CreaComp: Experimental Formal Mathematics for the Classroom*

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, Altenbergerstraße 69, A-4040 Linz, Austria

CreaComp provides an electronic environment for learning and teaching mathematics that aims at inspiring the creative potential of students. During their learning process, students are encouraged to engage themselves in various kinds of interactive experiments, both of visual and purely formal mathematical nature. 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. 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.

Keywords: Automated Theorem Proving, Computer-Algebra Systems, Maths Education

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. Until now, computer-support focuses on *(symbolic) computations*, e.g. calculating limits, derivatives, integrals, polynomial and matrix computations, and *visualization*, e.g. plotting real-valued sequences or functions or other mathematical objects in 2D or 3D: the availability of powerful algorithms allows to solve certain problems even in situations when the method would require a tremendous computational effort when executed by hand or when no solution method is known to the students at a certain level of educa-

*This work is done within the project "CreaComp: E-Schulung von Kreativität und Problemlösekompetenz" at the Johannes Kepler University of Linz sponsored by the Upper Austrian government.

tion. Additionally, different visual representations of the computational objects contribute to qualitative data analysis and exploration of non-obvious mathematical relationships. It is clear that modern symbolic computation systems are powerful enough to carry out all computations done in high-school mathematics and most of the computations taught at undergraduate university level. Hence, the questions of which methods should be taught to students in the first place, and for which tasks we rely on the help of the computer, are of utmost importance. There is no absolute answer to this and the "White-Box Black-Box principle", see [3], 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 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. [1,12]. In any case, computers can support students and teachers, but they can also introduce new obstacles compared to traditional learning/teaching, see e.g. [8].

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 Section 2, 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 this, the CreaComp approach now adds *formal reasoning* as a third important component by integrating the *Theorema* system, see [5–7]. *Theorema* is designed to become a uniform environment in which a mathematician gets support during all periods of his/her mathematical occupation. For the CreaComp project, however, *Theorema* mainly provides the mathematical language and the possibility of fully automated or interactive generation of mathematical proofs. Both MeetMath and *Theorema* are based on the well-known computer algebra system *Mathematica*, see [14].

There is often the distinction between *experimental mathematics* and *formal mathematics* and computer-supported teaching/learning is primar-

ily associated with experimental mathematics whereas all proving-related mathematics is predominantly considered to be done "by hand". The discussion is then "experimental mathematics vs. formal mathematics" and the question is to what extent can formal mathematics be supplemented/accompanied/enriched/replaced by experimental mathematics and vice versa, see [2]. The CreaComp approach should be seen much more as "experimental formal mathematics", we aim 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 for automatically generating human-readable proofs, 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. Although CreaComp is not intended as an environment for learning how to prove, the white-box nature of *Theorema* proofs may even assist the student in acquiring some proving skills.

In the sequel, we will briefly introduce the constituent components MeetMath and *Theorema* and their combination in Section 2, the main part will be an exemplary case study presenting parts of a CreaComp unit on equivalence relations and set partitions in Section 3. Mathematically, the learning unit on equivalence relations and set partitions starts from

- the *definition of binary relations* and elementary properties such as *reflexivity*, *symmetry*, and *transitivity*,
- then introduces *classes* and *factor sets*,
- proceeds with *set partitions* and *induced relations*,
- develops the theorems that *the factor set of an equivalence forms a set partition* and that *the induced relation of a set partition is an equivalence relation*, and
- finally concludes with the theorems that *building the factor set (of an equivalence relation)* and *building the induced relation (of a set partition)* are *inverse* to each other.

At all stages, interactive visualization tools are provided to illustrate the new concepts and their properties in small and easily comprehensible examples. For all theorems as well as for all auxiliary lemmata required in the proofs, we allow for automated proofs in natural language to be generated interactively by the students, i.e. we provide an interface to *Theorema*-provers that allows the student to generate fully automated proofs.

2. The CreaComp Project: An Overview

2.1. *The Components MeetMath and Theorema*

MeetMath denotes a family of interactive mathematics course-ware based on *Mathematica* equipped with a Java^{TM}-based navigation. The basic course MeetMath@Business&Economy and the didactic concepts for MeetMath course-ware have been initially developed in the framework of the project IMMENSE (for *I*nteractive *M*ultimedia *M*athematics *E*ducation in *N*etworked universities for *S*ocial and *E*conomic sciences) initiated by the Johannes Kepler University Linz already in 1999. For more detailed information about the project and partners see [11].

