Theorema 2.0: A Graphical User Interface for a Mathematical Assistant System

Wolfgang Windsteiger
RISC, JKU Linz
4232 Hagenberg, Austria

Abstract

Theorema 2.0 stands for a re-design including a complete re-implementation of the *Theorema* system, which was originally designed, developed, and implemented by Bruno Buchberger and his *Theorema* group at RISC. In this paper, we present the first prototype of a graphical user interface (GUI) for the new system. It heavily relies on powerful interactive capabilities introduced in recent releases of the underlying Mathematica system, most importantly the possibility of having dynamic objects connected to interface elements like sliders, menus, check-boxes, radio-buttons and the like. All these features are fully integrated into the Mathematica programming environment and allow the implementation of a modern interface comparable to standard Java-based GUIs.

1 Introduction

Although *Theorema 1.0*, see e.g. [1, 2, 3, 5], has been widely acknowledged as a system with one of the nicer user interfaces, we could observe that outsiders or beginners still had a very hard time to successfully use the *Theorema* system. This was true for entering formulae correctly as well as for proving theorems or performing computations. *Theorema 1.0* as well as *Theorema 2.0* are implemented on top of Mathematica, one of the leading computer algebra systems developed by Wolfram Research. Thus, the principal user interface to *Theorema* is given by the Mathematica notebook front-end. While the 2D-syntax for mathematical formulae available since Mathematica 3, see [6], is nice to read, a wrongly entered 2D-structure has always been a common source of errors. More than that, the user-interaction pattern in *Theorema 1.0* was the standard 'command-evaluate' known from Mathematica, meaning that every action in *Theorema 1.0* was triggered by the evaluation of a certain *Theorema* command implemented as a Mathematica program. As an example, giving a definition meant evaluation of a Definition[...]-command, stating a theorem meant evaluation of a Theorem[...]-command, proving a theorem meant evaluation of a Prove[...]-command, and performing a computation meant evaluation of a Compute[...]-command. For the new *Theorema 2.0* system, we envisage a more 'point-and-click'-like interface as one is used to from modern software tools like an emailing-environment or office software.

The main target user-group for *Theorema* are mathematicians, who want to engage in formalization of mathematics or who just want to have some computer-support in their proofs. Also for students of mathematics or computer science and for teachers at universities or high schools the system should be a tool helping to grasp the nature of proving. Therefore, nice two-dimensional input and output of formulae in an appearance like typeset or handwritten mathematics is an important feature. On the other hand, the unambiguous parsing of mathematical notation is non-trivial already in 1D, supporting 2D-notations introduces some additional difficulties.

Theorema is a multi-method system, i.e. it offers many different proving methods specialized for the proof task to be carried out. The main focus is mainly on a resulting proof that comes as close as possible to a proof done by a well-educated mathematician. This results in a multitude of methods, each of them having a multitude of options to fine-tune the behaviour of the provers.

This is on the one hand powerful and gives many possibilities for system insiders, who know all the tricks and all the options including the effect they will have in a particular example. For newcomers, on the other hand, the right pick of an appropriate method and a clever choice of option settings is often an insurmountable hurdle. The user interface in *Theorema 2.0* should make these selections easier for the user. Furthermore, there should be the possibility to extend the system by user-defined reasoning rules and strategies.

Finally, the integration of proving, computing, and solving *in one system* will stay a major focus also in *Theorema 2.0*. Compared to *Theorema 1.0*, the separation between *Theorema* and the underlying Mathematica system is even stricter, but the integration of Mathematica's computational facilities into the *Theorema* language has improved.

Some of the features described in this paper rely or depend on their implementation in Mathematica. This requires a certain knowledge of the principles of Mathematica's programming language and user front-end in order to understand all details given below. The rest of the paper is structured as follows: the first section describes the new features in recent releases of Mathematica that form the basis for new developments in *Theorema 2.0*, in the second section we introduce the new *Theorema* user interface, and in the conclusion we give a perspective for future developments.

