6/2 (2008), 345–362
tmcs@inf.unideb.hu
http://tmcs.math.klte.hu

Teaching
Mathematics and
Computer Science

Verification of human-level proof steps in mathematics education

Dominik Dietrich and Mark Buckley

Abstract. Automated mathematics tutorial systems need support from a reasoning module which can verify the correctness of students' contributions. However, current systems typically do not reason at a level similar to the student's reasoning level, and do not fully account for underspecified or ambiguous inputs. We present a domain-independent method for automatically verifying correct proof steps and detecting standard reasoning errors. We use a depth limited BFS proof search to determine and maintain multiple possible interpretations consistent with the given proof step, we are able to resolve or otherwise propagate underspecification and ambiguity which occurs due to unrestricted user input. Our approach has been implemented in ΩMEGA$^{\text{CoRe}}$.

Key words and phrases: proof tutoring, automated reasoning, proof checking, proof reconstruction, mathematics education.

ZDM Subject Classification: E55, R45, U55.

1. Introduction

In recent years a number of software tools have emerged which support automatically tutoring mathematics. In particular, tools based on computer algebra systems have been successfully used [22]. It has however been recognized that in addition to computational tasks, students must learn how to conduct mathematical argumentation and mathematical proofs [13]. Therefore tools have been developed which focus on teaching proving skills. For propositional and first order logic there are the CMU proof tutor [23], ProofEasy [8], alfie [28], Proofweb [18], Jape [27] and WinKE [9], and for higher-order logic, ETPS [2]. These systems focus on pure logic and support proof construction using for example Fitch-style diagrams or trees. Proof construction works essentially as follows: The student selects a formula to be modified and can then try to apply a deduction rule – mostly without having to state the result of the application of the rule.

A drawback of these types of system is that students must first learn the commands and input syntax to be able to interact with the system. Furthermore, students are able to construct a proof by randomly applying rules, without getting a deeper understanding of the meaning of these rules, which are at a very low level. In this regard it has also been shown that logic alone does not help humans do more abstract proofs [12].

Consequently some tutorial systems support teaching mathematics at a more abstract level, comparable to the type of mathematics taught in schools. These include the EPGY Theorem Proving Environment [25], the Geometry Tutors [19] and Tutch [1]. To teach this kind of mathematics, a tutor system must be able to interact with the student at a level of reasoning similar to that which the student uses to perform proofs. The student should be free to express mathematical arguments in a natural way, and not be restricted, for instance, to rule applications in the logic. Abel *et al.* also argue for abstracting away from the detailed logic level: “Larger proofs ... were tedious since each step had to be a single natural deduction inference. For practical reasoning this restriction is highly unsatisfactory...” [1]. Huang [17] finds that human proofs are justified at the so-called *assertion level*, that is by citing axioms, definitions, or theorems, or on the proof level, such as “by analogy”. Therefore the tutorial system must be able to reason at this abstract level. Allowing the student such freedom puts demands on the tutorial system’s ability to verify the correctness of student inputs. We argue that only a fully-fledged automated theorem prover (ATP) can offer sufficient support. However, closing the gap between abstract, human level reasoning and machine-oriented reasoning is a difficult task.

Another motivation for high-level reasoning is efficiency: Resolution of ambiguity and underspecification seems easier to conduct at the abstract rather than at the detailled level. This is because each choice at the abstract level usually corresponds to several choices at a lower level. The use of uninformed search techniques at the low level to analyse and reconstruct abstract steps can thus quickly lead to unacceptable system response times for complex problems. Also, shallow methods, such as edit distances, cannot hope to capture the fine grained distinctions necessary to verify formal correctness.

In this paper we describe a method of verifying students' proof steps using the mathematical assistance system ΩMEGA$^{\text{CoRe}}$. We apply a depth-limited BFS proof search operating directly at Huang's assertion level and a filter to determine possible consistent cognitive proof states of the student, which are then maintained in ΩMEGA$^{\text{CoRe}}$'s proof data structure. Each cognitive proof state is the representation of one possible interpretation of the proof steps performed so far. Stated simply, a proof step is considered correct if at least one current cognitive proof state can be extended in a consistent way to include the step.

An essential feature of our approach is that we can resolve the ambiguity and underspecification which occurs due to the unrestricted nature of the student's proof steps. Ambiguities which can not immediately be resolved are propagated as parallel cognitive proof states until enough information is available for their resolution. Our approach is domain independent, that is, proof steps can be verified in any mathematical domain which has been formalised in a standard specification language supported by ΩMEGA$^{\text{CoRe}}$. This removes the need for pre-authoring of solution graphs and also allows for unforeseen solutions. Proof steps of varying length can be verified and a measure of their size can be computed. Note that this requires a close correspondence between the student's step and its formal counterpart. In addition, the formal proof object which is incrementally built during the proving session is available to the tutorial environment.