The development of any course material including electronic supplement demands to focus not only on, possibly new, technology but on the corresponding didactic framework as well. MeetMath course units are structured as "didactical rooms", where different rooms reflect different phases of the learning process. Basically, four different types of didactical rooms have been distinguished:

(i) "Motivation/linking": motivation/linking addresses the students' knowledge about and their attitude to deal with certain content.
(ii) "Acquisition/confrontation": the central new concepts are presented in a complete and carefully paced argumentation. Relevant information, thoughts, and illustrations should be offered in a way that students are able to build up their own ideas and concepts and/or to modify existing, incorrect concepts.
(iii) "Strengthening": these rooms allow to verify and inspect newly developed concepts and ideas. Experiments and interactive elements enable students to stabilize the concepts and allow to identify incorrectly developed concepts by trial and error. Both success and failure are possible and allowed during strengthening.
(iv) "Assessment": assessment is a necessary tool for supervising the learning process. Since students are responsible for the development of their knowledge, they need some tools for measuring the progress and/or for finding out the status quo.

Theorema is a system that intends to bring computer-support during all phases of mathematical activity, such as proving, solving, and computing. The *Theorema* system provides a uniform language and logic, in which many of the above activities can be carried out, see [4]. The overall design principle of the *Theorema* system is to communicate with the user

in mathematical textbook style. The syntax of the language in *both input and output* is close to common mathematical notation, including special mathematical characters and two-dimensional syntax as typically used in mathematics. The *Theorema* language is a version of higher-order predicate logic with pre-defined basic mathematical objects such as numbers, sets, and tuples. The main focus in the development of the *Theorema* system over the past years has been put on the development of various general and special-purpose fully automated theorem provers.

Since *Theorema* is built on top of the well-known *Mathematica* system, we use the *Mathematica* notebook front-end as the user-interface for *Theorema*. When generating mathematical proofs, *Theorema* displays the full proof in a separate notebook document with each proof step explained in natural language. The structure of the proof is reflected in the cell structure of the proof notebook, so that the standard *Mathematica* technology of opening/closing nested cells by mouse-click can be used to collapse entire proof branches. This allows the reader to get an overview over the structure of the proof and then to zoom into parts of the proof by subsequently opening just the relevant cells. As an alternative to fully automated proof generation the *Theorema* system also allows for interactive proving, see e.g. [10]. For details on the *Theorema* system, the language, available provers, and applications of the *Theorema* system, we refer to the introductory papers [5–7]. Notably, an overview on CreaComp as an application of *Theorema* in education has been given in [13].

2.2. *The Combination of MeetMath and Theorema*

The most dangerous scenario of computer-supported mathematics is that the extensive use of computation and visualization leads to a tradition of "proof by inspecting particular examples" instead of "proof by mathematical proving". In our view, examples can and should accompany the development of mathematical content, they can contribute to shaping the students' intuition about mathematics, and the content presented in examples is typically well memorized. However, examples—even if many and well-chosen—can (almost) never substitute a proof of a proposition. It is one of the main goals of this project to highlight the importance of rigorous formal mathematical arguments in all facets of mathematical work, including for instance also software development.

The combination of MeetMath and *Theorema*, which is investigated in the frame of the CreaComp project, therefore aims at providing computer aid for visualization, computation, but most importantly also for

proving. Whereas computation and visualization ought to fertilize the *intuition* about mathematical objects, the proving phase should establish and enhance the *understanding* of mathematical argumentation, in particular that "validity in some examples" does not necessarily always mean "validity in *all cases*". An additional benefit of a theorem proving system as the student's assistant is that the students must think carefully about tacit assumptions they use in their argumentation, because the automated prover forces them to state all usable knowledge explicitly, see also e.g. [9]. 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 understand the evolution of mathematical theories much better. We consider this experience, even if maybe not necessary for a pure user of mathematics, as very enlightening for students of mathematics.

The interplay of MeetMath and *Theorema* is facilitated by the common underlying *Mathematica* technology, since both are based on the capabilities of the *Mathematica* notebook-frontend. CreaComp educational units are distributed in the form of *Mathematica* notebooks containing normal text intermixed with formal mathematical texts (definitions, theorems, lemmata, etc.) written in the *Theorema* language. Furthermore, we provide visualizations of the mathematical objects studied in the units. In addition to static plots we 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. 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.

Although the *Mathematica* notebook-frontend is often called a graphical user interface (GUI), *Mathematica* propagates a command-centered interaction pattern. In order to trigger a *Mathematica* computation, a command has to be typed into the notebook and then needs to be evaluated by pressing certain keys. For the CreaComp interface we make heavy use of interactive notebook elements like buttons and hyperlinks in order to prevent students from struggling with unfamiliar input syntax. Students' 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.

