CreaComp: Computer-Supported Experiments and Automated Proving in Learning and Teaching Mathematics[1]

Günther Mayrhofer[1] and Susanne Saminger[2] and Wolfgang Windsteiger[3]

[1] Institut für Algebra, guenther.mayrhofer@students.jku.at
[2] Institut für Wissensbasierte Mathematische Systeme, susanne.saminger@jku.at
[3] Institut für Symbolisches Rechnen, wolfgang.windsteiger@risc.uni-linz.ac.at
JKU Linz, A-4040 Linz, Austria

Abstract

We present an environment for learning and teaching mathematics that aims at inspiring the creative potential of students by enabling the learners to perform various kinds of interactive experiments during their learning process. Computer interactions are both of visual and purely formal mathematical nature, where the computer-algebra system *Mathematica* powers the visualization of mathematical concepts and the tools provided by the theorem proving system *Theorema* are used for the formal counterparts.

1. Introduction

Both in classroom and in scenarios of distance learning of mathematics the use of computer-algebra systems has become more and more popular. Recently computer-support focuses on *(symbolic) computations*, e.g. calculating limits, derivatives, integrals, and *visualization*, e.g. plotting real-valued sequences or functions: the availability of powerful algorithms allows to perform calculations even in situations when the method would require a tremendous computational effort when executed "by hand" or when the algorithms are not known to the students at a certain level of education. Since modern symbolic computation systems are powerful enough to carry out all computations done in high-school and undergraduate mathematics, the question of which methods should be taught to students and for which tasks we rely on the help of the computer is of utmost importance. There is no absolute answer to this and the "White-Box Black-Box principle", see [2], usually serves as the didactic guideline, by which, depending on the didactical goals certain methods are taught in detail in one phase of education (the white-box phase) whereas they can be applied as black-boxes, e.g. by using a computer, in later phases.

On the other hand, most of the available electronic learning material for mathematics only rarely focus on computer-support for acquiring such crucial mathematical skills as "exact formulation of mathematical properties" and "rigorously proving mathematical properties correct", for exceptions see e.g. [1, 7].

In this paper we present the CreaComp project, whose main goal is to develop electronic course material for self-study and also for use in classroom. In its current state, CreaComp comprises approximately 15 (only loosely connected) learning units on an undergraduate level, such as e.g. equivalence relations, polynomial interpolation, or Markov processes. The computer-aid provided in CreaComp units follows the didactical guidelines set up in the MeetMath project, see [6], and it promotes an approach of self-paced learning that has been centered around *gaining insight into mathematical concepts through computational and graphical interaction*. In addition to computational and graphical interaction the CreaComp approach now adds *formal reasoning* as a third important component by integrating the *Theorema* system, see [3, 4, 5]. *Theorema* is designed to become a uniform mathematical computer environment, but in the CreaComp context, among its many features it mainly provides the mathematical language and the possibility of fully automated generation of mathematical proofs.

[1]This work is done within the project "CreaComp: E-Schulung von Kreativitaet und Problemloesekompetenz" at the Johannes Kepler University of Linz sponsored by the Upper Austrian government.

The CREACOMP approach aims to combine the experimental approach of discovering mathematics through interactive visualizations and computations with the rigorous approach of proving every claim that is made. By using the *Theorema* system, an experimental flavor can also be given to the formal part, i.e. students can observe with little effort how the *available knowledge influences success or non-success of a proof*, how *tacit assumptions* are often used in human arguments, or how *mathematical theories evolve* from sometimes simple definitions.

We present a case study on the concept of equivalence relations and set partitions, in which we demonstrate the entire bandwidth of computer-support that we envision for modern learning environments for mathematics ranging from "getting first ideas and intuitions" over "checking the validity of first ideas on a wide variety of examples" until "rigorously proving one's own conjectures".

2. Computer-Support in CreaComp Learning Units

The most dangerous scenario of computer-supported mathematics is that the extensive use of computation and visualization may lead to a tradition of "proof by inspecting particular examples". The combination of MEETMATH and *Theorema* in the frame of the CREACOMP-project, therefore, aims at providing computer aid for visualization, computation, but most importantly also for proving. MEETMATH provides a didactical concept, in which the learning phase is structured into phases of *motivation*, *acquisition*, *strengthening*, and *assessment*. During acquisition phases we use computation and visualization in order to fertilize the *intuition* about mathematical objects. The newly acquired knowledge is then formulated rigorously in the *Theorema* language, and the subsequent proofs serve for *strengthening*. At the same time, the use of an automated theorem prover and its possibility to generate human-readable proofs within a few seconds can enhance the *understanding* of mathematical argumentation, and therefore serves the *acquisition*, this time, however, on the learning level of mathematical problem solving techniques. Moreover, we think that by having a machine to give formal proofs of all statements and therefore being forced to fill all gaps in the proofs, the students can better understand the evolution of mathematical theories.

The interplay of MEETMATH and *Theorema* is facilitated by the common underlying *Mathematica* technology, since both make use of the capabilities of the *Mathematica* notebook-frontend. CREACOMP educational units are distributed in the form of *Mathematica* notebooks containing structured text, inlined or displayed mathematical formulae, graphics, and all sorts of *Mathematica*/*Theorema* input and output. Formal entities such as definitions or theorems, which we later want to use in computations or proofs, are written as *Theorema* input cells. Input cells differ in layout from the surrounding text blocks so that they can easily be recognized as active content. These cells must be evaluated in order for their content to be accessible in the *Mathematica* kernel later. Note, however, that *Theorema* input is quite different from usual *Mathematica* input! *Theorema* understands most of the common mathematical notation, notably all sorts of quantifiers written in appealing two-dimensional form and, thus, *Theorema* input hardly differs from inlined mathematical formulae in the text. Although *Mathematica* and *Theorema* provide palettes and keyboard shortcuts for inputting two-dimensional expressions, *Theorema* input is always prepared in advance and the users are usually not required to type *Theorema* formulae.

Furthermore we provide visualization commands for certain mathematical objects. In addition to static plots we try to involve the students in actively exploring mathematical properties by providing *interactive visualizations* based on *Mathematica*'s GUIKit, a toolbox supporting the implementation of Java applications fed with *Mathematica* data. The availability of an interactive experiment is always indicated by *interaction buttons* in the running text. Buttons appear as gray boxed text and clicking a button calls its associated button-function, in which we start the interactive experiment, usually a Java application implemented in the *Mathematica* programming language and linked to *Mathematica* through the GUIKit. The application contains certain interactive elements such as text fields, buttons, checkboxes, or sliders, and some graphics that refreshes on any user interaction given through these active elements. The surrounding text and the text on the button roughly explains the associated experiment. In addition, all experimental tools are equipped with an online help that explains the possible interactions and, as a side-effect, it is meant to lead the students' experiments into a "reasonable direction" so that they might conjecture "relevant knowledge". Students have the freedom in exploiting the interactive tools in arbitrary manner, but it is important to provide them also some guidelines in what to try and what to observe, otherwise there is the danger that they get lost if they deviate from the intended track through the unit.

Finally, we encourage students to also prove their conjectures after their experiments. In this stage they may use the *Theorema* provers both in interactive or in fully automated mode. When it comes to automatically proving a conjecture expressed in *Theorema* language we provide a CREACOMP *prove panel* in order to hide the concrete call of the respective *Theorema* prover from the user. The prove panel serves as the uniform interface to *Theorema*'s automated provers, and it is made up of several buttons on a gray area spanning the entire width of the notebook. The prove-button on the left has a call to a *Theorema* prove method with appropriate parameters associated to its "button-pressed"-event. Every *Theorema* prover needs the knowledge base to be used in the proof as a parameter, thus, before actually starting the proof the user must compose the knowledge base in an interactive dialog. This so-called knowledge base composer displays the formula to be proven and it lists all definitions, propositions, theorems, etc. available at this stage. The user simply selects by mouse-click and sends the knowledge base to the prover by pressing the "Prove"-button in the dialog's bottom-right corner. Pressing the "Hint"-button in the bottom-left corner selects just the "appropriate" portion of knowledge for a successful proof. The appropriate knowledge for a certain proof cannot be detected automatically, the developer of the unit needs to hide this information within the prove panel so that the knowledge base composer can access it from there. Furthermore, the prove panel contains a button that allows to view a pre-generated successful proof. Both pre-generated and live-generated proofs appear in a separate window, and they come in human-readable format and explain each proof step in natural language. Moreover, *Theorema* proofs have some remarkable features: All formula references are active button elements, which will display the referenced formula in a small separate window, proof goals and assumptions can easily be distinguished by color, and the structure of the proof tree is reflected by the nested cell structure of the proof notebook, so that entire proof branches can be collapsed by a single mouse-click.