The structure of this paper is as follows: Section 2 introduces our corpus of human/human tutorial dialogue from which we take our data and gives some examples. We briefly present Huang's assertion level in Section 3 before describing the ΩMEGA$^{\text{CoRe}}$ mathematical assistance system in Section 4. Section 5, the core of the paper, shows how we represent and maintain the student's cognitive proof state and use it as the context to verify proof steps. In Section 6 we give example verification followed by a sketch of how to deal with erroneous proof steps in Section 7. Section 8 presents some evaluation data, Section 9 mentions some related work and Section 10 concludes the paper.

2. Tutorial dialogue on mathematical proofs

The framework in which our research is being carried out is the DIALOG project [5], which has the final goal of natural tutorial dialogue between a student and a mathematical assistance system. The student's task is to build a proof by performing natural language utterances which may contain proof steps. Each correct proof step extends the current partial proof. In the wider scope of the project, domain-specific hints are generated when the student is stuck or shows non-understanding of domain concepts. A natural language processing module performs an analysis of student utterances and a tutorial module decides on pedagogical strategies.

2.1. A corpus of mathematical tutorial dialogues

Our approach to the verification of proof steps is motivated by phenomena found in a corpus of tutorial dialogues, collected in the Wizard-of-Oz paradigm, between students and experienced mathematics teachers [11]. The goal of the experiment was to collect data on the use of natural and mathematical language in a tutorial interaction and on the behaviour of students and tutors with respect to the theorem proving task. After having seen some preparatory material introducing the theory of binary relations, students were asked to solve four exercises in a session with the tutorial system. No restrictions were made with respect to the type of language the student were allowed to use. Tutors rated each proof step with respect to correctness, granularity (or proof step size) and relevance to the current task. In the examples **S** refers to a student turn and **T** to a tutor turn.

Figure 1 shows a fragment of a tutorial session in which the student has been instructed to prove the theorem $(R \circ S)^{-1} = \left(S^{-1} \circ R^{-1}\right)$, where R and S are relations.

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

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 recognises 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 alternatives 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.

2.2. Phenomena observed in the corpus

We analysed those utterances from the corpus which contain contributions to the theorem proving task. We were able to identify a number of 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.

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 extentionality, but the rule is not stated explicitly. The student also does not specify that of the two subgoals introduced by set extentionality, he is now proving the subset direction, nor does he specify that there is a second subgoal. Further, the number of steps needed to reach this proof state is not given. Part of the task of analysing 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 the utterance **S8** is clearly a contribution to the proof, but since the step only introduces a new variable binding, there is no assertion whose truth value can be checked. However, anticipating that the student is using the definition of subset $A \subset B \Leftrightarrow x \in A \Rightarrow x \in B$ allows us to determine that the new variable binding is useful. Utterance **S8b** is also a correct contribution, 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.

These examples are evidence that verification in this scenario is not simply a matter of logical correctness, but must take into account for instance proof context.

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 natural language expressions, such as "hence" or "conjecture", it is not clear whether the formula is a newly derived fact or a newly introduced conjecture. 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 propagated. This must be done by maintaining multiple parallel interpretations, which are retained until enough information has been provided by the proof context to decide whether they are still consistent.

2.3. Linguistic markers of proof step type

The analysis of the students' natural language utterances is a very challenging area which we will not consider in this paper, however we refer to [16]. We assume that the NLP module can deliver the formula that the student intended to utter.

The corpus shows that proof steps are embedded in utterances which carry information about the type of the proof step which is important for its verification, such as the phrase "we consider the subgoal" in utterance **S8b** in Figure 1. The types of proof step we consider are `let`, which introduces new variables, `hence`, which indicates a forward reasoning step, `subgoal`, which indicates a reduction of a goal to subgoals, and `assume`, to make an assumption necessary to conduct a proof by contradiction, and `conjecture` to state a lemma. We additionally use `done` to claim that a proof goal has been closed and `back` to undo a proof step. Thus we assume our input to be a pair $\langle c, \{f_1 \dots f_2\}\rangle$ where c is a proof step type and f_i are formulas.

3. The assertion level

Once students have attained a good background in logic and start to prove more advanced theorems such as simple properties about sets and relations, it is no longer feasible to communicate the complete proof as a sequence of natural deduction inferences (c.f. [1]). Rather, students express their deduction steps as shown in the examples in Figure 1, in a style which comes close to the proof representation in mathematical textbooks. Huang analyses in [17] that human proofs are justified at the so-called *assertion level*, that is by citing axioms, definitions, or theorems, or on the proof level, such as "by analogy". To illustrate the difference between a typical proof step from a textbook and its formal counterpart in natural deduction consider the following example:

Given the definition of a subset

$$\forall U, V\ .U \subset V \Leftrightarrow \forall x.x \in U \Rightarrow x \in V \tag{1}$$

an assertion step consists of deriving $a_1 \in V_1$ from $U_1 \subset V_1$ and $a_1 \in U_1$. The corresponding natural deduction proof is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall U,V.\,U\subset V \Leftrightarrow \forall x.x\in U\Rightarrow x\in V}{\forall V.\,U_1\subset V_1 \Leftrightarrow \forall x \in U_1 \Rightarrow x\in V_1}\forall_E}{U_1\subset V_1\Leftrightarrow \forall x.x\in U_1\Rightarrow x\in V_1}\forall_E}{U_1\subset V_1\Rightarrow \forall x.x\in U_1\Rightarrow x\in V_1}\Leftrightarrow_E \qquad U_1\subset V_1}{\forall x.x\in U_1\Rightarrow x\in V_1}\Rightarrow_E}{a_1\in U_1\Rightarrow a_1\in V_1}\forall_E \qquad a_1\in U_1}{a_1\in V}\Rightarrow_E \tag{2}$$

Even though natural deduction proofs are far more readable than proofs in machine oriented formalisms such as resolution, we see that they are still at a much lower level than proofs typically found in mathematical textbooks. In the example above, a single assertion step corresponds to 7 steps in the natural deduction calculus. This is mainly because each natural deduction rule stands for a simple syntactical manipulation. It is clear that for teaching advanced techniques, this level of interaction is far too low and too tedious, and that the assertion level is much more suitable.

4. The ΩMEGA$^{\text{CoRe}}$ prover

ΩMEGA$^{\text{CoRe}}$ [24], a mathematical assistance environment comprising an interactive proof assistant, a proof planner, a structured knowledge base, a graphical user interface, access to external reasoners, etc., has been under development since the early 90's at Saarland University. Similar to HOL4, Isabelle/HOL, Coq, or Mizar, the overall goal of the project is to develop a system platform for formal methods (not only) in mathematics and computer science. In ΩMEGA$^{\text{CoRe}}$, user and system interact in order to produce verifiable and trusted proofs.

ΩMEGA$^{\text{CoRe}}$ is based on Autexier's CoRe calculus [3], which has been transformed into an assertion level inference mechanism [10]. CoRe and our assertion level inference mechanism are (higher-order) variants of the deep inference approach[1], that is, they support deductions deeply inside a given formula without requiring preceding structural decompositions as needed in ND (or sequent calculus). In ΩMEGA$^{\text{CoRe}}$ we thus have a smaller "distance" between abstract level proofs and their expansions to the verifiable assertion level. Most importantly, it supports reasoning directly at the assertion level, that is the proof step given in formula (1) corresponds to one inference step in the ΩMEGA$^{\text{CoRe}}$ system.

Proof representation. ΩMEGA$^{\text{CoRe}}$ provides an uniform abstract proof construction interface, the so called *task layer* [10], which is used by the interactive and automatic proof construction. The task layer uses a special tree-like proof data structure [4] (PDS) to maintain proof attempts. The nodes of the PDS are annotated with *tasks*, which are Gentzen-style multi-conclusion sequents augmented by a means of defining multiple foci of attention on subformulas that are maintained during the proof. Each subformula is annotated with a polarity $+$ or $-$. A subformula f of a formula F has negative (or positive) polarity if decomposing F introduces f on the left hand side (or right hand side) of the sequent.

Proof search is realised by reducing a task to a possibly empty set of subtasks by transformation rules which can be applied to subformulas. The basic transformation rules are *inferences*, representing the application of theorems, definitions, and axioms[2].

Each proof attempt is represented by an agenda. It maintains a set of subproblems which need to be solved to finish the proof of the overall conjecture the user wants to show. Formally an agenda is a triple $\mathcal{A} = \langle T_1, \ldots, T_n; \sigma; T_j \rangle$ where $T_1, \ldots, T_n$ are tasks, σ is a substitution instantiating meta-variables of the task, and $T_j =: \mathit{current}(\mathcal{A})$ is the task the user is currently working on. We will use the notation $\langle T_1, \ldots, T_{j-1}, \underline{T_j}, T_{j+1} \ldots T_n; \sigma \rangle$ to denote that the task T_j is the current task. The set of agendas in the PDS is denoted by $\{\mathcal{A}_1, \ldots, \mathcal{A}_m\} = \vec{\mathcal{A}}$.

Automation in Ωmega$^{\text{CoRe}}$. Automation of proof search is realised in two components: an extension of the multi-strategy proof planner MULTI [21] and the resource-guided agent-based reasoning system ΩANTS [26]. For the purpose of this paper we focus on MULTI, which integrates a basic set of algorithms

[1]`http://alessio.guglielmi.name/res/cos/index.html`

[2]Note that we employ the term inference in its general meaning; Taken in this sense an inference can be either valid or invalid, in contrast to the formal logic notion of an *inference rule*.

parametrised over strategic information. There are several search strategies implemented. Among others, it includes a breadth-first proof planning algorithm that is parametrised over inferences and control rules and which searches through the space of applicable inferences by using the heuristic function defined by the provided control knowledge.

5. Maintaining the student’s cognitive proof state

In this section we show how the $\Omega\text{MEGA}^{\text{CoRe}}$ prover can be used to (i) represent possible cognitive states the student might be in, and (ii) to judge about the correctness of a proof step of the student. We assume that tutoring takes place in a mathematical domain, which we define to be a tuple $\langle \mathcal{I}, p\rangle$ where $\mathcal{I}$ is a set of formulas representing axioms, definitions, and theorems of the current theory and p is the theorem to be proved.