In the spirit of MeetMath's didactical framework, the interactive visualizations serve mainly for motivation and acquisition. After introducing a new mathematical concept, we provide tools aiming at graphical visualiza-

tions of important properties that are to be investigated in the current unit. The user can interactively "play" with the tool, the on-line help gives instructions which phenomena can and should be observed. These interactive tools are always designed in such a fashion that in addition to pre-defined examples they allow to run user-defined examples as well as randomly generated examples. A symbolic computation system is indispensable as the engine behind these tools, because an instructive visualization often needs the computation of the problem's solution in the background. Symbolic computation methods can be applied for generating pre-computed symbolic solutions depending on example parameters, such that for visualization of a random example only the example parameters need to be instantiated in the symbolic solution.

During this phase, the students get an intuition about the new concept and in the best case they observe some of the intended properties during their experiments. Still staying in an "acquisition room" (see Section 2.1) we then formulate some conjectures in the *Theorema* language, which looks very much like standard mathematical formula language. Changing into a "strengthening room", we then ask the user to prove the conjecture using *Theorema*. Again, we provide a button interface for the prover call in order not to confuse the students with syntax details thereby distracting them from their main focus, the proof. The learning goal and, thus, the user interaction in this phase consists of choosing the appropriate knowledge base and observing its influence on the generated proof. Furthermore, students can investigate possible modifications in the formulation of the conjecture and/or parts of the knowledge in order to obtain a successful proof.

CreaComp consists of several thematic units that are intended for use in standard undergraduate courses for studies in mathematics and computer science as well as in mathematics courses for non-mathematical studies, e.g. business, marketing, social sciences, etc. Units are available for basic set theory, relations, functions, real-valued sequences and limits, continuous functions, fast computations using modular arithmetic, polynomial interpolation, Markov chains, cryptology, Gröbner bases, and other topics to be developed. We do not discuss content selection, i.e. the CreaComp environment does not define rules, *what* should be taught and what not and, in particular, what should be presented as white-box and what should be considered black-box. Rather, the above topics have been selected by the authors to be taught as white-boxes, and CreaComp provides a common frame *how* to present these in the particular computer environment.

In the remainder of this paper, we want to illustrate the structures

described above in a case study showing parts of the CreaComp unit on equivalence relations and set partitions.

3. The Case Study: Equivalence Relations and Set Partitions

Following the didactic principles taken from MeetMath, the typical flow of a CreaComp unit is to

- motivate the students by some real-world example,
- present new concepts by defining new objects or properties,
- let the students experiment with the new entities on concrete data,
- guide the students in their experiments such that possibly they are able to conjecture new properties,
- guide the students in rigorous proofs of their conjectures.

We try to illustrate the flavor of computer-support given in a CreaComp unit in the example chapter on "equivalence relations and set partitions".

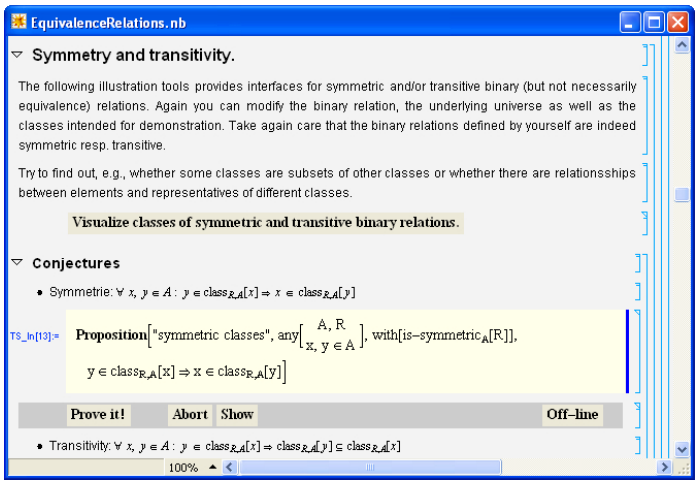

Fig. 1. A typical CreaComp educational unit.

