Adaptable Mixed-Initiative Proof Planning for Educational Interaction

Andreas Meier Erica Melis Martin Pollet

Fachbereich Informatik, Universität des Saarlandes,
and DFKI Saarbrücken,
66041 Saarbrücken, Germany
Email: email:{ameier|melis}@dfki.de, pollet@ags.uni-sb.de

Abstract

Today, most theorem proving systems are either used by their developers or by a (small) group of particularly trained and skilled users. In order to make theorem proving functionalities useful for a larger clientele we have to ask "What does an envisioned group of users need?"

For educational purposes a theorem prover can be used in different scenarios and can serve students with different needs. Therefore, the user interface as well as the choice of functionalities of the underlying prover have to be adapted to the context and the learner.

In this paper, we present proof planning as back-engine for interactive proof exercises as well as an interaction console, which is part of our graphical user interface. Based on the proof planning situation, the console offers suggestions for proof steps to the learner. These suggestions can dynamically be adapted, e.g., to the user and to pedagogical criteria using pedagogical knowledge on the creation and presentation of suggestions.

Key words: mathematics education, adaptive GUI, adaptive theorem proving

1 Motivation

So far, the main goal of developing automated theorem proving systems has been to output true/false for a statement formulated in some logic or to deliver a proof object. Interactive theorem proving systems aim to support the proof construction done by a user in different ways, they restrict the search space (the choices) by making valid suggestions for proof steps, they suggest applicable lemmas, or they produce a whole subproof automatically. These functionalities are useful, e.g., for checking a student's proof for validity or for verifying a program. They are not particularly helpful, when the goal is to

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

learn how to prove mathematical theorems with the assistance of a theorem proving system.

What is necessary in order to support learning mathematical proof?

Some usability features have been investigated, e.g., by cognitive science. Those studies [15] suggest that students' deficiencies in their mathematical competence with respect to understanding and generating proofs are connected with the shortcoming of students' self-guided explorative learning opportunities and the lack of (self-)explanations during problem solving. Such explorative learning can be supported by tools. Moreover, empirical results suggest that instruction with proof planning methods, which explicitly encode mathematical steps, can be a learning approach that is superior to the traditional learning of mathematical proving [8]. This motivates the integration of the proof planner ΩMEGA [13] with the user-adaptive learning environment ActiveMath [7]. However, the availability of the tool alone is insufficient for education because for effective learning more requirements have to be met.

In general, an educational context requires additional features (compared with a mere assisting system) such as

- adaptivity to the learner [4],
- allow for faulty proof attempts whose (coached) discovery and repair are a major source of learning [16], and
- feedback on the learner's activities.

Adaptation and personalization may effect the content, the possible or preferred problem solving strategies, the level of detail of proof, the appearance, etc. For instance, an exercise about the limit of a concrete sequence can be solved relying on the strategy to use only the definition of the limit or using the strategy to apply limit theorems. The choice of one of the proof ideas should depend on the learner's capabilities and knowledge. Another dimension which has to be adapted to the learner is the presentation of available methods. A less capable student will have more difficulties to choose between many methods in which case the likelihood of guessing increases. Adaption may also depend on the learner's activity history. For instance, the system should react when the student has recently not been able to apply that method several times.

It has been shown that successful learning does occur primarily, when the student discovers a failure or reaches an impasse and manages to recover from them [16]. Hence, for learning, a particularly interesting question is whether and how to employ a student's erroneous actions for learning from mistakes and how to react to errors. Erroneous situations or faulty suggestions are not appropriate for a system that is devised for problem solving assistance. For a learning tool, however, it might not be the best idea to always make only correct suggestions, because then the student might just click on suggestions

rather than learn anything. For instance, when the student should prove the limit of a sequence by applying theorems and she is already familiar with the theorems, then it would be too restrictive to suggest only applicable theorems. The decision on when to make which faulty suggestions will, of course, depend on the student's situation and on her capabilities, among others it depends on the student's learning goal, the learning context, her learning history, and the competency of the student (all represented in ActiveMath's student model) as well as on the pedagogical strategy. Therefore, suggestions should be dynamically generated depending on the student model and on the pedagogical strategy.