2 New in Recent Versions of Mathematica

We describe some of the new developments in recent Mathematica releases that were crucial in the development of *Theorema 2.0*.

2.1 Mathematica Dynamic Objects

Typical graphical user interfaces nowadays are implemented in the Java programming language and its derivations or extensions. Earlier versions of Mathematica offered the so-called *GUIKit* extension, which was based on Java and used MathLink for communication between Mathematica and the generated GUI. We used GUIKit earlier for the development of an educational front-end for *Theorema*, see [4], but the resulting GUI was cumbersome to program, unstable, and slow in responding to user interaction. As of Mathematica version 6, and then reliably in version 7, see [7], the concept of *dynamic expressions* was introduced into the Mathematica programming language and fully integrated into the notebook front-end. Dynamic expressions form the basis for interactive system components, thus, they are *the* elementary ingredient for the new *Theorema 2.0* GUI.

In short, every Mathematica expression can be turned into a dynamic object by wrapping it into `Dynamic`. As the most basic example, `Dynamic[`*expr*`]` produces an object in the Mathematica front-end that displays as *expr* and automatically updates as soon as the value of one of the parameters, on which *expr* depends, changes. In addition, typical interface elements such as sliders, menus, check-boxes, radio-buttons, and the like are available. On the one hand, the appearance of these elements depends on values of variables connected to them. On the other hand, every action performed on them, e.g. clicking a check-box or radio-button, changes the value of the respective variable. The set of available GUI objects is very rich and there is a wide variety of options and auxiliary functions in order to influence their behaviour and interactions. These features allow the construction of arbitrarily complicated dynamic interfaces and seem to constitute a perfect platform for the implementation of an interface to the *Theorema* system. A big advantage of this approach is that the entire interface programming can be done inside the

Mathematica environment, which in particular brings us a uniform interface on all platforms from Linux over Mac until Windows for free.

2.2 Cascading Stylesheets

Stylesheets are a means for defining the appearance of Mathematica notebook documents very similar to how stylesheets work in HTML or word processing programs. The mere existence of a stylesheet mechanism for Mathematica notebooks is not new, but what is new since version 6 is that stylesheets are cascading, i.e. stylesheets may depend on each other and may inherit properties from their underlying styles just like CSS in HTML. This of course facilitates the design of different styles for different purposes without useless duplication of code. The more important news is that stylesheets can now, in addition to influencing the appearance of a cell in a notebook, also influence the *behaviour* of a cell. This is a feature that we always desired since the beginning of *Theorema*: an action in Mathematica is always connected in some way to the evaluation of a cell in a notebook, and we wanted to have different evaluation behaviour depending on whether we want to e.g. prove something, do a computation, enter a formula, or execute an algorithm. Using a stylesheet, we can now define computation-cells or formula-cells, and the stylesheet defines commands for their pre-processing, evaluation, and post-processing.

Cascading is a nice feature for maintenance of stylesheets also, because is allows to separate settings responsible for behaviour from those for appearance. This is convenient for a system user, who typically would never wish to influence behaviour, because the functioning of the system relies on proper settings in this area. Still, adding new styles for different tastes and occasions such as presentations or lecture notes can be added with ease.

3 The *Theorema* Interface

As said, the Mathematica notebook front-end is the primary user interface for *Theorema*. "Working in *Theorema*" consists of *activities* that themselves require certain *actions*. As an example, a typical activity would be "to prove a formula", which requires actions such as "selecting a proof goal", "composing the knowledge base", "choosing the inference rules and a proof strategy", etc. The central new component in *Theorema 2.0* is the *Theorema commander*; it is the GUI component that guides and supports all activities. Of course, most activities work on mathematical formulae in one or the other way. Formulae appear as definitions, theorems or similar containers and are just written into Mathematica/*Theorema* notebook documents that use one of the *Theorema* stylesheets. We call the collection of all available formulae the *Theorema environment*. Composing and manipulating the environment is just another activity and therefore supported from the *Theorema* commander. The second new interface component in *Theorema 2.0* is the *virtual keyboard*; its task is to facilitate the input of math expressions, in particular 2D-input. Figure 1 shows a screen shot of *Theorema 2.0* with a *Theorema*-styled notebook top-left, the *Theorema* commander to its right, and the virtual keyboard underneath.