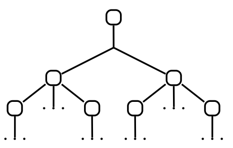
5.1. Representing the possible cognitive proof states

The PDS is used to simultaneously represent all of the possible cognitive proof states the student might be in. More specifically each agenda determines one possible cognitive proof state. Given a problem and a theory the inferences are automatically constructed from the formulas $\mathcal{I}$ and the *initial PDS* is constructed. The initial PDS consists of one initial task $T = \Gamma \vdash \Delta$ where Γ contains the assumptions of the problem and Δ is the proof problem to be shown. The *initial cognitive proof state* is determined by the *initial agenda*, containing only the initial task: $\mathcal{A} = \langle \underline{T}; \emptyset\rangle$

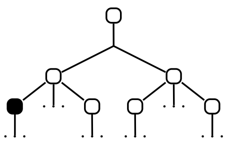
5.2. Updating the cognitive proof states

Given a set of possible cognitive proof states $\vec{\mathcal{A}}$ and a preprocessed utterance $s = \langle c, \{f_1, \dots, f_n\}\rangle$, we must determine all possible successor states which are consistent with the utterance s. For the sake of simplicity we only show how to determine the successor states of a single cognitive proof state $\mathcal{A} \in \vec{\mathcal{A}}$, denoted by $\varphi(\mathcal{A})$. Combining the result for each $\mathcal{A} \in \vec{\mathcal{A}}$ gives the complete set of successor states. If no agenda can provide a consistent successor state, i.e. $\cup_{\mathcal{A}\in\vec{\mathcal{A}}}\varphi(\mathcal{A}) = \emptyset$, the step is classified to be *incorrect*.

Given a pair $\langle c, \{f_1, \ldots, f_n\}\rangle$ and an agenda $\mathcal{A}$, $\varphi(\mathcal{A})$ is determined in four steps: (i) A depth limited BFS search is performed at *current*($\mathcal{A}$)[3] (including some control knowledge to cut off irrelevant branches of the search space), resulting in a set of successor nodes, (ii) from this, consistent successor nodes are selected and partial agendas are created from them, (iii) the partial agendas are completed, and finally (iv) the PDS is cleaned. The overall result is a confirmation of whether the step could be verified, along with the side-effect that the PDS has been updated to contain exactly the possible cognitive proof states resulting from the performance of the step. We now illustrate each step of the algorithm schematically.



Step (i). The node *current*($\mathcal{A}$) is expanded using a depth limited BFS search. The depth limiter specifies how many calculus steps[4] the student is allowed to perform implicitly. Intuitively we can think of this bound as reflecting the experience of the student, consequently it should be based on a student model. This bound is needed to guarantee termination of the verification algorithm, which might otherwise not terminate for a faulty step even if we assume a complete calculus.



Step (ii). Whereas step (i), (iii), and (iv) are independent of the proof step type c, step (ii) differs for each c. Therefore we introduce a filter function Θ_c to filter consistent cognitive successor states for each type c. We apply this filter function Θ_c to get those successor states which possibly represent the student's cognitive state after applying the utterance. Note that in general there may be several successor states satisfying Θ_c. To further restrict the consistent successor nodes only those with minimal distance to the expansion node are stored. For the types introduced in Section 2.3 we define the following filters:

let: A node is classified by Θ_{let} to be consistent if its corresponding task contains f as a subformula with negative polarity, no conditions and the free variables of f have been introduced. A negative subformula f of a formula F has no conditions if the decomposition of F to obtain f in the sequent calculus

[3]Note that without restriction we can assume *current*($\mathcal{A}$) $\neq \emptyset$ in (i) (otherwise we transform the $\mathcal{A}$ into a set of agendas representing all possibilities to select a task in $\mathcal{A}$).

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

introduces no branching and f occurs on the left hand side of the resulting sequent.

hence: A node is classified by Θ_{hence} to be consistent if its corresponding task contains f as a subformula with negative polarity and no conditions. The search is restricted in the sense that only new facts are derived during the expansion of the agenda.

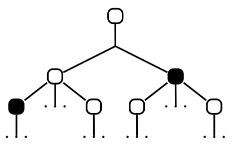
subgoal: Given the input $\langle \text{subgoal}, \{f_1, \ldots, f_n\}\rangle$ the subgoal filter tries to find successor nodes $s_1, \ldots, s_n$ in different branches such that each f_i occurs as a positive subformula in s_i. During the expansion, only reduction steps are allowed, i.e. steps that reduce the current goal.

assume: Assume works similarly to let except that the PDS is searched for a positive subformula where f is negated, i.e. $\neg f$ or f' with $\neg f' = f$.

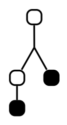
conjecture: Conjecture starts a new proof tree and tries to prove the conjecture using the currently available knowledge. If the conjecture could be proved, it is inserted into the knowledge base and hence available for the main proof.

done: Done is used to indicate that the proof is completed. It checks whether the current goal(s) can be closed.

Step (iii). The agendas obtained from step (ii) can be partial, i.e. not all subgoals that must be solved are in the agenda. Such situations occur if the prover reduces a task to more than one subtask but the Θ_c does not select a node from each branch. In this case the user has not modified any of the missing subtasks. Hence it is reasonable to extend the partial agenda by the tasks introduced by the reduction. We extend the agenda as shown on the right.



Step (iv). Usually there will be many nodes which are generated by the search but are rejected by the filter. All of these nodes are removed from the PDS.



6. Example verification

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 cognitive proof state and finishing with the cognitive proof state extended by the proof step. The initial cognitive proof state is $\{\langle \underline{\vdash (R \circ S)^{-1} = S^{-1} \circ R^{-1}}; \emptyset\rangle\}$ and the proof step to be verified is $\langle \text{let}, \{(x, y) \in (R \circ S)^{-1}\}\rangle$.

Having expanded the current task (step (i), shown in Figure 2), we apply the filter Θ_{let} to find the set of newly-created tasks which are consistent with the given proof step.

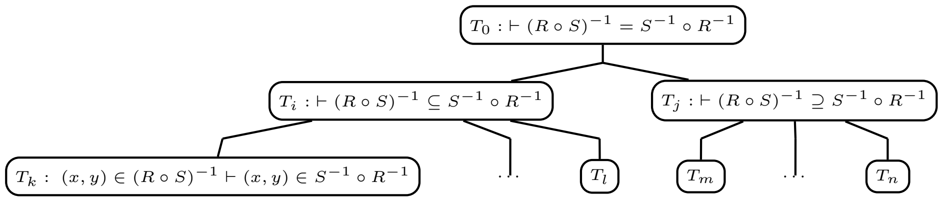

$T_0 : \vdash (R \circ S)^{-1} = S^{-1} \circ R^{-1}$
$T_i : \vdash (R \circ S)^{-1} \subseteq S^{-1} \circ R^{-1}$
$T_j : \vdash (R \circ S)^{-1} \supseteq S^{-1} \circ R^{-1}$
$T_k : (x, y) \in (R \circ S)^{-1} \vdash (x, y) \in S^{-1} \circ R^{-1}$
...
T_l
T_m
...
T_n

Figure 2. The expanded task after step (i) of verification (abbreviated).

Of the tasks in the tree, only the node containing the task 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 tasks, we must complete any partial agendas (step (iii)). Because the decomposition of the task T_0 introduced a subgoal split, the task T_j must be proved in addition to T_k. The resulting agenda is therefore $\{\langle \underline{T_k}, T_j; \emptyset\rangle\}$, that is, T_k is the now the current task, and T_j is still to be proved. Finally in step (iv), we prune the nodes which were rejected by the filter, resulting in the proof state shown in Figure 3. This becomes the initial state for the verification of the student's next proof step.

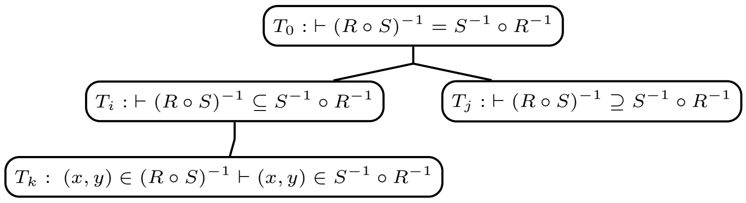

$T_0 : \vdash (R \circ S)^{-1} = S^{-1} \circ R^{-1}$
$T_i : \vdash (R \circ S)^{-1} \subseteq S^{-1} \circ R^{-1}$
$T_j : \vdash (R \circ S)^{-1} \supseteq S^{-1} \circ R^{-1}$
$T_k : (x, y) \in (R \circ S)^{-1} \vdash (x, y) \in S^{-1} \circ R^{-1}$

Figure 3. The resulting proof state after verification.

This example has illustrated the verification algorithm for a single agenda with a single successor state. For ambiguous steps, that is steps which result in more than one successor task after the filter in step (ii), we simply create a new agenda for each successor, and maintain these in the set of agendas $\vec{\mathcal{A}}$. Step (iii) is done for each new agenda separately, and only those nodes which do not occur in any agenda are pruned in step (iv). An example of this is the verification of proof step **S8a**. Here two successor agendas are created, $\{\langle \underline{T_i}, T_j; \emptyset\rangle\}$ and $\{\langle T_i, \underline{T_j}; \emptyset\rangle\}$, differing only in which of the two tasks is the current task.

For the case in which the current proof state already contains multiple agendas, the entire verification process is carried out for each in turn. This is the point at which ambiguities introduced in previous steps can be resolved: those agendas which do not lead to successor states are deleted, and those which have successors are maintained. For example, any proof step following **S8a** will belong to either the left or right branch of the proof only, and will have no successor tasks in the other branch. Thus the agenda in which the "wrong" current task had been chosen will be deleted.

7. Error detection

When no cognitive proof state consistent with the filter corresponding to the input proof step can be derived within n steps then the system is unable to verify a proof step. There are two possible causes: (i) The step was logically correct, but needs more than n deduction steps to be verified, (ii) the step was wrong. In both cases the prover can deliver the valuable information that it was unable to verify the proof step to the tutorial system. To also detect standard errors and to be able to inform the tutoring system about them we extend the current theory $\mathcal{T} = \langle \mathcal{I}, p\rangle$ to $\mathcal{T}' = \langle \mathcal{I}, \mathcal{I}_f, p\rangle$ where $\mathcal{I}_f$ are inferences representing typical errors in the domain. When the filter Θ_c does not find any consistent successor states in step (ii) of verification, the prover attempts to verify the proof step using the extended set of rules $\mathcal{I} \cup \mathcal{I}_f$. If this is successful then the student has made a mistake, and the prover can report the concrete error.

In the case of utterance **S10** in Figure 1, the prover will not find any consistent successor states for the call $\langle \texttt{hence}, (s, r) \in R \circ S\rangle$ starting from the proof state generated by $\langle \texttt{let}, (s, r) \in (R \circ S)^{-1}\rangle$. However by adding either the inference $(x, y) = (y, x)$ or $(x, y) \in R \Leftrightarrow (x, y) \in R^{-1}$ to the set of rules, consistent successor nodes can be found. The tutorial environment can then check in the proof object which inference from $\mathcal{I}_f$ has been used, and offer suitable feedback.

8. Evaluation

We have evaluated our verification module with 17 tutorial dialogues taken from the Wizard-of-Oz corpus described in Section 2, containing a total of 144 proof steps. The steps within a single dialogue are passed to the verification module sequentially until a step that is labelled as correct cannot be verified (using a proof search depth of four), in which case we move on to the next dialogue. We then compared the results of the automated proof step analysis with the original correctness judgements given by the tutors.

Of the 116 correct steps, 113 (97.4%) were correctly verified and we correctly classify 141 out of the 144 steps (97.9%) as correct or wrong. For the remaining three steps the verification fails. This is due to our restriction of the search space to either forward or backward search (at the assertion level) for efficiency reasons. These steps are not captured by the current filter functions because they require a mixture of both. It would be straightforward to allow such a mixture, and we could easily implement a new filter function which would capture these steps. However, we feel that these steps are exceptions and decided for the more efficient variant. All steps in the example dialogue (c.f. Figure 4) are correctly classified as valid by our verification module (used with proof depth four), taking approximately 13.2 seconds on a standard PC.

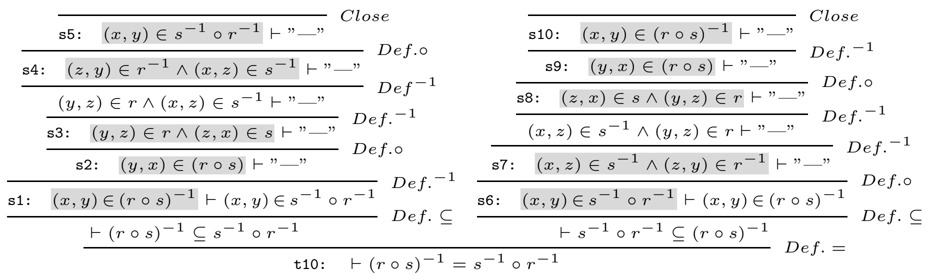

Close
s5: $(x,y) \in s^{-1} \circ r^{-1} \vdash$ ”—”
Def.∘
s4: $(z,y) \in r^{-1} \wedge (x,z) \in s^{-1} \vdash$ ”—”
Def^{-1}
$(y,z) \in r \wedge (x,z) \in s^{-1} \vdash$ ”—”
Def.$^{-1}$
s3: $(y,z) \in r \wedge (z,x) \in s \vdash$ ”—”
Def.∘
s2: $(y,x) \in (r \circ s) \vdash$ ”—”
Def.$^{-1}$
s1: $(x,y) \in (r \circ s)^{-1} \vdash (x,y) \in s^{-1} \circ r^{-1}$
Def. ⊆
$\vdash (r \circ s)^{-1} \subseteq s^{-1} \circ r^{-1}$
Close
s10: $(x,y) \in (r \circ s)^{-1} \vdash$ ”—”
Def.$^{-1}$
s9: $(y,x) \in (r \circ s) \vdash$ ”—”
Def.∘
s8: $(z,x) \in s \wedge (y,z) \in r \vdash$ ”—”
Def.$^{-1}$
$(x,z) \in s^{-1} \wedge (y,z) \in r \vdash$ ”—”
Def.$^{-1}$
s7: $(x,z) \in s^{-1} \wedge (z,y) \in r^{-1} \vdash$ ”—”
Def.∘
s6: $(x,y) \in s^{-1} \circ r^{-1} \vdash (x,y) \in (r \circ s)^{-1}$
Def. ⊆
$\vdash s^{-1} \circ r^{-1} \subseteq (r \circ s)^{-1}$
Def. =
t10: $\vdash (r \circ s)^{-1} = s^{-1} \circ r^{-1}$

Figure 4. Annotated ΩMEGA$^{\text{CoRe}}$ assertion level proof for the example dialogue.

The main reason for the efficiency and the simplicity of our algorithm lies in the fact that we directly search at the assertion level. Consequently, a small search depth is sufficient to verify a correct proof step. Indeed, a search depth of four already suffices to obtain the results shown above. Most importantly, our proof representation is very close to the proof representation of the student, as illustrated in Figure 4. Here, the formulas given by the student are shaded. The number of assertion level steps required (13, excluding the automatic *Close* steps) is still comparable to the number of proof steps as uttered by the student in the original dialogue (10), which provides evidence that the ΩMEGA$^{\text{CoRe}}$ assertion level proof is at a suitable level of granularity. In particular this allows for a further analysis of proof given by the student.

9. Related work

Some of the wider challenges for domain reasoning in the context of tutoring mathematical proofs, for instance dealing with ambiguity and underspecification, have been highlighted in an analysis of an earlier corpus collected within the DIALOG project [6]. We show a possible approach to accounting for these challenges.

Our approach has some similarities to a number of implemented systems. The SLOPERT system [30] operationalises human tutoring on differentiation problems, using a task model for calculating derivatives and a set of buggy rules that model typical errors. The mathematical assistance system Scunak [7] implements a domain-independent approach to verifying proof steps in textbook proofs which maintains multiple proof interpretations. Rather than using proof search, steps are considered correct if they have a continuation of at least one further proof step. Also related to our work are the EPGY Theorem Proving Environment [20], using Otter to justify or reject proof steps proposed to the environment, and Zinn's [29] computational framework for the analysis of textbook proofs.

By using existing mathematical knowledge to verify proof steps within a current proof situation, our approach differs from more traditional solution checking approaches in tutoring systems which match student contributions to pre-authored or pre-compiled answers [14], [15]. This is not suitable for complex mathematical domains, characterised by a possibly infinite solution space. We also directly account for underspecification and multiple interpretations of student proof step utterances. Our treatment of underspecification allows students to enter arbitrary formulas without the need to state which rules are needed for verification. Further, we construct one global, coherent proof object at the assertion level for each dialogue instead of just looking from step to step. The provision of a formal proof object allows further analysis of the granularity and relevance of proof steps, not just their correctness. Finally, there is a strong theorem prover working in the background.

10. Conclusions and further work

We have presented a domain-independent method of verifying correct proof steps and detecting typical errors performed by students interacting with a tutoring system for mathematical proofs in natural language. Our approach can be used in arbitrary mathematical domains without the need for pre-authoring solutions. We assume preprocessed input, however our input language can easily be extended by adding new filters Θ_c. Most importantly, we directly search for proofs at the assertion level, which enables us to employ a simple depth-limited breadth-first search algorithm in our proof step verification module. Interestingly, a depth limit of just four assertion level steps already enables our approach to correctly classify 95.9% of the proof steps in our corpus.

Future work will include utilising a student model to check whether the proof step is not only logically correct but also within the scope of the student’s current knowledge. We will also investigate verifying goal-directedness of proof steps. In order to offer further support to the tutoring system, we will use proof search to supply suggestions of meaningful next steps when the student gets stuck.

References

[1] A. Abel, B. Chang and F. Pfenning, *Human-readable machine-verifiable proofs for teaching constructive logic*, Proc. of the IJCAR Workshop on Proof Transformations, Proof Presentations and Complexity of Proofs (PTP01), 2001.

[2] P. B. Andrews, C. E. Brown, F. Pfenning, M. Bishop, S. Issar and H. Xi, A system to help students write formal proofs, *Journal of Automated Reasoning* **32** (2004), 75–92.

[3] S. Autexier, *The CORE Calculus*, Proc. of the 20th International Conference on Automated Deduction (CADE-20), vol. 3632, LNAI, (R. Nieuwenhuis, ed.), Springer, Tallinn, Estonia, 2005.

[4] S. Autexier, C. Benzmüller, D. Dietrich, A. Meier and C.-P. Wirth, *A Generic Modular Data Structure for Proof Attempts Alternating on Ideas and Granularity*, Proc. of MKM, Springer, Bremen, 2006.

[5] C. Benzmüller, H. Horacek, I. Kruijff-Korbayova, M. Pinkal, J. Siekmann and M. Wolska, *Natural Language Dialog with a Tutor System for Mathematical Proofs*, Cognitive Systems, vol. 4429, LNAI, (Ruqian Lu, Jörg Siekmann and Carsten Ullrich, eds.), Springer, 2007.

[6] C. Benzmüller and Q. B. Vo, *Mathematical Domain Reasoning Tasks in Natural Language Tutorial Dialog on Proofs*, Proc. of AAAI, AAAI Press / The MIT Press, Pittsburgh, 2005, 516–522.

[7] C. E. Brown, *Verifying and Invalidating Textbook Proofs using Scunak*, Proc. of MKM, Springer, Wokingham, 2006, 110–123.

[8] R. Burstall, Teaching people to write proofs, *CafeOBJ Symposium, Numazu, Japan* (1998).

[9] M. D’Agostino and U. Endriss, *WinKE: A Proof Assistant for Teaching Logic*, Proc. of the First International Workshop on Labelled Deduction, 1998.

[10] D. Dietrich, *The Task-Layer of the* ΩMEGA *System*, Diploma Thesis, FR 6.2 Informatik, Universität des Saarlandes, Saarbrücken, Germany, 2006.

[11] C. Benzmüller *et al.*, *A corpus of tutorial dialogs on theorem proving; the influence of the presentation of the study-material*, Proc. of LREC, ELDA, Genoa, 2006.

[12] S. S. Epp, The role of logic in teaching proof, *American Mathematical Monthly* **110**(10) (2003), 886–899.

[13] E. Klieme *et al.*, *The Development of National Educational Standards*, German Federal Minstry of Education and Research, Technical Report, 2004.

[14] A. C. Graesser, K. Wiemer-Hastings, P. Wiemer-Hastings and R. Kreuz, AutoTutor: A simulation of a human tutor, *Cognitive Systems Research* **1** (1999), 35–51.

[15] N. Heffernan and K. Koedinger, *Building a 3rd generation ITS for symbolization: Adding a Tutorial Model with Multiple Tutorial Strategies*, Proc. of the ITS Workshop in Algebra Learning, 2000.

[16] H. Horacek and M. Wolska, *Interpretation of Mixed Language Input in a Mathematics Tutoring System*, Proc. of AIED-05 Workshop on Mixed Language Explanations in Learning Environments, 2005, 27–34.

[17] X. Huang, *Reconstructing Proofs at the Assertion Level*, Proc. 12th CADE, (Alan Bundy, ed.), Springer – Verlag, 1994, 738–752.

[18] C. Kaliszyk, F. Wiedijk, M. Hendriks and F. van Raamsdonk, *Teaching logic usign a state-of-the-art proof assistant*, Proc. of the International Workshop on Proof Assistants and Types in Education, (H. Geuvers and P. Courtieu, ed.), 2007, 33–48.

[19] K. Koedinger and J. R. Anderson, *Reifying implicit planning in geometry: Guidelines for model-based intelligent tutoring system design*, Computers as Cognitive Tools, (S. P. Lajoie and S. J. Derry, eds.), 1993, 15–46.

[20] D. McMath, M. Rozenfeld and R. Sommer, *A Computer Environment for Writing Ordinary Mathematical Proofs*, LPAR '01: Proc. of the Artificial Intelligence on Logic for Programming, Springer – Verlag, London, UK, 2001, 507–516.

[21] A. Meier and E. Melis, *Multi: A Multi-strategy Proof-Planner*, Proc. of CADE-20, LNAI 3632, (R. Nieuwenhuis, ed.), Springer, Tallinn, Estonia, 2005.

[22] D. S. Scott, *Symbolic Computation and Teaching*, AISMC, of Lecture Notes in Computer Science, vol. 1138, (J. Calmet, J. A. Campbell and J. Pfalzgraf, eds.), Springer, 1996, 1–20.

[23] W. Sieg and R. Scheines, Computer Environments for Proof Construction, *Interactive Learning Environments* **4**(2), 159–169.

[24] J. Siekmann, C. Benzmüller and S. Autexier, Computer Supported Mathematics with ΩMEGA, *Journal of Applied Logic* **4**(4) (2006), 533–559.

[25] R. Sommer and G. Nuckols, A Proof Environment for Teaching Mathematics, *Journal of Automated Reasoning* **32**(3), Kluwer Academic Publishers, MA, USA (2004), 227–258.

[26] V. Sorge, *A Blackboard Architecture for the Integration of Reasoning Techniques into Proof Planning*, Ph.D thesisi, FR 6.2 Informatik, Universität des Saarlandes, Saarbrücken, Germany, 2001.

[27] B. Sufrin and R. Bornat, *User Interfaces for Generic Proof Assistants Part I: Interpreting Gestures*, Proc. of User Interfaces for Theorem Provers, (S. Aitken and P. Gray, eds.), 1996.

[28] B. von Sydow, *Alfie, a proof editor for propositional logic*, 2000.

[29] C. Zinn, Computational framework for understanding mathematical discourse, *Logic Journal of the IGPL* **11**(4) (2003), 457–484.

[30] C. Zinn, *Supporting Tutorial Feedback to Student Help Requests and Errors in Symbolic Differentiation*, Proc. of ITS, (M. Ikeda, K. D. Ashley and T.-W. Chan, eds.), Springer, 2006, 349–359.

DOMINIK DIETRICH
DEPARTMENT OF COMPUTER SCIENCE
SAARLAND UNIVERSITY
POSTFACH 151150
66041 SAARBRÜCKEN
GERMANY

E-mail: `dietrich@ags.uni-sb.de`

MARK BUCKLEY
DEPARTMENT OF COMPUTATIONAL LINGUISTICS
SAARLAND UNIVERSITY
POSTFACH 151150
66041 SAARBRÜCKEN
GERMANY

E-mail: `buckley@coli.uni-sb.de`

(Received September, 2007)