The structure of the article is as follows. After a brief introduction to proof planning and to ActiveMath in Section 2, we introduce in Section 3 four learning scenarios for the application of a proof planner. In Section 4, we focus on the realization of one scenario, the interactive proof planning scenario, and on examples of its usage. In particular, this section explains how exercises can be adapted to particular learners and how suggestions are presented in an interaction console. This console together with a multi-modal display of the partial proof plan is our current GUI for proof exercises. Section 5 contains a summary and future work.

2 Preliminaries

2.1 Proof Planning

For interactive proof exercises two functionalities of the ΩMEGA proof planner [13] are employed: automatic proof planning [9] and the support of interactive proof planning by an agent-based command suggestion mechanism [1]. In this section, we introduce both mechanisms separately. Later on (Section 4), we describe their combination.

2.1.1 Automatic Proof Planning

Proof planning in ΩMEGA was originally conceived for *automated* theorem proving and aims at reducing the search space by proving at the level of *methods*, which can encapsulate complex proof steps, and by introducing meta-level guidance [2,10].

Proof planning starts with a goal that represents the conjecture to be proved and with proof assumptions. It continues by applying methods for which the application conditions are satisfied and this generates new assumptions or reduces a goal to subgoals until no goal is left. The resulting sequence of instantiated methods constitutes a solution proof plan.

Generally, proof construction may require to construct mathematical objects, i.e., to instantiate existentially quantified variables by witness terms. In proof planning meta-variables are used as place holders for witness terms, and the planning process proceeds until enough information is collected to in-

stantiate the meta-variables. A domain-specific constraint solver can help to construct mathematical objects that are elements of a specific domain. During the proof planning process the constraint solver checks the (in)consistency of constraints on meta-variables and collects consistent constraints in a constraint store. Then, it computes instantiations for the meta-variables that satisfy the collected constraints [11].

To structure the repertoire of proof planning methods and make the proof planning process more hierarchical, strategies have been introduced [9]. A simple proof planning strategy is specified by a set of methods and search heuristics. Different proof planning strategies correspond to and implement different proof ideas.

The proof planner has an automatic and an interactive mode. In the automatic mode the proof planner searches for a solution proof plan, i.e., in each intermediate state it searches for applicable methods and valid instantiations. Mathematics-oriented heuristics guide this search. In interactive proof planning, the user has to make all the search decisions and these include the choice of strategies and methods and the instantiation of meta-variables.

2.1.2 Command Suggestion Mechanism

Originally, the command suggestion mechanism was developed to support a user during proof construction with tactics in ΩMEGA [1]. All user communication with the ΩMEGA system is performed via commands, among others, also the application of tactics to manipulate the proof plan under construction. The command suggestion mechanism computes and suggests commands, which specify *applicable* tactics, as well as all arguments necessary to apply the tactic to the current proof state, i.e., to which proof nodes and with which parameters. This frees the user from the search for applicable tactics.

The suggestion mechanism is implemented by agents. The command is associated with a tactic has arguments for the input of the tactic. For each argument there is one agent which checks whether there are suitable (i.e., matching) instantiations of the argument in the current proof context. The agents run concurrently and ΩMEGA displays a suggestion to the user as soon as an agent was successful.

For instance, consider the =Subst tactic, which applies a proof node with an equation $a = b$ to a proof node with the formula $F[a]$ (where a is a subterm of F) and derives a new proof node with the formula $F[b]$. Associated with this tactic is the command `=SUBST-COMMAND`. This command has the arguments 'proof node with equation to apply', 'proof node to apply equation to', and 'position to apply the equation to' (if there are several occurrences of a in $F[a]$ there are different possibilities to substitute a by b). For each of these arguments an agent can be specified, which searches in the current proof plan for suitable instantiations of the argument. For instance, the agent associated with the first argument searches for proof nodes with equations. When it finds instantiations it records them as partial suggestion for the command

=SUBST-COMMAND on a blackboard. The agents for the other two arguments become active as soon as there is a suggestion for the equation and then they try to find instantiations for their arguments, completing the suggestions for the command =SUBST-COMMAND. Partial suggestions are already presented to the user. When the user selects a partial suggestion, she has to complete the missing arguments.

The command suggestion mechanism is not restricted to the creation of suggestions of tactic applications. In Section 4, we describe how we use the same mechanism in order to create suggestions for method applications as well as suggestions for meta-variable instantiations. Moreover, note that it is not necessary that there are agents for each argument of a command applying a tactic (or a method). If an agent for an argument is missing, then all created suggestions are partial, and the user has to specify the missing parts.

2.2 ActiveMath *and its User Model*

ActiveMath is a web-based learning environment (for mathematics) [7]. It generically integrates several back-engines for exercising and exploratory learning – among them the computer algebra systems Maple and MuPad.

ActiveMath generates (mathematics) learning material user-adaptively, i.e., dependent on the learner's goals, learning scenarios, preferences, and mastery level. The adaptivity is based on a *student model* that includes

- the history of the learner's actions
- her preferences
- her mastery level of concepts and skills with a range between 0 and 1.

The student model is updated by diagnoses of the learning activities such as reading and problem solving. A student's exercise performance is evaluated and the evaluation is passed to the student model for updating it.

3 Educational Usage of the Proof Planner

Today's typical usage of interactive (and automated) theorem provers to prove (or disprove) a theorem and maybe store the completed proof is very different from an educational usage for learning mathematical proofs because

- proving should take place at a relatively abstract level rather than at the level of a logical calculus.
- several of the activities in previous theorem proving applications are not strictly necessary at the level of learning at school or university. For instance, loading a theory is left to more advanced levels such as theory development.
- some activities that are effective for learning should be enabled even though they are not available in typical theorem proving systems and their user

interfaces. This includes
· browsing method descriptions,
· browsing constraint states,
· requesting meta-level information,
· restricted suggestion of applicable steps,
· possible suggestion of steps, which are not applicable or not useful.

The reason for using proof planning in a learning environment for mathematics is that methods represent typical steps in mathematical reasoning. Therefore, the communication and teaching of those methods should enable a student to extend and improve their reasoning skills [6].

We identified several pedagogically motivated scenarios for educational applications of ΩMEGA that differ with respect to the overall learning goal and the employed functionalities of ΩMEGA.

Replay and presentation of a proof plan Existing proof plans (or parts of proof plans) are presented to the learner. The proof plan can be presented all at once or stepwise. This scenario targets an understanding of the effects of the application of individual methods and of how several method applications combine to a proof plan. The learner's activities in this scenario are restricted to browsing (in the proof plan and additional information such as the constraint state) in order to get more insights as well as replaying a proof plan step-by-step.

Interactive proof planning The learner constructs a proof plan for a given problem or has to complete a partial proof plan by selecting and applying methods and by determining objects for meta-variables (instances for meta-variables). We expect that here a student can learn how to apply methods and the heuristics for proving various theorems as well as to use tools to simplify the determination of mathematical objects. The learner's main activity is the selection of methods to be applied and the specification of the desired application of the method as well as the determination of mathematical objects (also, browsing the current proof plan and requesting additional information, e.g., the constraint state).

Island planning The learner has to construct a proof sketch for a given problem. We expect this scenario to support creative proof skills and consider this the most important mode for computer-supported learning of mathematical proof. Moreover, this scenario may support the understanding of the hierarchical assembly of proofs. While neglecting details, the learner can express a proof idea, detect intermediate goals, and suggest how the goals depend on each other. The main user interactions in this scenario are adding proof islands and links (i.e., dependencies) between islands (browsing the current island plan and requesting additional information occurs as well).

Free exploration The learner has full interactive access to the ΩMEGA system. She can upload the theory for the proof process, can define the prob-

lem, and initiate the proof process. Moreover, she can freely access all kinds of proof manipulations (application of methods, instantiation of meta-variables, speculation of islands). This scenario is only sensible for advanced learners. We expect that this scenario fosters inquiry learning.

Polya problem solving Polya suggested a framework for teaching meta-reasoning in mathematical problem solving. His famous book "How to Solve It" [12] has the form of a how-to manual. It is a formulation of a set of problem solving heuristics cast in form of brief questions and commands within a frame of four problem solving stages: (1) Understand the problem. (2) Devise a plan. (3) Carry out the plan. (4) Look back at the solution. Some questions and commands for (2) are, for instance: Do you know a related problem? Did you use all the data? Following Polya's ideas, each of the above scenarios can be enriched by a meta-cognitive framework. We expect this scenario to foster meta-cognitive abilities.

All these scenarios need a corresponding appearance in the proof GUI. For instance, the GUI-objects and functionalities of interactive proof planning and island planning differ. User-adaptivity to the learner and the context is our key design goal for the educational tool that is based on our proof planner.

4 The Interactive Proof Planning Scenario

The first scenarios realized are the replay and presentation of a proof plan and the interactive proof planning. Currently, both scenarios employ an extended version of *LΩUI* [14], ΩMEGA's GUI.

Exercises with ΩMEGA are represented in ACTIVEMATH's knowledge base by OMDocs [3]. This representation includes a textual and/or diagrammatic representation, a formalization of the theorem, and exercise-specific settings. When a student decides to perform an ΩMEGA-exercise during an ACTIVEMATH session, ACTIVEMATH launches ΩMEGA with its (extended) user interface. A bi-directional XML-RPC communication is established between ΩMEGA and a proxy in ACTIVEMATH. When the exercise is finished, some information about the ΩMEGA-session is sent to ACTIVEMATH's user model. For instance, the number of successful and of faulty method applications for relevant methods is returned as well as the number of remaining goals.

In the following, we will restrict our description to the interactive proof planning scenario and how it can be adapted to the context and the learner. Moreover, we discuss an interaction console, an extension of *LΩUI* for the interactive proof planning scenario.

4.1 Configurations for Interactive Proof Planning

First, we briefly explain the technical realization of the interactive proof planning scenario and its parameters, which are taken from the exercise representation or are dynamically determined when ΩMEGA is launched.

The interactive proof planning scenario combines the ΩMEGA proof planner and the agent-based command suggestion mechanism. It uses the agent mechanism for the adaptive generation of method and meta-variable suggestions, i.e., it employs sets of commands and agents that compute method and meta-variable suggestions. Therefore, a set of commands and agents is the first parameter of the scenario.

The scenario runs the independent agents concurrently. This allows for an any-time behavior of the interactive proof planning scenario, which immediately reports suggestions to the user who can then decide to apply one of the suggestions or to wait for other suggestions. Time consuming processes of unfinished suggestion computations are terminated, when the learner selects a suggestion. ΩMEGA tries to apply a selected suggestion, which is either successful and results in a new proof state and new suggestions for the learner or fails and results in some failure feedback for the learner.

The two other parameters of the interactive proof planning scenario are a strategy and the level of automation. Proof planning strategies implement different ideas for proving. That is, different strategies, which comprise different methods, tackle a proof planning problem in a different way. If a strategy is selected for the problem at hand, then there has to be at least one command and corresponding agents for each method of the strategy.

Some of the methods of a strategy should not be displayed to the learner for selection but should be applied automatically. The methods of a strategy that are applied automatically is also called the level of automation.

A complete instantiation for the interactive proof planning scenario are called a configuration. A configuration consists of

- a strategy,
- commands and agents,
- the level of automation.

4.2 Adaptivity of Configurations

Next, we discuss how the instantiation of a configuration for a concrete interactive proof planning exercise enables the adaption to the learner and the context. In particular, the distinction between methods, commands, and agents allows to control the level of freedom in the interaction with the proof planner. The setting can be more or less restricting, more or less guiding, and it can realize certain pedagogical strategies. For instance, suggestions that can lead to impasses can deliberately be introduced into the choice. The adaption is realized by the different parts of a configuration as described in the following.

4.2.1 Selection of a Strategy

The configuration can determine a preference for a proof planning strategy. There can be different proof planning strategies available for a proof problem,

i.e., different proof ideas for the problem. Consider, for instance, problems about properties of residue classes [5]. Residue classes are equivalence classes of integers and represent algebraic structures. The proof problems include algebraic properties of residue classes, such as associativity or the existence of inverse elements, and isomorphy of residue classes. There exist at least three strategies to tackle a residue class problem: the first strategy tries to solve the problem by applying known theorems, the second strategy reduces a residue class problem to a set of equations, which have to be solved afterwards, the third strategy introduces a case split over the (finitely many) elements of a residue class. The decision for a strategy depends on the knowledge of a learner (whether she knows the theorems that are the prerequisites of the first strategy, whether she knows the methods employed by a strategy) and her performance in previous exercises (e.g., when the other strategies have been trained already). Such configuration heuristics can be encoded by pedagogical rules. For instance (simplified for better comprehension)

```
IF studentKnowledge(prerequisites (firstStrategy)) > medium
AND studentKnowledge(firstStrategy) < medium
THEN present exercise-for(firstStrategy)
```

```
IF  studentKnowledge(firstStrategy)  > medium
AND studentKnowledge(secondStrategy) > medium
AND studentKnowledge(thirdStrategy)  < medium
THEN present exercise-for(thirdStrategy)
```

4.2.2 Selection of Commands and Agents

The agent-based command suggestion mechanism can be configured by adapting the sets of commands and agents depending on the learner, her user model, and the learning goal of an exercise. Configuration heuristics for commands and agents can be encoded by pedagogical rules as well.

In general, the commands and agents can encode different degrees of guidance and restriction. The concrete choice of commands and agents can depend on different aspects related to the exercise and the learner. For instance, agents can compute more or less complete suggestions of commands depending on the experience of a learner: (1) A novice learner would start with much support and (almost) fully-specified suggestions. (2) Later on, the support can fade and more input is requested from the learner in order to make her think more and overcome misconceptions in the application of a method. Moreover, agents can compute applicable or faulty suggestions of commands: (1) If the goal is to rapidly prove a conjecture (as in the pure proof assistant situation), then agents are chosen, which check for applicability and provide only fully-specified, applicable suggestions. (2) If the learner should understand why a particular method is applicable, what the particular method actually does, and for which purpose it is applied, then agents are chosen, which provide faulty suggestions for the application of this method. This way a student

may experience a typical error and receive help in order to avoid that error subsequently.

In the following, we discuss what kind of commands and agents for method suggestions can be computed automatically and what further kinds of commands and agents can be provided by the author of an exercise.

An initial set of commands with corresponding sets of agents can be automatically generated from the method specification. The automatically generated agents can vary wrt. whether they provide input for a particular argument or whether this input is demanded from the learner. A command for a method, which has as arguments a goal and premises, can have suggesting agents for the premises or leave the selection of premises to the learner.

For instance, exercises for group homomorphisms are designed to teach properties of abstract groups and homomorphisms between groups. Relevant properties are, e.g., the associativity of the group operation, the existence of a unit element, and the existence of inverse elements. For each property, there exists a method, which applies the property. For instance, the method Group-Is-Assoc applies associativity, i.e., it reduces a goal $F[((g_1 \circ g_2) \circ g_3)]$ to a goal $F[(g_1 \circ (g_2 \circ g_3))]$, if there are premises $group(G, \circ)$ (where G is a set and $\circ$ is a binary operation) and $g_i \in G$ for $i = 1, 2, 3$. Five agents are needed in order to compute suggestions for Group-Is-Assoc, one for the goal and four for the premises of the method. From the specification of the method, five corresponding agents can be automatically created. When all these agents are employed, then fully-specified and applicable suggestions for the method are created. If some of the agents are omitted, then only partially-specified suggestions are created, such that the student has to specify the missing input.

Other commands with corresponding sets of agents that realize particular suggestions, e.g., a suggestion that does not lead to a solution or one that represents a typical error for a particular learning goal, have to be added to an ActiveMath exercise by the author of the exercise. The additional commands and agents are evaluated and merged with the automatically generated ones.

As explained above, the automatically generated agents create only applicable suggestions. What other suggestions are may interesting in homomorphism exercises? Homomorphism exercises typically involve two group structures $(G, \circ)$ and $(G', \circ')$, where the one structure is the domain structure of a homomorphism h and the other one is the codomain structure of h. An author who wants to teach to distinguish between the codomain and the domain of h could specify agents, which mix elements of both structures, e.g., suggesting an application of Group-Is-Assoc to the goal $F[((g_1 \circ g_2) \circ g_3)]$ with the premises ($group(G', \circ')$ and $g_i \in G$ for $i = 1, 2, 3$. Moreover, an author who wants to teach the different group properties and when they are applicable could specify agents, which suggest applications of the methods that do not match with the goal of the method, e.g., suggesting an application of Group-Is-Assoc to the goal $F[g_1 \circ g_1^{-1}]$.

