-540-24849-1_24.

[74] Magdalena Wolska, Mark Buckley, Helmut Horacek, Ivana Kruijff-Korbayov & Manfred Pinkal (2010): *Linguistic Processing in a Mathematics Tutoring System: Cooperative Input Interpretation and Dialogue Modelling*. In Matthew W. Crocker & Jrg Siekmann, editors: *Resource-Adaptive Cognitive Processes*, Cognitive Technologies, Springer Berlin Heidelberg, pp. 267–289. Available at http://dx.doi.org/10.1007/978-3-540-89408-7_12.

[75] Beverly Park Woolf (2008): *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning*. Morgan Kaufmann.