In all facets of the CREACOMP user interface we make heavy use of interactive notebook elements like buttons and hyperlinks in order to prevent students from struggling with unfamiliar input syntax. Due to the use of Java and *Mathematica*'s GUIKit, user experiments are mainly based on common modern user interface actions such as selecting items by clicking radio-buttons or checkboxes, opening dialogs by button-click, etc. instead of the usual *Mathematica* command interface.

3. Case Study: Equivalence Relations and Set Partitions

We try to illustrate the flavor of computer-support given in a CREACOMP unit in the example chapter on "equivalence relations and set partitions", in which we span from the definition of an *equivalence relation* in set theory and the definition of the *relation induced by a set of sets* to the main theorem that the relation induced by the factor set of an equivalence relation R is equal to R.

The unit starts by some motivation why we want to study equivalence relations, including hyperlinks to related units, and a summary of *Mathematica*/*Theorema* commands needed in this section. Firstly, the notion of a binary relation R on some universe A is defined in the *Theorema* language, immediately followed by the definitions of the main concepts leading to equivalence relations, namely *reflexivity*, *symmetry*, and *transitivity*. An interaction button opens the first interaction possibility, where the user can choose arbitrary relations to be visualized by their adjacency matrix in form of a chessboard-like raster array. The relations can be user-defined, or random reflexive/symmetric/transitive relations can be generated. A graphical interpretation of some properties of relations can be seen in these experiments.

Next the concept of the *class* of x w.r.t. a relation R on a universe A is introduced. Using *Theorema*, this writes as[2]

Definition[“class”,any[x,A,R], class[x, A, R]:=$\{a \in A | \langle a, x\rangle \in R\}$].
Although its appearance comes close to how a definition would appear in a textbook, this *Theorema* definition is *active input* and after evaluating the input cell, the definition can be referred to in this session as Definition[“class”]. Then the user is confronted with a visualization tool for classes. The experiment in this tool is similar to the above mentioned, only in addition to the adjacency matrix it shows an additional graphics with a small disk for each element of the universe arranged in a circle with a unique color assigned to each of them. An elements color is shown in the color of its label, if an element y belongs to the class of x, then y’s disk is colored in x’s color, if y belongs to more than one class, then its disk is gray. Via checkboxes the user can mark the elements, whose classes are to be visualized.

The notions reflexivity, symmetry, and transitivity are then explored in their relationship to classes. Relations having the respective properties are first visualized using the interactive tool just described, then conjectures are formulated in the *Theorema* language, e.g. that for a reflexive relation the union of all classes is equal to the universe, or that for a symmetric transitive relation the classes of two elements x and y coincide as soon as x belongs to the class of y.

Of course, not all conjectures are always true. Some are false in general, but they may be valid in special cases. For example, after inspecting relations in the visualization tool some students might believe, for instance, that every element is member in its own class After trying the proof they would realize that the prover fails to prove this proposition. In general, failure of a *Theorema* proof can have various reasons: the provided knowledge is insufficient or the proposition as stated is not provable, maybe not even true! This is just what we want to emphasize in teaching mathematics, and conventional learning material does not provide room for this experience. Finally, failure can also be due to an inappropriate proving method or inappropriate parameters for the method, but we eliminate these sources by packing the call to the prover into the prove panel.