Author-specified commands and agents can also provide over-specified sug-

gestions. An over-specified suggestion comprises not only the specification of all input necessary to apply a method but also (parts of) the results of the application of the method (i.e., new assumptions and goals in the proof state resulting from the application of the method). For example, consider a method that uses a Computer Algebra System to simplify a term in a goal by performing arithmetical simplifications. The automatically generated command comprises only one argument, the goal to which the simplification should be applied. An author who wants to teach the learner to perform the simplifications can specify a command with an additional argument for the result of the simplification. Then, the learner has to provide input for this argument in suggestions for the application of the method and the result of the method application can afterwards be compared with the input of the learner.

4.2.3 Level of Automation

The automation of the application of certain methods avoids bothering the learner with the specification of proof steps, which she already knows. Methods that decompose logical quantifiers and connectives are typical examples for automated methods. Moreover, methods that perform some normalization or re-writing of assumptions and goals can be applied automatically, in case the learner is diagnosed to understand the outcome of these methods.

4.3 Interaction Console

Currently, the user interface for interactive proof planning exercises is *LΩUI* enriched by the interaction console as shown in Figure 1. The figure's screen shot shows the GUI in a stage, where methods have already been applied to the initial problem. The interaction of the learner with the interactive proof planning scenario is via the interaction console only, which is shown in detail in Figure 2. This relatively simple dialog window offers choices of the method and meta-variable rather than ΩMEGA's full functionality.

The interaction console presents the suggestions computed by the interactive proof planning scenario. The learner can choose a goal from the first column, the 'Goals' column, of the interaction console. The available method suggestions for this goal are then shown in the second column, the 'Actions' column. When the learner selects an action from the second column, she may be asked for additional parameters, e.g., which assumptions should be used as premises. The third column of the interaction console, the 'Variables' column displays suggestions for the instantiation of meta-variables in the proof plan. The last column, the 'Undo' column, allows to delete proof steps and remove instantiations of meta-variables. The button with the computer symbol at the top of the interaction window starts an automatic proof planning step, in case the learner got stuck altogether during the proof construction.

To make the most frequent user interactions easy to perform only few key strokes are necessary for those interactions. Following this philosophy means

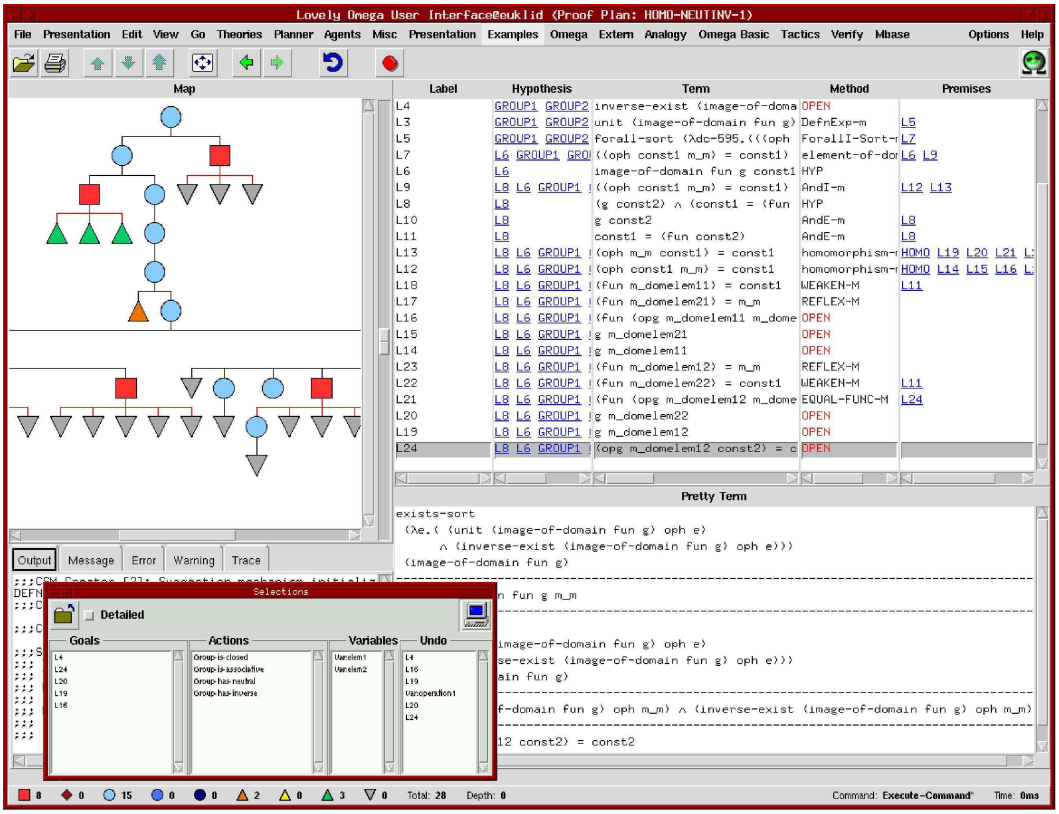

Fig. 1. The user interface with a graphical proof presentation, the sequence of proof lines, and the interaction console.

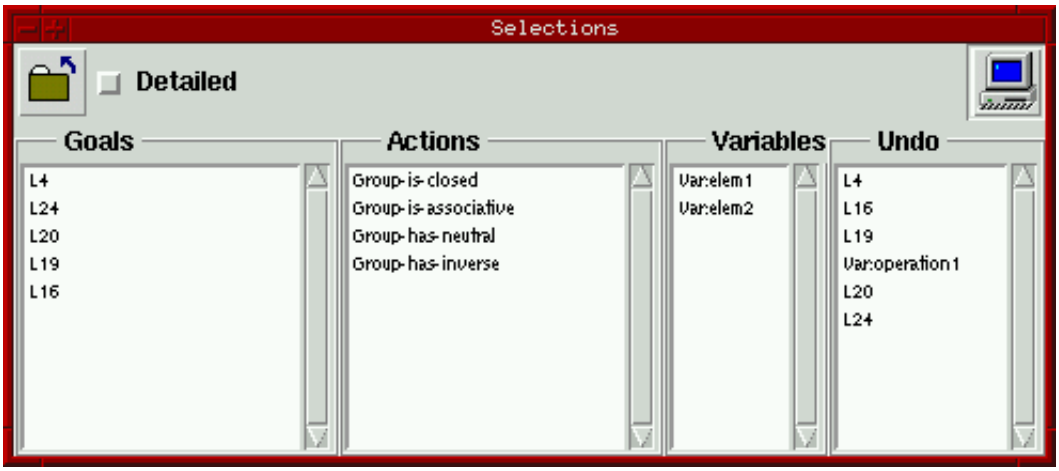

Fig. 2. The interaction console for interactive proof planning.

that selecting a method's name for input is faster than typing full method names. Moreover, to remember a name causes more load on the working memory than to recognize the name as an appropriate one. Therefore, again choosing a method from a list causes less distraction from the actual learning and understanding of proof. However, in certain learning situations and for certain learners it may be important to remember a method or to look it up

in the course material before using it in a proof. That is, choice menus are not always appropriate (similar to multiple choice questions which may not serve the learning purpose enough).

5 Conclusion

In this article, we discussed the application of the ΩMEGA system as a tool for learning mathematical proofs. We identified several design goals, that are required from a pedagogical perspective, for example, adaptivity to the needs and capabilities of the learner (which we identified as the key design goal for the educational usage of ΩMEGA), promotion of explorative learning, and allowance for faulty proof attempts, which are usually not met by most interactive theorem provers. Motivated by these requirements we designed four scenarios in which a learner can interact with ΩMEGA.

So far, we implemented two of these scenarios, presentation and replay of proof plans and interactive proof planning. The adaption of the interactive proof planning scenario is achieved by configurations, which can consist of different proof planning strategies, different suggestion agents, and different levels of automation. The suggestion agents realize different degrees of guidance and restriction and they can provide suggestions, which may lead to an error.

A configuration can be chosen according to the knowledge and mastery-level of the learner. Configurations lead to the following features that can be useful for learning:

- bearable complexity,
- dynamic suggestions for interaction,
- faulty suggestions can be used to trigger learning from failure,

Future Work

The described integration of ΩMEGA and ActiveMath does not yet comprise all the adaptions and functionalities needed for the implementation of tutorial requirements. For example, a configuration for the interactive proof planning scenario is currently chosen at the beginning and is then fixed during the whole exercise. This prevents dynamic adaption to the performance of the learner. Subject for adaption during an exercise are strategies (when the learner is not able to apply the initially selected strategy, then choose an alternative one), specificity of suggestions (when the learner fails to provide input for under-specified suggestions, then provide more specific suggestions), and availability of feedback.

The current GUI does not adapt the presentation of proofs, suggestions, and feedback to the needs of different learners. This will be the goal in the near future. Moreover, we are currently examining possibilities for the automated generation of faulty and over-specified suggestions.

References

[1] C. Benzmüller and V. Sorge. A Blackboard Architecture for Guiding Interactive Proofs. In *Artificial Intelligence: Methodology, Systems and Applications, Proc. of the of the 8th International Conference (AIMSA'98)*, volume 1480 of *LNAI*, pages 102–114. Springer-Verlag, 1998.

[2] A. Bundy. The use of explicit plans to guide inductive proofs. In *Proc. 9th International Conference on Automated Deduction (CADE-9)*, volume 310 of *LNCS*, pages 111–120. Springer-Verlag, 1988.

[3] M. Kohlhase. OMDoc: Towards an internet standard for the administration, distribution and teaching of mathematical knowledge. In *Proceedings AISC'2000*, 2000.

[4] H. Mandl, H. Gruber, and A. Renkl. *Enzyklopädie der Psychologie*, volume 4, chapter Lernen und Lehren mit dem Computer, pages 436–467. Hogrefe, 1997.

[5] A. Meier and V. Sorge. Exploring properties of residue classes. In *8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus-2000)*. AK Peters, New York, NY, USA, 2000.

[6] E. Melis. Why proof planning for maths education and how? In *Festschrift in Honor of Jörg Siekmann*, LNAI. Springer-Verlag, 2003.

[7] E. Melis, J. Buedenbender, E. Andres, A. Frischauf, G. Goguadse, P. Libbrecht, M. Pollet, and C. Ullrich. ActiveMath: A generic and adaptive web-based learning environment. *Artificial Intelligence and Education*, 12(4):385–407, 2001.

[8] E. Melis, C. Glasmacher, C. Ullrich, and P. Gerjets. Automated proof planning for instructional design. In *Annual Conference of the Cognitive Science Society*, pages 633–638, 2001.

[9] E. Melis and A. Meier. Proof planning with multiple strategies. In *First International Conference on Computational Logic*, volume 1861 of *LNAI*, pages 644–659. Springer-Verlag, 2000.

[10] E. Melis and J. Siekmann. Knowledge-based proof planning. *Artificial Intelligence*, 115(1):65–105, 1999.

[11] E. Melis, J. Zimmer, and T. Müller. Extensions of constraint solving for proof planning. In *European Conference on Artificial Intelligence*, pages 229–233, 2000.

[12] G. Polya. *How to Solve it.* Princeton University Press, Princeton, 1945.

[13] J. Siekmann, C. Benzmüller, V. Brezhnev, L. Cheikhrouhou, A. Fiedler, A. Franke, H. Horacek, M. Kohlhase, A. Meier, E. Melis, M. Moschner, I. Normann, M. Pollet, V. Sorge, C. Ullrich, C.P. Wirth, and J. Zimmer. Proof development with ΩMEGA. In *Proceedings of the 18th Conference on Automated Deduction (CADE-18)*, volume 2392 of *LNAI*, pages 144–149. Springer-Verlag, 2002.

[14] J. Siekmann, S. Hess, C. Benzmüller, L. Cheikhrouhou, A. Fiedler, H. Horacek, M. Kohlhase, K. Konrad, A. Meier, E. Melis, M. Pollet, and V. Sorge. $\mathcal{LOUI}$: $\mathcal{L}$ovely Ωmega $\mathcal{U}$ser $\mathcal{I}$nterface. *Formal Aspects of Computing*, 11(3):326–342, 1999.

[15] K. Reiss, F. Hellmich, and J. Thomas. Individuelle und schulische Bedingungsfaktoren für Argumentationen und Beweise im Mathematikunterricht. In *Bildungsqualität von Schule: Schulische und außerschulische Bedingungen mathematischer, naturwissenschaftlicher und überfachlicher Kompetenzen. Beiheft der Zeitschrift für Pädagogik*, pages 51–64. Beltz, 2002.

[16] K. VanLehn, S. Siler, C. Murray, T. Yamauchi, and W.B. Baggett. Human tutoring: Why do only some events cause learning? *Cognition and Instruction*, 2001.