3.1 The *Theorema* Environment

The *Theorema* environment is composed in *Theorema*-styled Mathematica notebooks, which have all the capabilities of normal Mathematica notebooks plus the possibility to process expressions in *Theorema* language inside environment cells. This means that *Theorema* expressions are embedded in a full-fledged document format for mathematical writing. Mathematica notebooks consists of hierarchically arranged cells, whose nesting is visualized with cell brackets on

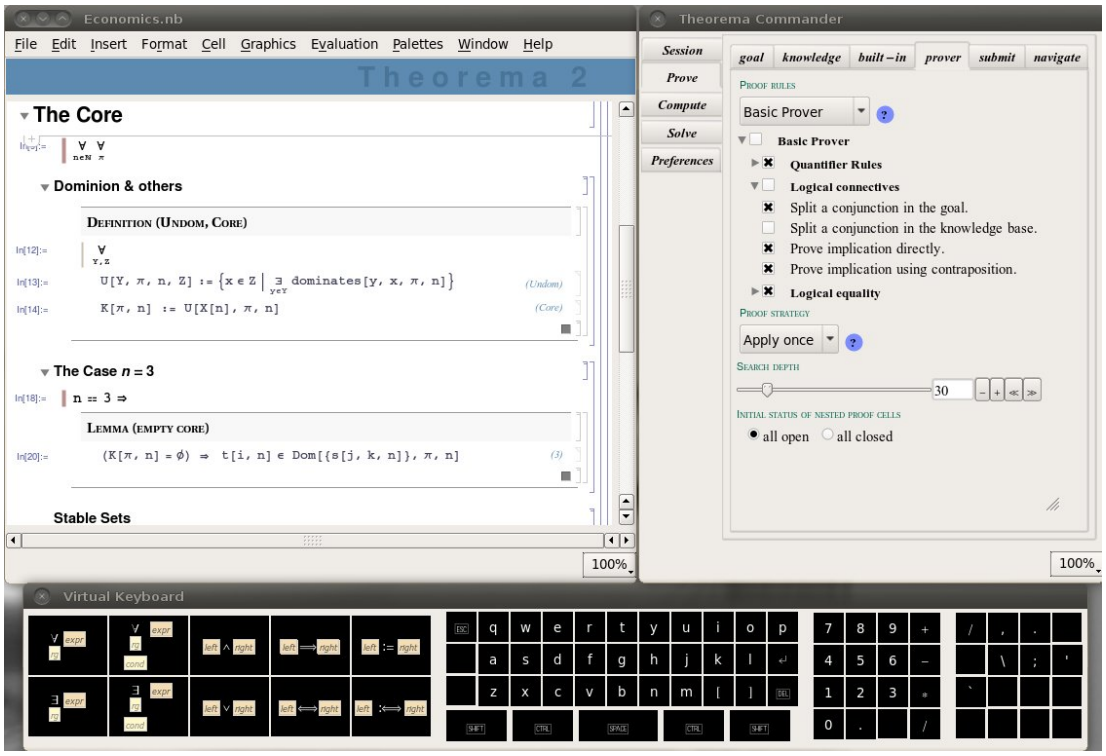

Figure 1: The *Theorema 2.0* GUI

the right margin of the notebook. Figure 1 shows a notebook using a stylesheet that renders the cell brackets with thin blue lines and displays section headings with a small open/close icon to their left for quick opening and closing entire section blocks. Note in particular that each environment forms a group for its own.

Environment cells contain mathematical expressions in *Theorema* syntax with an additional label. If no label is given by the user, an incremental numerical label is automatically assigned. If a chosen label is not unique within a notebook, the user is warned but uniqueness is not enforced. Invisible for the user, the formula is stored in the *Theorema* environment using a datastructure that carries a *unique key* for each formula consisting of the absolute pathname of the file, in which it was given, and the unique cell-ID in that notebook, which is provided by the Mathematica front-end. The formula key allows to uniquely reference each formula in the current environment. As we will explain later, the user never sees nor needs the concrete formula key explicitly.