Theorema’s possibility to inspect failing proof attempts comes very handy at this point, so students can investigate, which part of the proof led to failure. In the example above, they detect that the prover closes all branches successfully only $\langle x, x\rangle \in R$ needs to be proven for arbitrary x and R. Thorough inspection of the available knowledge at that stage shows that only $x \in A$ is known and the student might learn that the propsition as stated *cannot be proven* unless more is known about R. Thus, they have to add more side-conditions to the proposition. In *Theorema* this can be done using the with[...] expression. At this place, the unit now contains a proposition with a placeholder, where the students can insert the additional side-condition. Now a user can try different conditions in order to succeed in proving the proposition. Of course, in order to succeed with the above proof, R must have the property that $\langle x, x\rangle \in R$ is true for all $x \in A$, i.e. R must be reflexive on A. Hence, we must put this side-condition and can then successfully prove the adapted proposition. This proof is not particularly difficult, still the formal rigor in which it is carried out is instructive for students.

The remaining content of this unit explains some properties of sets of sets. In particular, the students can learn about the factor set of a relation, i.e. the set of all classes, and which properties a set of sets make it a “partition”. Moreover, the concept of an “induced relation” of a set of sets is introduced, namely the set of all pairs, for which there is a set containing its both components. In

[2]The definition in *Theorema* format can use arbitrary mathematical syntax and special symbols. In the concrete case, we use currying and write R and A as subscripts to “class” hence “class of ... w.r.t. R in A” having only one parameter x.

each step the procedure is similar: first *introduce* new objects or properties, then *visualize* in order to gain insights, *guess and propose* conjectures, *formalize* the conjectures precisely, and *prove* them automatically with *Theorema*. The key observations are then:

- if R is an equivalence relation on A then the factor set of R is a partition of A and its induced relation equals R.
- if P is a partition of A then the induced relation of P is an equivalence relation and its factor set equals P.

All these theorems including also all auxiliary lemmata necessary for compact proofs of the main statements are proved fully automatically within this learning unit.

4. Conclusion and Future Work

CREACOMP is work in progress. Therefore we do not have results on evaluation of the units in classroom yet. Further work will go into computer-supported assessment, which has already been implemented in the frame of MEETMATH, see [6]. Assessment is heavily based on randomly generated test exercises based on example patterns, where the power of a symbolic computation system in the background is essential for checking correctness of user solutions. Also, the use of *Theorema* provers for checking user answers can be investigated.
Theme: Designing and using Dynamic Mathematics environments

5. References

[1] R.B. Andrews, C.E. Brown, F. Pfenning, M. Bishop, S. Issar, and H.Xi. ETPS: A System to Help Students Write Formal Proofs. *Journal of Automated Reasoning*, 32:75–92, 2004.

[2] B. Buchberger. Should Students Learn Integration Rules? *ACM SIGSAM Bulletin*, 24(1):10–17, January 1990.

[3] B. Buchberger, A. Craciun, T. Jebelean, L. Kovacs, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz, and W. Windsteiger. Theorema: Towards Computer-Aided Mathematical Theory Exploration. *Journal of Applied Logic*, 2005. To appear.

[4] B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger. The Theorema Project: A Progress Report. In M. Kerber and M. Kohlhase, editors, *Symbolic Computation and Automated Reasoning (Proceedings of CALCULEMUS 2000, Symposium on the Integration of Symbolic Computation and Mechanized Reasoning)*, pages 98–113. St. Andrews, Scotland, Copyright: A.K. Peters, Natick, Massachusetts, 6-7 August 2000.

[5] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, and D. Vasaru. A Survey of the Theorema project. In W. Kuechlin, editor, *Proceedings of ISSAC'97 (International Symposium on Symbolic and Algebraic Computation, Maui, Hawaii, July 21-23, 1997), ACM Press 1997.*, pages 384–391, 1997.

[6] S. Saminger. MeetMATH — visualizations and animations in a didactic framework. In M. Borovcnik and H. Kautschitsch, editors, *Technology in Mathematics Teaching (Special groups and working groups). Proceedings of ICTMT 5, Klagenfurt (Austria)*, volume 26 of *Schriftenreihe Didaktik der Mathematik*, pages 217–222, Wien, 2002. öbv & hpt Verlagsgesellschaft.

[7] R. Sommer and G. Nuckols. A Proof Environment for Teaching Mathematics. *Journal of Automated Reasoning*, 32:227–258, 2004.