Fig. 1 shows a screen-shot of a part of the notebook on equivalence relations containing the most important interactive interface elements. We see structured text containing inline mathematical formulae hierarchically grouped in nested cells intermixed with formal parts (e.g. definitions, theo-

rems, etc.) written in the *Theorema* language. *Theorema* blocks are written in *Mathematica* input cells and they 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. *Theorema* input can use most of the common mathematical notation[a], 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. Only occasionally, we leave parts of a formula blank and indicate with a "□"-placeholder that formula parts need to be inserted for the placeholder. CreaComp *interaction buttons* indicate the availability of an interactive experiment and they appear as gray boxes, the text above and on the button roughly explains the associated experiment. The unit shown in Fig. 1 contains an interaction button for visualizing classes of symmetric and transitive relations, which will be described in more detail in Section 3.1. Mathematical conjectures are formulated in *Theorema* language immediately followed by a CreaComp *prove panel* containing a prove-button, an abort-button, a show-button, and an off-line-button, see Section 3.2 for details.

3.1. *The Interface to Computer-Supported Experiments*

The interface to interactive experiments is always provided by so-called CreaComp interaction buttons. As an example, we describe the visualization tool behind the interaction button shown in Fig. 1. The notions "symmetry" and "transitivity" of a binary relation R on a universe A and the notion of a "class of an element x w.r.t. R and A" have been introduced earlier. At this point we want to investigate properties of classes when R is symmetric and/or transitive. When pressing the interaction button, a new window as shown in Fig. 2 appears on the screen. The window essentially contains 4 components:

[a]Some notation is supported already by standard *Mathematica*, like subscripts, fractions, summation/integration, and a bunch of special characters. Input syntax is configurable and *Theorema* uses this feature to support most of the common mathematical language. *Theorema* uses the braces $\{,\}$ for sets and anglebrackets $\langle,\rangle$ for tuples, which contrasts *Mathematica*, which uses braces for lists, i.e. tuples, and which does not have sets as distinguished objects.

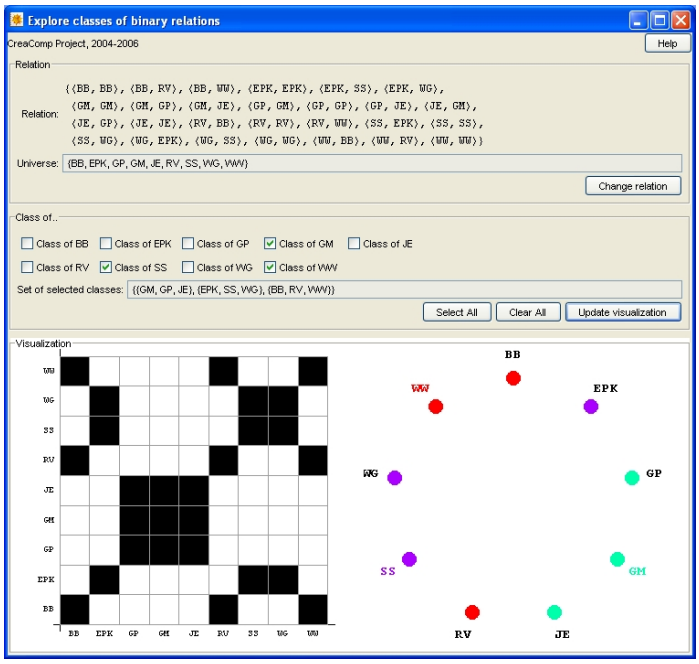

Fig. 2. Interactive visualization of classes.

(i) The top box displays the relation R as a set of pairs in *Theorema* notation and the universe set A. The relation R can be changed by either explicitly giving a new set of pairs or by having a random symmetric and/or transitive relation automatically generated by the system.

(ii) The middle box allows to select the classes to be visualized and it displays the respective classes as a set of sets in *Theorema* notation.

(iii) The bottom box shows graphical visualizations of the relation and the selected classes. The raster on the left corresponds to R's adjacency matrix and the arrangement of disks on the right indicates A's elements and each of the selected elements' class by color: each element is assigned a unique color shown by the color of its label and all elements in its class have their disk in the same color. Since intersecting classes are a key-feature to be discovered by this experiment, we display elements belonging to more than one class by a grey disk. The absence of grey disks in the visualization in Fig. 2 indicates disjoint classes in this example.

(iv) The help button in the top-right corner displays individual help for this

experiment by explaining which interactions can be made and which phenomena the student is expected to investigate.

Interactive visualizations are implemented using *Mathematica*'s GUIKit, which allows to build Java GUIs containing *Mathematica* data. In the example, the bottom graphics are re-calculated and re-drawn whenever there is user-interaction in one of the top boxes and this is the prototypical interaction pattern for CreaComp experiments: mathematical objects are visualized by *Mathematica* graphics that adapt to user-input given through intuitive interface elements such as check-boxes, radio-buttons, roll-down menus, etc., and special dialog windows that allow *Theorema* input that will be correctly parsed and processed before the respective graphics are re-generated. The on-line help 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 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.

For all interactive tools, their design is learner centered, i.e. the developer of a unit needs to decide which visualizations are instructive at which place in a unit. For each case it needs to be decided whether available visualizations in *Mathematica* (including the numerous extension packages) can be re-used or whether specialized graphics need to be programmed. Finally, the interaction patterns need to be developed and implemented in the Java interface in such a fashion that the desired learning process is supported best. We observe that the *Mathematica* programming part is almost neglectable (regardless of availability of visualizations in packages) compared to the interface design and programming, having in mind unexperienced users that should intuitively follow intended paths as inspired by the interface layout and its behavior, respectively.

3.2. *The Interface to Automated Proving*

We discuss the CreaComp interface to *Theorema* provers using the example shown in Fig. 1. In plain *Theorema*, the command to generate the proof would be

Prove[Proposition["symmetric classes"], by → SetTheoryPCSProver,
using → {Definition["symmetry"], Definition["class"]}, ProverOptions →
{GRWTarget → {"goal","kb"}, UseCyclicRules → True},SearchDepth→50]

In practice we have often realized, that the call of the appropriate prover with the appropriate options turns out to be a real hurdle, not only for students. The problem is not syntax but a successful call requires detailed knowledge about available provers, their exact names, their options, and the influence of these options on the generation of the proof. Since the primary goal of the course is to learn about equivalence relations rather than learning how to use *Theorema* (or *Mathematica*), we decided to hide the concrete prover calls and instead provide a uniform interface to *Theorema*'s automated provers by a so-called CreaComp prove panel, see Fig. 1.

The prove panel is a highlighted part in the notebook directly following a proposition to be proven, and it consists of buttons controlling a *Theorema* prover. The prove-button on the left has a call to a *Theorema* prove method with appropriate parameters and options as shown above 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 can compose the knowledge base in an interactive dialog[b]. Fig. 3 shows such a dialog window: it 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 code this information within the prove panel so that the knowledge base composition tool can access it from there.

The abort-button in the prove panel aborts a running proof, the show-button shows the proof attempt, typically after having aborted the prover. The off-line-button on the right-margin of the prove panel allows to view a pre-generated successful proof. Both pre-generated and live-generated proofs appear in a separate window as shown in Fig. 4. The proof comes in human-readable format and explains each proof step in natural language.

3.3. *The Entire Unit*

After having explained the interaction possibilities that are spread all over the material whenever appropriate we can now browse through the unit "Equivalence Relations and Set Partitions", which introduces the students

[b]This feature will not be used in every proof. Sometimes the appropriate knowledge base will be hardcoded in the prove-button.

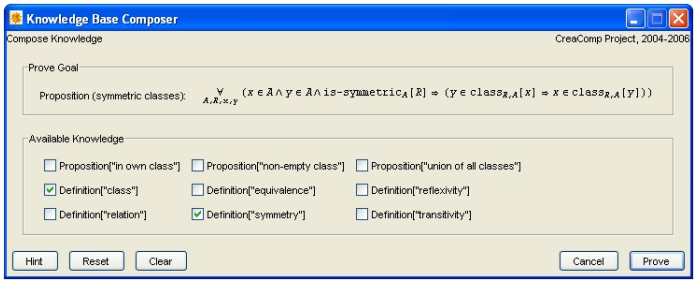

Fig. 3. Interactive knowledge base composition.

to the correspondence between equivalence relations and set partitions. First of all we introduce the mathematical objects to work with in this theory. The definitions are given in *Theorema* language and start with the definition of relations as a subset of some cartesian product. Since we want

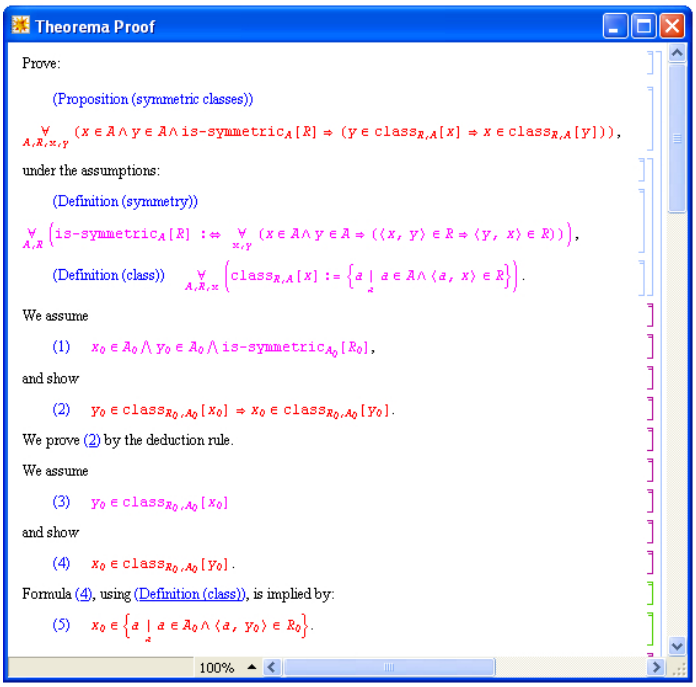

Fig. 4. *Theorema* proof.

to study classes and factor sets of equivalence relations, we restrict our theory to binary relations on some universe set. For other kinds of relations there are links to other CreaComp units, which focus on e.g. general relations or order relations. Furthermore, the first section of this unit defines the main properties interesting for equivalence relations, namely reflexivity, symmetry, and transitivity.

In order to develop some intuition on these properties of relations, we provide a visualization tool to illustrate these properties. The students can experiment with different relations, some are pre-defined, some are randomly generated or user-defined. This tool is similar to the one shown in Fig. 2 only that there is no mentioning of the concept of classes yet.

The next step is to introduce the concept of classes for binary relations. Traditionally, (equivalence) classes are defined for equivalence relations only. This is due to the nice properties of classes that can be proven in this setting. However, in the spirit of theory exploration, see e.g. [5], we find it interesting and natural to define *classes for arbitrary relations* and then study the dependence of properties of classes from properties of the underlying relation. In the *Theorema* language the definition of a class looks similar to common mathematical language, so the students can read and understand this definition easily without learning additional syntax:

Definition["class", any$[x, A, R]$,

$\mathsf{class}_{R,A}[x] := \{a \in A | \langle a, x\rangle \in R\}$]

Even for relations with only few properties, e.g. only reflexive, they can prove certain (weak) properties of classes; the stronger the properties of the relation the nicer the properties of the classes, until finally, towards the end of the unit, we find the classical results for equivalence classes. We think this evolution of the theory is an interesting aspect for students to experience.

Of course, conjectures proposed by the students or suggested in our material need not really be 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 conjecture the following:

Proposition["in own class", any$[A, R]$,

$\underset{x \in A}{\forall}\ x \in \mathsf{class}_{R,A}[x]$]

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 empha-

size 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-button.

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 $\langle x_0, x_0\rangle \in R_0$ needs to be proven for arbitrary x_0 and R_0. Thorough inspection of the available knowledge at that stage shows that only $x_0 \in A_0$ is known and the student might learn that the proposition as stated *cannot be proven* unless more is known about R_0. In general, there are various possibilities how to modify the setup for the next proof attempt: the proof goal can be modified, assumptions can be changed, new assumptions can be added, or even a different proving method could be applied. We assist the student by writing a skeleton of a new formula into the notebook at this point, thereby fixing which kind of modification is sensible. In the above example, a side-condition to the proof goal is the way to go. In *Theorema* this can be done using the with[...] expression and we provide

Proposition["in own class", any[A, R], with[$\square$],

$\underset{x\in A}{\forall}\ x \in \mathsf{class}_{R,A}[x]$]

where the student is asked to substitute some condition for the "$\square$"-placeholder. Of course, in order to succeed with the above proof, R_0 must have the property that $\langle x, x\rangle \in R_0$ is true for all $x \in A_0$, i.e. R_0 must be reflexive on A_0. Hence, they must put the side-condition "is-reflexive$_A[R]$" 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 and set partitions and they can investigate properties that turn a set of sets into a partition. Moreover, the concept of an *induced relation* is introduced, namely

Definition["induced relation", any[A, S],

$\mathsf{induced\text{-}relation}_A[S] := \{\langle x, y\rangle \underset{x,y\in A}{|} \underset{M\in S}{\exists}\ x \in M \wedge y \in M\}$]

In each step the procedure is similar:

- *introduce* new objects or properties,
- *visualize* the properties 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 factor-set$_A[R]$ is a partition of A and induced-relation$_A$[factor-set$_A[R]] = R$.
- if P is a partition of A then induced-relation$_A[P]$ is an equivalence relation and factor-set$_A$[induced-relation$_A[P]] = P$.

All these theorems including also all auxiliary lemmata necessary for compact proofs of the main statements are proved fully automatically within this learning unit. In order to demonstrate the readability of *Theorema* proofs, we show one proof in all details *as it is generated in the* Theorema *system*, compare its appearance also to Fig. 4.

Lemma["induced relation is transitive", any$[A, P]$, with[is-partition$_A[P]$], is-transitive$_A$[induced-relation$_A[P]]$]

In the knowledge base for this proof, we have the definitions of transitivity and induced relation and an auxiliary proposition proved earlier, namely

Proposition["intersecting classes are equal", any$[A, P]$, with[is-partition$_A[P]$], $\underset{X,Y\in P}{\forall} X \cap Y \neq \emptyset \Rightarrow X = Y$]

Proof. We assume

(1) is-partition$_{A_0}[P_0]$

and show

(2) is-transitive$_{A_0}$[induced-relation$_{A_0}[P_0]]$.

Formula (2), using (Definition (transitivity)), is implied by:

(3) $\underset{x,y,z\in A_0}{\forall} \langle x, y\rangle \in \text{induced-relation}_{A_0}[P_0] \wedge \langle y, z\rangle \in \text{induced-relation}_{A_0}[P_0] \Rightarrow \langle x, z\rangle \in \text{induced-relation}_{A_0}[P_0]$.

We assume

(4) $x_0 \in A_0 \wedge y_0 \in A_0 \wedge z_0 \in A_0 \wedge \langle x_0, y_0\rangle \in \text{induced-relation}_{A_0}[P_0] \wedge \langle y_0, z_0\rangle \in \text{induced-relation}_{A_0}[P_0]$

and show

(5) $\langle x_0, z_0\rangle \in \text{induced-relation}_{A_0}[P_0]$.

Formula (5), using (Definition (induced relation)), is implied by:

(8) $\langle x_0, z_0\rangle \in \{\langle x, y\rangle \underset{x,y\in A_0}{|} \underset{M\in P_0}{\exists} x \in M \wedge y \in M\}$.

In order to prove (8) we have to show

(9) $\underset{x,y\in A_0}{\exists} (\underset{M}{\exists} M \in P_0 \wedge x \in M \wedge y \in M) \wedge \langle x_0, z_0\rangle = \langle x, y\rangle$.

Since $x := x_0$ and $y := z_0$ solves the equational part of (9) it suffices to show

(10) $x_0 \in A_0 \wedge z_0 \in A_0 \wedge \underset{M}{\exists} M \in P_0 \wedge x_0 \in M \wedge z_0 \in M$.

Formula (10.1) is true because it is identical to (4.1)
Formula (10.2) is true because it is identical to (4.3)
Formula (4.4), by (Definition (induced relation)), implies:

(12) $\langle x_0, y_0\rangle \in \{\langle x, y\rangle \underset{x,y\in A_0}{|} \underset{M\in P_0}{\exists} x \in M \wedge y \in M\}$.

From (12) we know by definition of $\{T_x \underset{x}{|} P\}$ that we can choose an appropriate value such that

(13) $\underset{M}{\exists} M \in P_0 \wedge x1_0 \in M \wedge x2_0 \in M$,

(14) $\langle x_0, y_0\rangle = \langle x1_0, x2_0\rangle$.

Formula (14) simplifies to

(16) $x_0 = x1_0 \wedge y_0 = x2_0$.

By (13) we can take appropriate values such that:

(17) $M_0 \in P_0 \wedge x1_0 \in M_0 \wedge x2_0 \in M_0$.

Now, let $M := M_0$. Thus, for proving (10.3) it is sufficient to prove:

(21) $M_0 \in P_0 \wedge x_0 \in M_0 \wedge z_0 \in M_0$.

Formula (21.1) is true because it is identical to (17.1).
Formula (21.2), using (16.1), is implied by:

(22) $x1_0 \in M_0$.

Formula (22) is true because it is identical to (17.2).
Proof of (21.3) $z_0 \in M_0$: Formula (4.5), by (16.2), implies:

$\langle x2_0, z_0\rangle \in \text{induced-relation}_{A_0}[P_0]$

which, by (Definition (induced relation)), implies:

(23) $\langle x2_0, z_0\rangle \in \{\langle x, y\rangle \underset{x,y\in A_0}{|} \underset{M\in P_0}{\exists} x \in M \wedge y \in M\}$.

From (23) we know by definition of $\{T_x \underset{x}{|} P\}$ that we can choose an appropriate value such that

(24) $\underset{M}{\exists}\, M \in P_0 \wedge x3_0 \in M \wedge x4_0 \in M$,

(25) $\langle x2_0, z_0\rangle = \langle x3_0, x4_0\rangle$.

Formula (25) simplifies to

(27) $x2_0 = x3_0 \wedge z_0 = x4_0$.

By (24) we can take appropriate values such that:

(28) $M_1 \in P_0 \wedge x3_0 \in M_1 \wedge x4_0 \in M_1$.

Formula (21.3), using (27.2), is implied by:

(32) $x4_0 \in M_0$.

Formula (17.3), by (27.1), implies:

(33) $x3_0 \in M_0$.

From (28.2) together with (33) we know

(35) $x3_0 \in M_1 \cap M_0$.

From (35) we can infer

(36) $M_1 \cap M_0 \neq \emptyset$.

Formula (36), by (Proposition (intersecting classes are equal)), implies:

(37) $\underset{A,P}{\forall}$ is-partition$_A[P] \wedge M_0 \in P \wedge M_1 \in P \Rightarrow M_1 = M_0$.

Formula (1), by (37), implies:

(71) $M_0 \in P_0 \Rightarrow (M_1 \in P_0 \Rightarrow M_1 = M_0)$.

From (17.1) and (71) we obtain by modus ponens

(72) $M_1 \in P_0 \Rightarrow M_1 = M_0$.

From (28.1) and (72) we obtain by modus ponens

(73) $M_1 = M_0$.

Formula (32) is true because of (28.3) and (73). □

The proof as shown above will appear in a separate window and features interactive elements that cannot be rendered in the above "static reproduction": All formula references are active button elements, which will display the referenced formula in a separate window, proof goals and assumptions can easily be distinguished by color, see also Fig. 4, 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 the configuration used for the above proof, *Theorema* does not display every single proof step it applies. In this example, for instance, it tacitly splits conjunctions in the goal and the knowledge base into its parts. This explains formula labels referring to formulae not actually present in the proof, e.g. formula (4.1) refers to the first conjunct in formula (4).

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 [11]. 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 both generating the exercises as well as checking correctness of user solutions. Although proving forms an essential part of our approach to teaching, we plan to assess neither the students' performance in proving nor their use of an automated theorem proving system. Rather, we test facts about mathematical concepts and we hope that proving enhances students' understanding of the mathematics involved. As a different branch, the use of *Theorema* provers for checking user answers can be investigated.

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. J. M. Borwein. The Experimental Mathematician: The Pleasure of Discovery and the Role of Proof. *International Journal of Computers for Mathematical Learning*, 10(2):75–108, May 2005.

3. B. Buchberger. Should Students Learn Integration Rules? *ACM SIGSAM Bulletin*, 24(1):10–17, January 1990.
4. B. Buchberger. Symbolic Computation: Computer Algebra and Logic. In F. Bader and K.U. Schulz, editors, *Frontiers of Combining Systems, Proceedings of FROCOS 1996 (1st International Workshop on Frontiers of Combining Systems), March 26-28, 1996, Munich*, volume 3 of *Applied Logic Series*, pages 193–220. Kluwer Academic Publisher, Dordrecht - Boston - London, The Netherlands, 1996.
5. 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*, 4(4):470–504, 2006. ISSN 1570-8683.
6. 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.
7. 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*, pages 384–391, 1997. ISBN 0-89791-875-4.
8. P. Drijvers. Learning Mathematics in a Computer Algebra Environment: Obstacles are Opportunities. *Zentralblatt für Didaktik der Mathematik*, 34(5):221–228, 2002.
9. G. Hanna. Proof, Explanation and Exploration: An overview. *Educational Studies in Mathematics*, 44:5–23, 2000.
10. Florina Piroi and Temur Kutsia. The Theorema Environment for Interactive Proof Development. In G. Sutcliffe and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning. Proceedings of the 12th International Conference, LPAR'05*, volume 3835 of *Lecture Notes in Artificial Intelligence*, pages 261–275. Springer Verlag, 2005.
11. 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.
12. R. Sommer and G. Nuckols. A Proof Environment for Teaching Mathematics. *Journal of Automated Reasoning*, 32:227–258, 2004.
13. R. Vajda. E-training of Formal Mathematics: Report on the CreaComp Project at the University of Linz, June 26 2006. Contributed talk at ACA 2006.
14. Stephen Wolfram. *The Mathematica Book*. Wolfram Media, Inc., 5th edition, 2003.