In mathematical practice, universal quantification of formulae and conditioning is often done on a global level. As an example take definitions, which often start with a phrase like "Let $n \in \mathbb{N}$. We then define ...", which in effect expresses a universal quantifier for n plus the condition $n \in \mathbb{N}$ for all notions introduced in the current definition. For this purpose, we provide *declaration cells*, which may either contain one ore several "orphaned" universal quantifiers (each containing a variable and an optional condition, but missing the formula, to which they refer) or an "orphaned" implication (missing its right hand side). The idea is that the scope of these quantifiers or implications ranges from their location in the notebook to the

end of the nearest enclosing cell group. In the example in Figure 1, this is used in DEFINITION (UNDOM, CORE) with a universal quantifier for Y and Z valid for both formulae inside this definition. The cell grouping defined in the stylesheet ensures that a definition gets its own cell group that limits the scope of the quantifier.

We generalized the idea of declarations inside an environment towards declarations inside an arbitrary cell group. This has the effect that a declaration cell can be put anywhere in a notebook, and its scope ranges as described above from its position to the end of the nearest enclosing cell group. In Figure 1, this is used twice:

1. There is a '$\underset{n\in\mathbb{N}}{\forall}\ \underset{\pi}{\forall}$' at the beginning of Section 'The Core'. This means, that, without further mentioning, n and π are universally quantified with an additional condition $n \in \mathbb{N}$ in the entire section including all its subsections.

2. There is a '$n = 3 \Rightarrow$' in Subsection 'The Case $n = 3$', so that this condition on n affects only in this subsection.

At the moment of giving a formula to the system, i.e. evaluating the environment cell in Mathematica, all declarations valid at this position are silently applied and the actual formula in the *Theorema* environment has all respective quantifiers and implications attached to it just as if they were written explicitly with each formula. This comes very close to how mathematicians are used to write down things and this is very convenient. For bigger documents, one might loose the overview on which declarations are valid at some point. The *Theorema* commander gives some assistance in this situation: by just pressing a button one can obtain a list of all declarations valid at the current cursor position in the selected notebook. Also, you can always view the entire *Theorema* environment (with all formulae currently available including all quantifiers and conditions) from the *Theorema* commander.

3.2 The *Theorema* Commander

The *Theorema* commander, see Figure 1 top-right, is the main GUI component in current *Theorema 2.0*. It is a two-level tabview with activities on the first level and the corresponding actions for each activity on the second level. The first-level activity-tabs can be accessed through the vertical tabs on the left margin. Currently, the supported activities are 'Session', i.e. working on the *Theorema* environment, 'Prove', 'Compute', 'Solve', and 'Preferences'. As the system develops, this list may increase. For each of these activities, the respective actions can be accessed via the horizontal tabs on top. Moving through them from left to right corresponds to a wizard guiding the user through the respective activity. Proving is presumably the most involved activity and we will describe some ideas for its support in the next paragraph in more detail. The remaining parts of the *Theorema* commander are of similar fashion, we will only mention some highlights in the concluding paragraph of this section.

The 'Prove'-activity The example in Figure 1 displays the 'Prove'-tab. It shows actions such as 'goal', 'knowledge', etc. that just correspond to the actions required for proving a formula in *Theorema*, namely defining the proof goal, specifying the knowledge available in the proof, setting up built-in knowledge, and selecting the desired prover to be used. Defining the proof goal is as simple as just selecting a formula in an open notebook with the mouse. The selected formula is shown in the 'goal'-tab, and it changes with every mouse selection. Finally, the choice is confirmed by just pressing a button in the 'goal'-tab. From this moment on, whatever the mouse selects, the proof goal is fixed until the next confirmation.

Goal confirmation automatically proceeds to the tab for composing the knowledge base, see Figure 2 (left). The *knowledge browser* displays a tab for each open notebook or loaded knowledge archive[1]. In each tab, a hierarchical overview of the file/archive content showing only the section structure, environments, and formula labels is displayed. Simply moving the mouse cursor over the label opens a tooltip displaying the whole formula, clicking the label jumps to the respective position in the corresponding notebook/archive. Each entry in the browser has a check-box attached to its left responsible for toggling the selection of the respective unit. In this way, individual formulae, environments, sections, up to entire notebooks can be selected or deselected with just one mouse-click, and the formulae selected in this way constitute the knowledge base for the next prove call. The formula label displayed in the browser is only syntactic sugar, the check-box is connected to the unique key of each formula in the environment, see Section 3.1.

The next action within the 'Prove'-activity is the selection of built-in knowledge[2], see Figure 2 (right). The *built-in browser* works like the knowledge browser described above. Instead of section grouping we have (not necessarily disjoint) thematic groups of built-ins like sets, arithmetic, or logic.

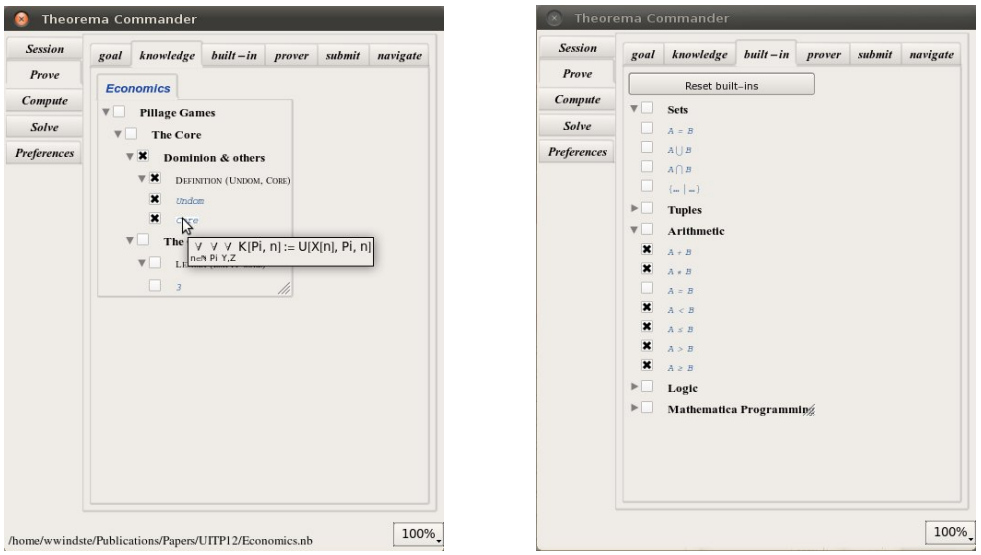

Figure 2: Graphical support for the 'Prove'-activity: the knowledge browser (left) and the built-in browser (right).

After having composed the relevant built-in knowledge, the user needs to select the prover. A *prover* in *Theorema 2.0* consists of a structured list of inference rules accompanied with a prove strategy. Accordingly, the 'prover'-tab, see Figure 3 (left), shows menus for choosing the inference rules and the strategy, respectively, together with short info panels explaining the current choice. A list of inference rules is a list, whose entries are individual inference rules or

[1] *Archives* are another new development in *Theorema 2.0*. An archive gives the possibility to store the formulae from a notebook efficiently in an external file, such that they can be loaded quickly into a *Theorema* session. We do not go into further details in this paper.

[2] With *built-in knowledge* we refer to knowledge built into the *Theorema* language semantics. As an example, '+' is by default an uninterpreted operator. Using some built-in knowledge one can link '+' to the addition of numbers available in the *Theorema* language. This is a feature inherited from *Theorema 1.0*.

themselves lists of inference rules. In addition, every list of inference rules has a name. There is no limit to the nesting of inference rule lists. The 'prove'-tab displays an *inference rule browser* corresponding to the selected rule list, which works like the knowledge browser described above using the rule list structure for the hierarchy and the list names instead of section titles. With the inference rule browser the user can efficiently deactivate individual inference rules, e.g. for influencing whether an implication will be proved directly or via contraposition. In addition, some options for the prover can be set or adjusted from this tab.

