User Interface Features in *Theorema*: A Summary ⋆

Florina Piroi

Florina.Piroi@risc.uni-linz.ac.at

Research Institute for Symbolic Computation (RISC),
Johannes Kepler University, Linz,
Schloss Hagenberg, 4232 Hagenberg i.M., Austria

Abstract. This paper presents the main features of *Theorema*'s user interface. We briefly describe how mathematical knowledge can be expressed in the *Theorema* Formal Text Language and how the knowledge can be used for proving, solving, computing. We illustrate how the system presents the proofs it generated and how the user can influence the proof search process interactively.

1 Introduction

Mathematical Knowledge Management (MKM) is a new research area at the intersection of mathematics and computer science. The "Call for Papers" of the First International Workshop on Mathematical Knowledge Management (held in September 2001 at Research Institute for Symbolic Computation, University of Linz, Austria) recognized the need for efficient, new techniques – based on sophisticated formal mathematics and software technology – for taking fruit of the enormous knowledge available in current mathematical sources and for organizing mathematical knowledge in a new way [11].

MKM aims both at the (partial or full) computer-support of all phases of the mathematical theory exploration cycle and at the structured storage of concepts, propositions, problems, and algorithms, such that they can be easily accessed, used and applied at a later time. In this sense, MKM is a logical activity: All formulae (axioms, definitions, etc.) must be available in the coherent frame of a logical system, e.g. some version of predicate logic. The main operation of MKM on these formulae is essentially formal reasoning (in particular formal proving), i.e. reasoning guided by explicit algorithmic rules [24]. This logic aspect is, in our opinion, fundamental to the future of MKM.

In [7] three main problems of the mathematical knowledge management area are identified, namely: retrieving mathematical knowledge, building up mathematical knowledge bases, and educating mathematicians to work efficiently with

⋆ Partially supported by the RISC PhD scholarship program of the government of Upper Austria, and by the Spezialforschungsbereich (SFB) grant F1302, Austrian Science Foundation (FWF).

and improve the existing knowledge bases. The paper also describes how each of these activities can be realized within *Theorema*.

Theorema is a software system that emphasizes on the *logical* aspect of MKM. The system aims at providing, in the frame of one uniform logic, computer support to all aspects of the mathematical exploration cycle: formalizing and introducing new notions, conjecturing and proving facts about the introduced notions, extracting algorithms from proved theorems, using these algorithms for computing and solving, writing (interactive) lecture notes, publishing. Some of the early papers on the design of the system are [2,3] and [4,5]. A progress report on *Theorema* is given in [8]; more recent papers on the current status of the system can be found on the web-site of the project (www.theorema.org). In the following, we describe the various interface features of the *Theorema* system, namely: proof presentation, formal language and syntax, interactive proving, syntactical manipulation of large groups of formulae. The interface features are the results of the common development efforts of the *Theorema* group members. The work has been done under the leadership of Bruno Buchberger.

2 Formal Language

Theorema is implemented on top of the computer algebra system *Mathematica* [27]. The document-centered front-end of *Mathematica* is suitable for writing mathematical documents. In one document, called notebook, text can be mixed with mathematical formulae and graphics. Additionally, the front-end of *Mathematica* can be configured through the *Mathematica* programming language. Using this feature, the *Theorema* system implements a two-dimensional syntax of mathematical formulae and a formal text language for defining new notions, properties of the notions, algorithms, for building up mathematical theories and using them for proving. The syntax of formulae is similar with the syntax used by mathematicians. For example, below is a formula in *Theorema* syntax:

$$\boxed{\forall_{f,B}\ (\texttt{is-bounded}[f,\ B] \Longleftrightarrow \forall_x\ |f[x]| \leq\ B).}$$

In order to describe mathematical knowledge, besides than writing down formulae in predicate logic, we need to be able to combine the formulae with auxiliary texts (labels, keywords – "Definition", "Proposition", etc.) and we need to be able to compose them hierarchically. For this reason we have implemented a "*Theorema* Formal Text Language" [8]. In this language, the formula given above can be introduced as:

$$\boxed{\begin{array}{l}\texttt{Definition}[\ \text{“}\texttt{Is bounded}\text{”}, \texttt{any}[\ f,\ B],\\ \qquad\qquad \texttt{is-bounded}[\ f,\ B] \Longleftrightarrow \forall_x |f[x]| \leq B \qquad \text{“} \leq \texttt{bound}\text{”}].\end{array}}$$

The Formal Text Language contains three categories of elements: *Environments*, for organizing knowledge in propositions, definitions, etc.; *Built-ins*, for assigning certain interpretations to certain symbols; and *Properties*, for asserting properties of operators. *Built-ins* and *properties* give us the possibility to fully control the interpretation of symbols. For example, we could chose that '$\oplus$' is interpreted to the *Mathematica* '+' (i.e. 'Plus'):

$$\texttt{Built-in}[\text{"}\texttt{My +}\text{"},\ \oplus \longrightarrow \texttt{Plus}].$$

Properties is used in a similar way.

In order to be able to process the knowledge expressed by the *Theorema* Formal Text Language we provide a "*Theorema* Command Language". Using this language we can *prove* propositions, we can *compute* values (or *simplify* formulae), or we can *solve* problems. For example, having the knowledge of the 'gcd' introduced as:

$$\texttt{Definition}[\text{"}\texttt{Greatest Common Divisor}\text{"}, \texttt{any}[m,\ n],\ \texttt{gcd}[m,\ n] = \texttt{max}_t[t|m \wedge t|n]],$$

we may want to prove that $\texttt{gcd}[m,\ n] = \texttt{gcd}[m-n,\ n]$:

$$\texttt{Prove}[\texttt{Proposition}[\text{"}\texttt{Euclid}\text{"}, \texttt{any}[m,\ n], \texttt{gcd}[m, n] = \texttt{gcd}[m-n, n]],\ \texttt{using} \longrightarrow \texttt{Definition}[\text{"}\texttt{Greatest Common Divisor}\text{"}]],$$

or we may want to compute the gcd of 12 and 18:

$$\texttt{Compute}[\texttt{gcd}[12, 18], \texttt{using} \longrightarrow \texttt{Definition}[\text{"}\texttt{Greatest Common Divisor}\text{"}],\ \texttt{built-in} \longrightarrow \texttt{Built-in}[\text{"}\texttt{Numbers}\text{"}]].$$

For a more detailed description of the *Theorema* Formal Text Language and *Theorema* Command Language see, for example, [8].

3 Label Management

In the *Theorema* system great emphasis is put on "systematic theory exploration", rather than "isolated theorem proving". Systematic mathematical theory exploration is explained by the concept of "exploration situations", concept introduced in [6]. The paper also introduces the parameters that characterize an exploration situation, namely: "known notions", "known facts about known notions", a "new notion", "axioms that relate the new notion with the known notions", and finally, "a class of goal propositions that completely explore the relation of the new notion with the known notions". Various approaches to systematic, computer-supported mathematical theory exploration are presented in [6].

Systematic theory exploration usually involves a large number of formulae (see, for example, the algorithm synthesis [9]). *Theorema* provides tools for the assignment of labels to formulae and collections of formulae so that blocks of mathematical knowledge stored into notebooks can be identified and combined in various ways without going into the "semantics" of the formulae. Concisely, the tools allow to reference specific parts of mathematical formulae collections. The referenced parts can be quickly composed into a new notebook or given as input to the formal reasoners of *Theorema* [24].

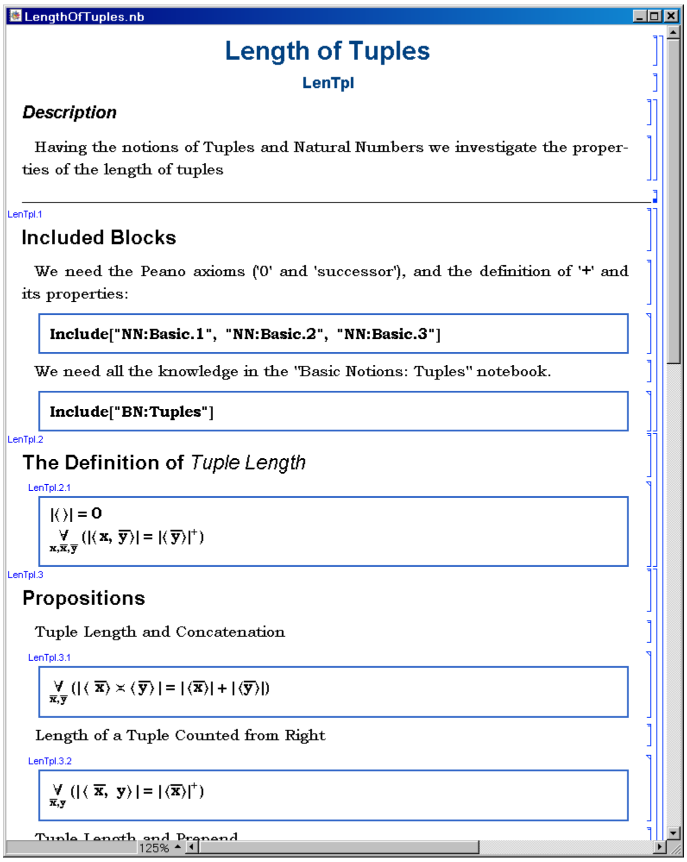

Fig. 1. A *Theorema* notebook.

For example, Figure 1 shows a notebook that collects formulae expressing knowledge about the length of tuples. The formulae in this notebook are part of the systematic theory exploration about tuples. We call such notebooks *Theorema* notebooks. Individual and groups of formulae have label attached: the groups of formulae with the header "Propositions" have the label "LenTpl.3", while the individual formulae in the group have the labels "LenTpl.3.1", "LenTpl.3.2", etc. The labels are automatically generated at user request. The notebook in Figure 1 also uses (parts of) other *Theorema* notebooks, namely parts of the *Theorema* notebook collecting formulae about natural numbers ("NN:Basic.1", "NN:Basic.2", "NN:Basic.3") and the notebook that contains basic notions about tuples ("BN:Tuples").

Knowledge stored in such *Theorema* notebooks can be used for reasoning. A usage example is:

$$\texttt{Prove}[\text{“}\texttt{LenTpl.3.1}\text{”}, \texttt{using} \longrightarrow \langle \text{“}\texttt{BN : Tuples}\text{”}, \text{“}\texttt{NN : Basic}\text{”}\rangle,\ \texttt{by} \longrightarrow \texttt{TupleEqIndProver}].$$

Users can also make temporary assignments of labels to specified collections of formulae. For example:

$$\texttt{Theory}[\text{“}\texttt{Tuples and Natural Numbers}\text{”},\ \langle \text{“}\texttt{BN : Tuples}\text{”}, \text{“}\texttt{NN : Basic.1}\text{”}\rangle]$$

and the above 'Prove' command can also be formulated as:

$$\texttt{Prove}[\text{“}\texttt{LenTpl.3.1}\text{”}, \texttt{by} \longrightarrow \texttt{TupleEqIndProver},\ \texttt{using} \longrightarrow \text{“}\texttt{Tuples and Natural Numbers}\text{”}].$$

4 Proof Presentation

The *Theorema* system contains various provers for general and specific domains: a propositional and a predicate logic prover [8], the Prove–Solve–Compute (PCS) prover for predicate logic with equality [25], induction provers over natural numbers and over lists [10], a set–theory prover [26], a Gröbner Bases based prover for boolean combinations of polynomial equalities and inequalities, etc.

The provers operate on "proof-situations" consisting of a set of assumptions and a goal (sequents with only one goal). Each prover is composed of a set of inference rules, each rule is typically expressed as a rewrite rule which transforms a proof-situation into one or more proof-situations. The proof-object is a tree representation of the development of the proof: each node corresponds to a proof-situation, while its successors in the tree correspond to the inferred proof-situations. The root of the proof-object corresponds to the initial proof - situation, while the leaves correspond to "proved" (successfully solved), "disproved", "failed" or "pending" (not yet resolved) proof-situations.

The proof-object is used as an internal representation of the proof. The external representation of the proof is displayed to the user in a separate notebook called "proof-notebook". The proof-notebook is produced by a post-processing of the proof-object, which generates a structured cell representation in a *Math-*

ematica notebook. On user request, the post-processing can invoke proof simplification procedures which remove superfluous branches and steps from the proof-object. The proof-notebook in the figure below presents part of the proof of a conjecture about equivalence relations.

The nested structure of the proof object is reflected by nested *Mathematica* cell brackets so that the user can open and close entire sub-trees of the proof object depending on which parts and sub-parts of the proof she wants to inspect. For example, in the figure above, the branch containing the proof of formula (18.1) is collapsed. Each inference step is typically represented as a phrase in natural language accompanied by some formulae and referring to some other formulae by their labels. Various color codes distinguish the (temporary) proof goals from formulae in the (temporary) knowledge base. Links to labelled formulae are realized as hyperlinks that display the formula referenced in a small auxiliary window. (See Figure 2.)

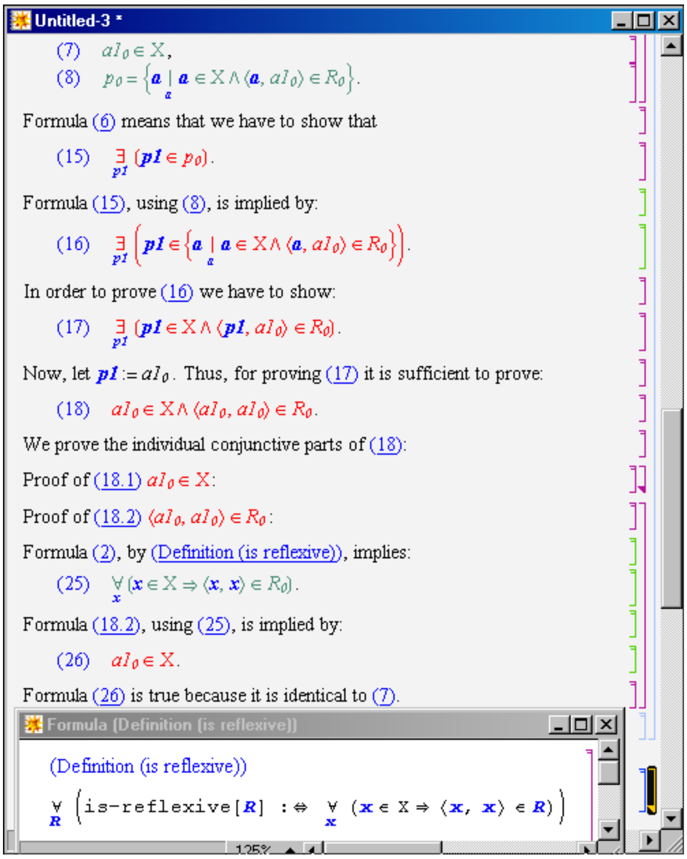

Fig. 2. A proof-notebook.

Using hyperlinks, a reader of *Theorema* proofs can avoid back and forth jumps in the proof, in order to understand the validity of a specific step. A "magnifying glass" tool also helps the user understand the validity of a certain step by composing a special window (focus window) which presents the user the relevant information for the proof step in question (formulae used, formulae generated, substitutions used, etc., see Figure 3 below).

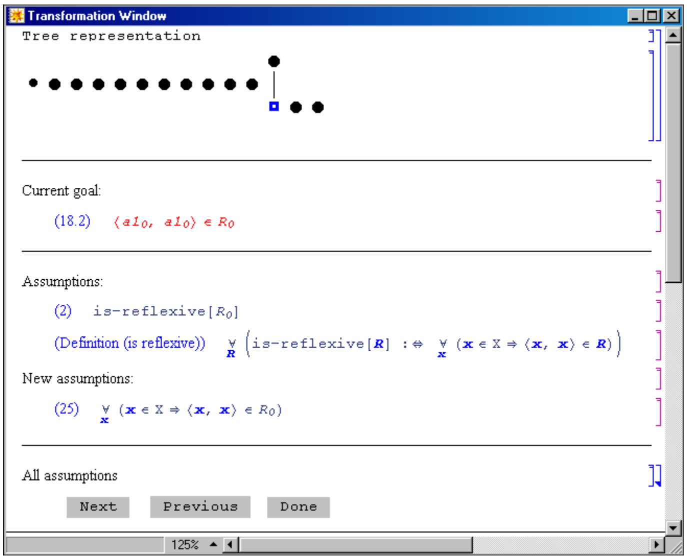

Fig. 3. A Focus Window.

5 Interactive Proving

Normally, a proof call will try to solve/prove the goal by applying the inferences and the heuristics implemented in the prover (or the combination of provers) which is used. One of the advantages of natural-style proving is that the user can easily understand and follow both the inference rules which are available in the prover, as well as the proof situation at any given moment. We capitalize on this advantage and, in order to improve the effectiveness of the *Theorema* provers, we implemented a general mechanism which allows a flexible interaction between the user and the system during the development of the proof. Using this, one can choose between fully automatic and interactive proof-development, one can easily navigate through the proof-object, and one can provide various hints (e.g. suitable instantiations) to the prover.

In the default proof-development style, the *Theorema* provers apply the inference rules automatically and repeatedly, until either a proof is obtained or no inferences can be applied anymore. The users only see the final output of this process, presented to them in the proof-notebook. In contrast, when interactively searching for proofs, the system is compelled to stop after each application of an inference rule, to present the produced proof sofar, and to wait for a decision from the side of user. While the system waits for a user decision to continue, in the interactive mode, the user can perform one or more of the following actions:

- select a proof situation in the proof;
- inspect a selected proof situation;
- add or remove assumptions in a selected proof situation;
- suggest instances for universally or existentially bound variables;
- add or remove branches in the proof;
- choose one among different provers to continue the proof, eventually change its options;
- make the system expand the proof by one inference rule application;
- ask the system to finish the proof without anymore user interventions;
- put an end to the proving session and exit the environment.

When proving interactively, the system displays the proof in a special proof-notebook and shows, additionally, several *Mathematica* style menu-palettes. The special proof-notebook displays the current (possibly unfinished) proof. The notebook also gives users the possibility to navigate inside the proof-tree in order to continue the proof on a certain branch, introduce new branches, select a node in the proof-tree, etc., actions triggered with the help of the menu-palettes. The menu-palettes contain various interactive commands and settings, each being represented by a button which triggers the respective action. Most of the commands require arguments (e.g. 'Start' needs a 'Prove' command, 'New Goal' needs a formula). The arguments are extracted from previous selections (i.e. clicks on cell brackets) in the notebooks used when proving interactively.

The interactive proving style is triggered by the evaluation of a '`StartInteractive`[ ]' call. This command will open the "*Theorema* Interactive" menu-palette from which all the other palettes can be invoked. The proving process is started by a click on the 'Start' button of the "*Theorema* Interactive" palette, after the user has selected a cell in a notebook that contains a 'Prove' call. As a small example, in Figure 4 we can see three notebook windows and several menu-palettes. In the "InteractiveExample.nb" window we can see the part of the notebook that contains the command starting the interactive proving environment and a 'Prove' call. From the "*Theorema* Interactive" menu-palette we have opened the "Proof Operations" palette and the "Debug" palette by clicks on the 'Advanced Op' and 'Debug' buttons, respectively. The unfinished proof shown in "The Proof Window" proof-notebook, is the result of executing the 'Start' command for the mentioned 'Prove' call, then several 'Next' commands, and a '$\forall$' Inst' command. The '$\forall$ Inst' command instantiated ϵ with ϵ_0 in formula '(4)' resulting in formula with the label '(Added 1)'. The instance ϵ_0 is provided by the user via a dialog window like the one presented in Figure 5.

The window in the lower left part of Figure 4, "Debug Messages", is used for displaying information about the content of proof-situations. The information

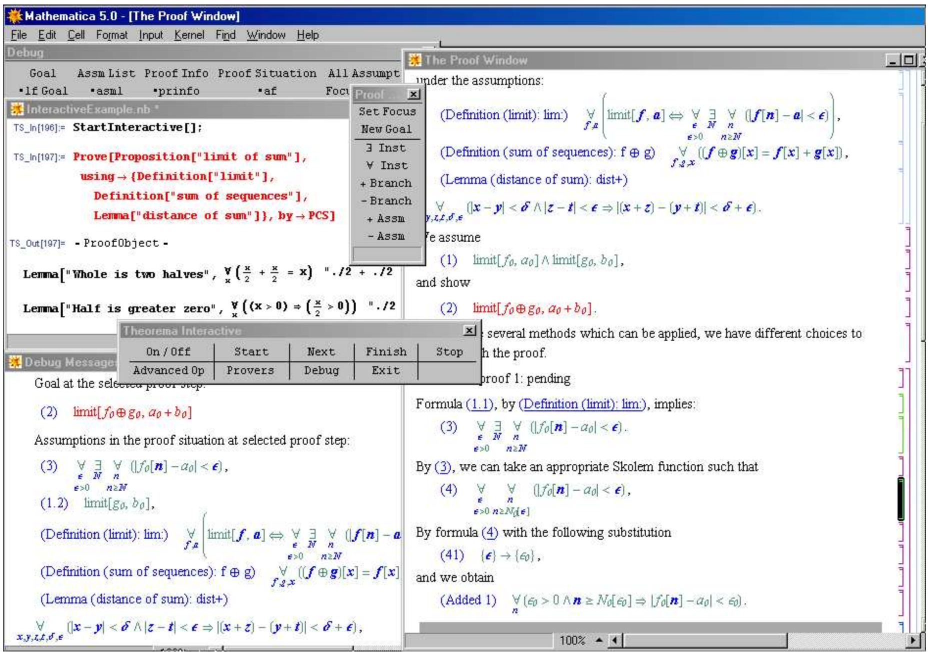

Fig. 4. A screen shot of the interactive environment.

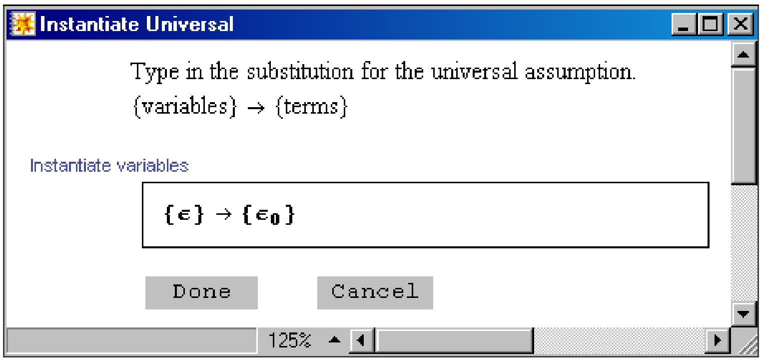

Fig. 5. Variable instantiation.

can be displayed in both user-friendly manner (as in Figure 4) or in *Theorema* internal form.

For a detailed description of interactive proving with *Theorema* see [23].

6 Final Words

We have briefly described the user interface features of the *Theorema* system. The work presented is the outcome of the common efforts of the members of the *Theorema* group over the last years under the leadership of Bruno Buchberger. The authoress' contribution is mainly in developing and improving the features described in sections 3, 4 and 5. We have underlined the points the system puts an emphasis on: a formal text language, "systematic theory exploration" over "isolated theorem proving", automated proof generation over proof checking, human-readable proof presentation.

Presenting proofs in natural language is a subject of interest in other systems, too. For example, in [12] the author describes a procedure that annotates CoQ proofs and then generates an english text of the proof. A verbalization of Nuprl proofs, described in [16], uses a language generator that has two components: a content planner which selects the information that should be included in the text, and a linguistic component that maps concepts to words and builds sentences. In PROVERB [17], the proof is first transformed to an adequate level of abstraction in which certain sequences of low-level proof steps are replaced by one higher level proof step. The abstracted proof is, then, processed and a natural language presentation is generated. PROVERB is embedded in the environment of the interactive prover Omega [1].

It seems that label management in the sense specified in Section 2 is not an explicit goal in the current (MKM) systems. Within most of the proving systems, the users are typing their documents in an Emacs-based editor or something similar (see for example, Mizar [22], HOL [15], CoQ [13]). Where translators are provided, the files can be stored, later, in LATEX, MathML or OpenMath formats. In this form, documents produced by proving assistants can be included in libraries of digital mathematics like HELM [14] and MBase [20]. Systems that concentrate on representing and publishing mathematics (on the web) make use of document translators and formulae editing tools that translate formulae and documents to different formats. For example, the JOME OpenMath Editor [19] creates and manipulates OpenMath objects, and within ActiveMath [21], jEditOQMath is a package of tools for editing and managing documents in OMDoc format [18].

Theorema is a system that integrates different general and special provers, solvers and simplifiers into one coherent system. At the time being, the *Theorema* system does not provide an interactive mode for computing and solving. Proving interactively gives users means guide the proof generation, as described in Section 5, but does not allow them to chose among inference rules to be applied, as most of the interactive systems do. This is a subject to further developments of the system.

References

1. C. Benzmüller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, A. Meier, E. Melis, W. Schaarschmidt, J. Siekmann, V. Sorge. OMEGA: Towards a Mathematical Assistant. In W. McCune, editor, *Automated Deduction - CADE 14*, volume 1249 of Lecture Notes in Artificial Intelligence, Springer-Verlag, July 1997. Townsville, North Queensland, Australia.
2. B. Buchberger. Proving, Solving, Computing. In T.Ida, Y,Guo, editors, *Proceedings of Multiparadigm Logic Programming Conference*, Bonn, Germany, Springer Vienna – New York. September, 6 1996. Invited talk.
3. B. Buchberger. Using *Mathematica* for Doing Simple Mathematical Proofs. In *4th International Mathematica User's Conference*. Tokyo, Wolfram Media Publishing, November 1996.
4. B. Buchberger. *Mathematica*: A System for Doing Mathematics by Computer?. In A. Miola, M. Temperini, editors, *Advances in the Design of Symbolic Computation Systems*, Springer Vienna – New York, 1997, pp. 2–20. Invited talk at the DISCO'93 Conference, Gmunden, Austria.
5. B. Buchberger. *Theorema*: Theorem Proving for the Masses Using *Mathematica*. Invited talk at the Worldwide *Mathematica* Conference, Chicago, USA, June 1998.
6. B. Buchberger. Theory Exploration with *Theorema*. *Analele Universitatii Din Timisoara, Ser. Matematica-Informatica*, Vol. XXXVIII, Fasc.2, 2000, (Proceedings of SYNASC 2000, 2nd International Workshop on Symbolic and Numeric Algorithms in Scientific Computing, Oct. 4–6, 2000, Timisoara, Romania, T. Jebelean, V. Negru, A. Popovici eds.), pp. 9–32.
7. B. Buchberger. Mathematical Knowledge Management Using *Theorema*. In B. Buchberger, O. Caprotti, editors, *Proceedings of the First International Workshop on Mathematical Knowledge Management: MKM'2001* RISC, A–4232 Schloss Hagenberg, September 24–26, 2001. ISBN 3–902276–01–0.
8. B. Buchberger, C. Dupré, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, W. Windsteiger. The *Theorema* Project: A Progress Report. In M. Kerber, M. Kohlhase, editors, *Symbolic Computation and Automated Reasoning. Proceedings of CALCULEMUS 2000, Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, pp 98–113, St. Andrews, Scotland, A.K. Peters, Natick, