Using Computer Algebra Systems as Cognitive Tools

The ActiveMath group: Erica Melis, Jochen Büdenbender, Adrian Frischauf, Georgi Goguadze, Paul Libbrecht, Carsten Ullrich
dev@activemath.org

DFKI Saarbrücken, D-66123 Saarbrücken, Germany

Abstract. We describe how Computer Algebra Systems (CASs) can be used as cognitive tools in a learning environment. In particular, we show how a CAS is employed in ActiveMath, how different types of CAS-exercises are designed, and how feedback can be produced with the help of the very same CAS. We report the results of a first preliminary formative evaluation of ActiveMath' CAS-exercises in a university course and some modifications of ActiveMath caused by this evaluation.

1 Introduction

The ActiveMath learning environment [9] presents a variety of (interactive) learning materials to the student rather than exercises or examples only. The course materials include motivations, concepts, elaborations, exploratory animations, worked-out examples, and exercises with feedback.

As its name suggests, ActiveMath emphasizes the active role of the student and leaves space for exploratory learning. This feature is greatly supported by the integration of cognitive tools. Currently, ActiveMath integrates the Computer Algebra Systems (CASs) Maple and MuPAD and a proof planner as cognitive tools. They provide the backbone for interactive problem solving that does not need pre-defining few possible problem solutions, and for dynamically producing local (problem solving) feedback to the user's actions. Furthermore, the user's exercising performance is used to update ActiveMath's long term user model.

The term *cognitive tool* was coined in [7] and generally denotes instruments supporting cognitive processes by extending the limits of the human cognitive capacities, e.g., the working memory. When applied to learning, such tools can help, e.g., to remember, to practice, to hypothesize, to solve a problem. In particular, when learning is difficult because it is too complex or because several things have to be done at the same

time, these tools can help considerably. This is well-known for simulation tools [6], dynamic geometry systems, etc.

In this paper, we mainly describe how specific interactive exercises with CASs can be designed and report on their actual usage. To begin with, let's summarize *why* the Computer Algebra Systems can support active learning of (mathematical) problem solving. They help the learner a) to *explore* a problem interactively and *directly experience* the result of a calculation, even a complex one; b) to *focus* on a particular subtask or skill in solving a problem rather than paying all the attention to a detailed computation; c) to correct misconceptions and errors and support her by local *feedback*; d) to learn how to handle the cognitive tool for after-learning usage.

In the remainder of the article, we discuss some advantages and difficulties of integrating CASs into ITSs. Section 3 concretely looks at CAS-exercises in ActiveMath. Section 4 contributes a preliminary formative evaluation and reports our experiences with ActiveMath' CAS-exercises in real-life mathematics lessons attended by first-year university students.

2 Computer Algebra Systems

There exist quite a number of CASs that are used for professional and for educational purposes, e.g., Maple [1] or MuPAD [10]. These systems are stand-alone and have elaborate algorithms to perform even complicated computations. As stand-alone tools they have been used in some schools but as far as we know there is no learning environment integrating a CAS. Instead, existing ITS use their specific problem solving modules such as the physics problem solver in Andes [2].

However, the CAS facilities differ largely from those specifically designed problem solvers: on the one hand, a CAS computes only one solution path and does not compute the search space for a solution as, e.g., the physics problem solver does in Andes. A CAS does neither directly provide information evaluating the user's input apart from correct/incorrect and impossible inputs. On the other hand, a CAS is a very powerful stand-alone system that can solve complex and complicated problems. This makes it a cognitive tool that can help to solve even real-world problems in a school lesson's time.

For developers of ITSs and authors the use of an existing third-party CAS has the disadvantage that the CAS is a black-box system as opposed to a white-box system the developers would have implemented or can at least modify themselves. This makes enhancing the CAS with more

functionalities rather difficult. In comparison, when the developers of a learning environment have also implemented the service system, they can easily adapt it to suit the pedagogical and technical needs of the ITS (which would also be possible if the system was available as open source).

In what follows we investigate the questions (1) what can be done with such black-boxes, (2) what is the advantage for an ITS that integrates a CAS, and vice versa (3) what is the advantage of not just taking the stand-alone CAS but rather integrating it into the larger learning environment? As for the third question, when a CAS is integrated into a context which presents conceptual content and examples, it is possible for the learner to see the big picture of the domain and to refer to or being referred to knowledge that is relevant for solving an exercise such as definitions, similar examples, or missing prerequisite knowledge. Moreover, such an environment offers user-adaptivity, a feature that stand-alone CASs are not designed to provide. For instance, ActiveMath can restrict the available features of a CAS: a student who is learning mathematical integration should not use a CAS to solve his exercises completely, but using the CAS as a calculator for auxiliary calculation is acceptable.

3 CAS-Exercises in ActiveMath

The philosophy of ActiveMath suggests that the user controls her exercise activities herself and no single solution needs to be followed as long as a correct solution results at some point. Moreover, since handling the CAS can be one of the learning goals, the user's input is CAS commands.

Figure 1 shows a screen shot of a CAS-exercise session. With the start of an exercise, a console appears. It displays the text of the exercise again such that problem and later the solution can be seen in *one* window. In Figure 1 the text asks to prove an inequality. The learner performed three calculation steps which all were correct. However, her first attempt to finish the exercise was not successful, because the result can further be simplified to $0 \leq (a+b)^2$ which is *obviously* correct because of the square on the right side. After two more steps the learner successfully finished this exercise and received a positive feedback.

The student can save the current dialog as a text. This allows to quickly resume a session and to inspect a solution later. Furthermore, the student can annotate CAS-exercises (as well as any other learning material) with public or private notes. ActiveMath also offers a scratch pad facility, for auxiliary calculations. A CAS can be used to produce visualizations. ActiveMath uses these to more adequately present mathe-

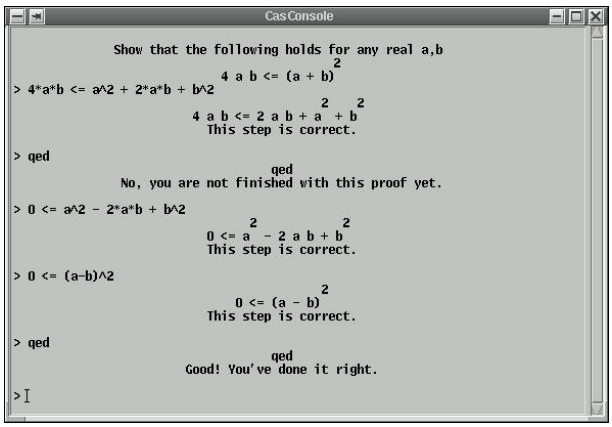

Fig. 1. An example CAS-session

matical concepts which have a dynamical aspect, e.g., the convergence of a sequence or the definition of the velocity of a body.

3.1 Integration Mechanism

We have developed a framework that allows an easy integration of existing mathematical systems within ActiveMath. In this framework, the CAS is started on the server via a console applet on the client in the browser. A proxy handles the communication between the CAS (where the student's answers are evaluated), the console (to provide input/output), and ActiveMath (e.g., to update the user model).

3.2 Exercise Types

ActiveMath offers a variety of exercises in general. This includes multiple choice questions, proof planning exercises, CAS-exercises, and control questions. We experimented with two different types of CAS-exercises, single-step exercises, where only one input is required, and multi-step exercises, where each single step is evaluated and feedback is given. In addition to the usual check of correctness of a step, multi-step exercise results are evaluated with respect to the finality of the input (see Figure 1).

3.3 Local Feedback in CAS-Exercises

In ActiveMath we distinguish between global and local feedback [8]. Global feedback may be given after reading, navigating, and, of course, exercising. In the case of exercising with a CAS, the result of the exercise session updates ActiveMath's user model, e.g., if the student fails to solve an exercise for a certain concept, its mastery value is decreased. Subsequently, this can trigger suggestions for learning certain content, solving certain exercises, etc. As opposed to global feedback, local feedback is given during exercising and refers to the activities in the learner's problem solving attempts. This local feedback is what is commonly provided by most ITS [5].

In this section we analyze how the CAS integrated into ActiveMath can be employed for producing local feedback for CAS-exercises. The philosophy of ActiveMath suggests the following: to allow for more than a single solution, to relieve the author from the burden of foreseeing and implementing every possible (in)correct solution step, and to exploit the capabilites of the CAS. The idea is to use the CAS to compute whether an input satisfies a *test condition* and to deliver feedback depending on the answer. We distinguish between five classes of test conditions:

i test the equality of the input with respect to a predefined value. For example, if the user has to find the Euler indicator of a given number, his input will also be a number which is checked for equality against a solution either provided by the author or calculated by the CAS.

ii check the validity of an expression in which the input is a subexpression. For example, if the correct input is supposed to be the inverse of a given permutation, then the product of these two should be equal to the identity permutation. Any mathematically incorrect input does not satisfy the test condition and thus, results in a feedback stating that this product is not equal to the identity and hence, the input is incorrect. Such a feedback is meant to stimulate further learning by wondering how the feedback is related to the original task.

iii test the equivalence of the input with respect to a given expression. This leaves quite some freedom for the user's input because it is judged only modulo equivalence rather than equality. Here, an important case is the check for equivalence in a sequence of transformations/manipulations of formulas (e.g., equations) as shown in Figure 1. Only the correctness but not the progress towards a goal is judged.

iv test the equivalence of the solution set corresponding to the input and the solution set corresponding to a given formula. This method can

be used if the exercise deals with (in)equations. Theoretically, this class is equal to class (iii). Practically however, this method helps to overcome the cases when the CAS is not able to check for equivalence directly.

v test whether the input is sufficiently simplified. For example the input `limit(x+2, x=2)` will not be accepted as the final solution of an exercise, even if it is correct. This term should be further simplified to 4, which will be accepted.

In the following we elaborate on how these classes of test conditions are used to provide several kinds of feedback. A common classification of feedback distinguishes between Knowledge of Result, Knowledge of Correct Result, Answer Until Correct, and Elaborated Feedback. The CAS-exercises in ActiveMath cover them by the five types of local feedback explained in the following:

Help. Since the student inputs her calculation steps via the CAS input syntax, a Help function explains the appropriate CAS commands for this exercise. Help can be accessed by typing "help". This function can also be used to author hints for the student.

Input-Error. A common type of errors is incorrectly written input. These input errors may have different causes: (1) incorrectly typed commands or expressions (given the CAS language) which cannot be interpreted by the CAS or (2) a misconception of the intended input. The first, e.g., unbalanced brackets, could be avoided by a schematic input editor provided with the planned *intermediator* described in the section 'near future'. Errors caused by misconceptions occur, when the user's input is syntactically correct but does not represent the right kind of mathematical object. In this case, the CAS' calculation of the test condition fails. If the CAS can detect the failure, appropriate feedback can be provided.

Correct/Incorrect. If an exercise requires direct input of the solution, the CAS can check it and inform the learner immediately about the correctness of her input. If the answer is correct, the exercise session ends. Otherwise, the learner is invited to provide a new input. In exercises that require several calculation steps before the final solution, every step is evaluated and feedback about its correctness is given. The check for correctness depends on the mathematical objects which occur in the solution steps. We use the classes described in (i) - (v). An example for (iii) is the exercise "Please enter the power set of $\{1, 7\}$" where the learner's input is compared to the solution $\{\emptyset, \{1\}, \{7\}, \{1, 7\}\}$

Elaboration-on-Error. In some exercises the feedback can be more elaborate than simply 'correct' or 'incorrect'. An exercise which needs a

comparison of solution sets (iv) is the following: "Solve the inequation $1 \leq x \leq 4$, where x is a real number". The correctness of the student's input can be checked by comparing it with the correct solution which is $x \in [-2,-1] \cup [1,2]$, e.g., with the following CAS code: `if bool(solve(new,x) = solve(solution,x)) then result:=1 end_if` where `new` contains the learner's input and `solution` is the correct solution. One option for an elaborate feedback is to point the learner to elements belonging to her solution set but not to the set of the correct solution and vice versa.

Show-Solution. Finally, an authored solution or one automatically computed by the CAS is yet another kind of feedback in CAS-exercises in ActiveMath. If the author provides a pre-determined solution, the learner can access it by typing "solution" and then the exercise is considered finished immediately. This feature can be suppressed for pedagogical reasons.

4 Preliminary Empirical Evaluation

4.1 Experimental Design

The target population for ActiveMath is university-level students with different skills and mastery-levels in mathematics. Hence, to evaluate if and how the current version of ActiveMath influences the student's active problem solving and learning (including usability considerations), we were planning to run a real-life study with 100 first-year students in obligatory weekly sessions augmenting the lectures of an introductory calculus course at the university of Saarland. Disappointingly, due to the unforeseen last-minute choice of the lecturer and administrative difficulties, we were unable to carry out the study in a better coordination with the lecturer's material and homeworks and in *obligatory* sessions, i.e., with as many as 100 students and with the opportunity to test groups of students. That's real life... Still, at the moment we are more interested in an evaluation with a real-life setting than in a lab experiment with ideal but somewhat unrealistic conditions.

We had to resort to varying numbers of voluntary attendants with an average of six students for one semester and could not include a proper pre-test but had to rely on the student's last school grades in mathematics. Thus, we decided to observe the attending students, to record their questions and the human tutor's answers to gain an initial understanding of how and if CAS-exercises are usable and contribute to active learning. Moreover, we had to make use of the content of a full calculus course that

is based on the rather traditional textbook [3] that was already encoded in ActiveMath and to a number of additional interactive exercises.

As a result, we conducted a preliminary formative evaluation with the objective to successively improve ActiveMath functionalities and design. The evaluation was conducted over one semester with first-year students of computer science. They attended the obligatory course *Mathematics for computer scientists* that covered calculus up to integration in one dimension. The course comprised four hours lecturing per week, homeworks, and weekly traditional obligatory exercises as well as our additional voluntary ActiveMath session once a week.

At the beginning, 29 out of 150 students registered for the voluntary ActiveMath session which was our sole empirical source. Because of the optional nature of the ActiveMath sessions the number of attendees dropped considerably during the semester. In interviews, the students explained that they were under time pressure and avoided any activity not directly contributing to the homeworks or to the announced examination.

The evaluation sessions took place in computer rooms, where every student had her own computer and each session was attended and supervised by a tutor who first briefly introduced the handling of ActiveMath and later gave hints and answered their questions (all recorded).

Apart from the observations and the recording of questions and answers we used the methods questionnaires and interviews. Before the actual ActiveMath sessions 29 students filled an on-line questionnaire and at the end a second questionnaire was filled by six students only and more interviews were conducted to compare with the student's initial attitudes and motivation.

4.2 Results

The first questionnaire showed a rather low average mathematics grade. The last mathematics school grades varied from 1 (very good) to 4 (sufficient), with an average of 2.75 (satisfying). All students frequently used a computer prior to the evaluation. Former use of learning software was restricted to vocabulary trainers in a few cases. The second questionnaire and interviews invariably showed that the student's motivation and attitudes towards mathematics and active learning increased.

Qualitative Observations Our observations confirmed a strong increase of learning motivation. Students enjoyed using CAS-exercises, especially to explore new content. The feedback even led to little competitions

among the students in which each student wanted to solve the exercises first.

These "competitions" can explain another unexpected observation: the more elaborate feedback was considered less important. The students were more interested in whether their solutions were correct or incorrect. We assume that the need for elaborate feedback will increase when the exercises become more difficulty and when no human feedback is available.

In fact, we noticed that the need for help by the human tutor gradually decreased. Initially, the students had problems using ActiveMath, in particular, problems with the CAS-syntax. This was remedied by the implementation of the help facility.

Similarly, the navigation behavior of students changed during the semester. In the first sessions, the students traversed the curriculum in a book-like manner (going forward or backward page-wise). After they became more familiar with the system, they navigated more freely and took advantage of the overall content presentation of ActiveMath. Then, they also used the dictionary, e.g., to search for additional exercises. Not surprisingly, students with good mastery felt more secure about the content and, hence, used the advanced features, e.g., inspected the user model to find out about their weaknesses, whereas weaker students were more focused on solving exercises and did not make so much use of additional features ActiveMath offers for self-guided learning.

Technical Problems and Their Solution We identified a number of technical problems in the realistic course setting, in particular, when many students attended a session.

Performance. The system's performance dramatically decreased when a large number of students simultaneously used ActiveMath. Hitherto ActiveMath was not developed with regard to efficiency, so that several standard optimization solutions, e.g., caching of content, were not yet implemented. As a first alleviation, we run the knowledge base now within the same process rather than as a resource-intensive independent process.

Presentation. The presentation of mathematical formulas in the console as well as in the ActiveMath html pages is not at all perfect yet. This is due to the browser-based presentation. Therefore, we are investigating a SVG or MathML presentation for the browser. The current CAS-console is implemented as a rather basic java applet that requires no additional software installation but can offer a limited user interface only. The next version of the CAS-console will require a newer java version with enhanced graphical capabilities. Several small presentation problems, e.g., an un-

necessary amount of brackets in the formulas, were solved quickly. ActiveMath' learning materials are not stored as predefined HTML pages, but encoded in an knowledge representation that separates content from presentation. This allows to adapt the presentation rather straightforward because only a single presentation rule has to be changed.

5 Conclusion and Near Future

We have integrated Computer Algebra Systems into the learning environment ActiveMath and employed their computational power to support active and exploratory learning as well as evaluation and feedback for the student's problem solving actions.

Three of the most interesting technical novelties for ITSs in general – apart from the technical integration of CASs – might be the distinction between five classes of test conditions for user input in CAS, the dynamic generation of several types of local feedback in CAS-exercises with the help of the CAS, and the saving of the student's solution for later usage.

We conducted a preliminary formative study to evaluate the content and design of CAS-exercises and their surroundings in ActiveMath. The results were promising in terms of exploratory learning and increased motivation and also gave rise to some technical changes that made the interface more effective for students.

To address those questions that we were unable to investigate in the described preliminary evaluation, we are planning to conduct a more formal study with more students. In one of the next studies we not only plan to test the actual post-performance but also plan to investigate the question how the difficulty and complexity of exercises influences the student's use and benefit from ActiveMath.

As a first step towards an authoring tool, we will prepare templates for each class of test conditions to facilitate the creation of CAS exercises.

For the technical development, very soon the integration of CASs into ActiveMath will be augmented by an *intermediator*, a module that provides several useful functionalities among others abstract input syntax, abstract authoring language, user-adaptive help and feedback, and more sophisticated detection of syntax errors.

An extended version of this paper is available as a technical report [4].

References

1. B.W. Char, G.J. Fee, K.O. Geddes, G.H. Gonnet, and M.B. Monagan. A tutorial introduction to MAPLE. *Journal of Symbolic Computation*, 2(2):179–200, 1986.

2. C. Conati, A.S. Gertner, K. VanLehn, and M. Druzdzel. On-line student modeling for coached problem solving using baysian networks. In A. Jameson, C. Paris, and C. Tasso, editors, *User Modeling: Proc. of UM97*, pages 231–242, 1997.
3. B.I. Dahn and H. Wolters. *Analysis Individuell*. Springer-Verlag, 2000.
4. The ActiveMath group. Using computer algebra systems as cognitive tools. Technical Report RR-02-01, DFKI Saarbrücken, Saarbrücken, 2002.
5. B. Jacobs. Aufgaben stellen und Feedback geben. Technical report, Medienzentrum der Philosophischen Fakultät der Universität des Saarlandes, 2001.
6. W.R. Joolingen and T. Jong. Design and implementation of simulation-based discovery environments: the SMISLE solution. *Journal of Artificial Intelligence and Education*, 7:253–277, 1996.
7. S. Lajoie and S. Derry, editors. *Computers as Cognitive Tools*. Erlbaum, Hillsdale, NJ, 1993.
8. E. Melis and E. Andres. Evaluators and suggestion mechanisms for activemath. Technical report, DFKI, 2002.
9. E. Melis, E. Andres, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. Activemath: System description. In J. D. Moore, C. Redfield, and W. L. Johnson, editors, *Artificial Intelligence in Education*, pages 580–582, 2001. IOS Press.
10. Andreas Sorgatz and Ralf Hillebrand. MuPAD. *Linux Magazin*, (12/95), 1995.