The next step is submitting the proof task. The respective tab collects all settings from the previous actions, in particular the chosen goal and knowledge base, and displays them for a final check. Hitting the 'Prove'-button submits all data to the *Theorema* kernel and proceeds to the 'navigate'-tab, see Figure 3 (right), which displays the corresponding proof tree as it develops during proof generation. The nodes in the proof tree differ in shape, color, and content depending on node type and status. As soon as the proof is finished, some proof information is written back into the notebook, in which the proof goal is defined. In addition to an indicator of proof success or failure and a summary of settings used at the time of proof generation, this information contains two important buttons:

1. A button to display the proof in natural language in a separate window. This feature is in essence the same as we had it in *Theorema 1.0*, see e.g. [5]. The 'navigate'-tab in the *Theorema* commander is connected to the proof display in that all labels in the proof tree representation are hyperlinks to the respective text blocks in the proof display describing the corresponding proof step, which is a nice possibility to navigate through a proof.

2. A button to restore all settings in the *Theorema* commander to the values they had at the time of proof generation, which is a quick way to rerun a proof.

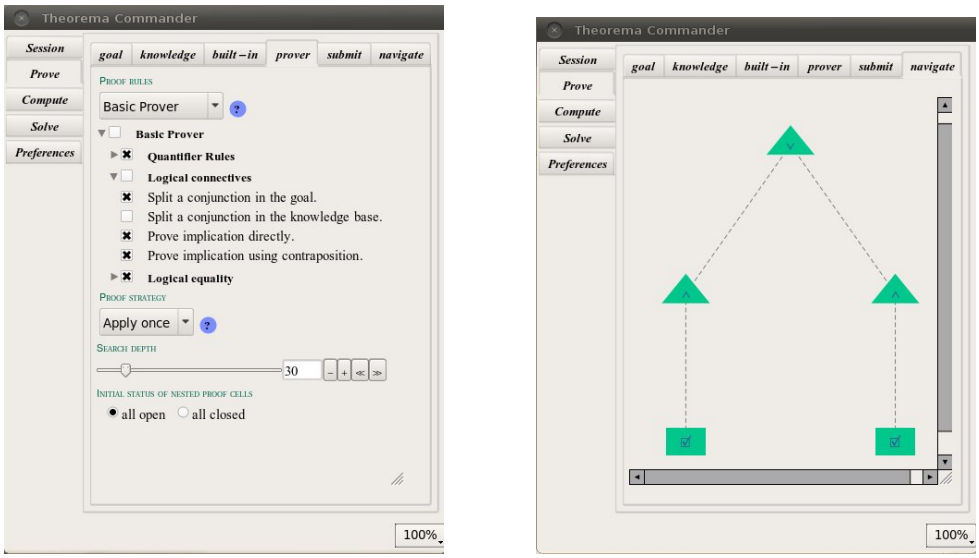

Figure 3: Graphical support for the 'Prove'-activity: the 'prover'-tab (left) and the 'navigation'-tab (right).

Other activities The 'Session'-activity consists of structuring formulae into definitions, theorems, etc., arranging global declarations, inspecting the environment, inputting formulae, and the development and maintenance of knowledge archives. The 'Compute'-activity contains setting up the expression to be computed, defining the knowledge base, and defining the built-in knowledge using knowledge- and built-in browsers as described for proving above. Knowledge selections for proving are independent from those used for computations. In the 'Preferences'-activity we collect everything regarding system setup, such as e.g. the preferred language. The entire GUI is language independent in the sense that no single english string (for GUI labels, button labels, explanations, tooltips, etc.) is hardcoded in its implementation, but all strings are constants, whose definitions are collected in several language-setup files. For a translation to a new language, only these files have to be copied and the english texts in them translated. The language selection menu in the 'Preferences' will immediately offer the new choice for the language, the user selects it, and voilà the GUI runs in the new language. Language support is important in particular for educational purposes that we envisage for *Theorema 2.0*.

An important detail that makes this approach possible is the decision to make the source code available under GPL license. This gives all users access to the source code and in particular the language-setup files. An attractive perspective for user contribution to the system could also be the development of new proof strategies. They are just Mathematica programs applying inference rules, and there is a rich library of *Theorema* programs that is ready for use in the implementation of strategies.

3.3 The Virtual Keyboard

The last component to be described briefly is the *virtual keyboard*, see the screenshot in Figure 1. Although much input can be given through buttons and palettes, such as buttons for frequently used expressions in the 'Session'-tab or the built-in Mathematica palettes, symbols or digits in an expression are most conveniently typed directly on the keyboard. When working with *Theorema 2.0* on a tablet computer or on an interactive white-board, however, e.g. in an educational context, we have no physical keyboard available. For situations like this we provide the virtual keyboard, which is an arrangement of buttons imitating a physical keyboard. It consists of a character block for the usual letters and a numeric keypad (numpad) for digits and common arithmetic operators like on common keyboards. As a generalization of the numpad, we provide a *sympad* (to the far right) and an *expad* (to the left) for common mathematical symbols and expressions, respectively. Using modifier keys like Shift, Mod, Ctrl and more, every key on the board can be equipped with many different meanings depending on the setting of the modifiers. We believe that the virtual keyboard is a very powerful input component for mathematical expressions, which will prove useful even in the presence of a physical keyboard.

4 Conclusion

Theorema 2.0 is currently under development. The components described in this paper are all implemented and the screenshots provided show a running and working system, it is not the sketch of a design. However, the interface presented here is incomplete and it will grow with new demands. From the experience with Mathematica's GUI components gathered up to now we are confident that all requirements for a modern interface to a mathematical assistant system can easily be fulfilled based on that platform.

Some of the features are implemented currently as 'proof of concept' and need to be completed in the near future to get a system that can be used for case studies. As an example,

the *Theorema* language syntax, from parsing via formatted output to computational semantics, is only implemented for a fraction of what we already had in *Theorema 1.0*. Due to the fact that the already implemented parts are the most complicated ones and that we paid a lot of attention to a generic programming style, there is hope that progress can be made quickly in that direction.

The bigger part of the work to be done is the re-implementation of all provers that we already had in *Theorema 1.0*. What we already have now is the generic proof search procedure and the mechanism of inference rule lists and strategies with their interplay. Two sample strategies, one that models more or less the strategy used in *Theorema 1.0* and another one that does a more fine-grained branching on alternative inference rules being applicable, are already available, but no report on their performance can be given at this stage. The big effort is now to provide all the inference rules for standard predicate logic including all the extensions that the *Theorema* language supports. As soon as this is completed we can engage in case studies trying out the system in some real-world theory formalization and in education, for which we plan a hybrid interactive-automatic proof strategy to be available.

References

[1] 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.

[2] 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.

[3] 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)*, pages 384–391. ACM Press, 1997.

[4] G. Mayrhofer, S. Saminger, and W. Windsteiger. CreaComp: Experimental Formal Mathematics for the Classroom. In Shangzhi Li, Dongming Wang, , and Jing-Zhong Zhang, editors, *Symbolic Computation and Education*, pages 94–114, Singapore, New Jersey, 2007. World Scientific Publishing Co.

[5] W. Windsteiger, B. Buchberger, and M. Rosenkranz. Theorema. In Freek Wiedijk, editor, *The Seventeen Provers of the World*, volume 3600 of *LNAI*, pages 96–107. Springer Berlin Heidelberg New York, 2006.

[6] S. Wolfram. *The Mathematica Book*. Wolfram Media/Cambridge University Press, third edition, 1996.

[7] Wolfram Research, Inc. Mathematica edition: Version 7.0, 2008. Champaign, Illinois.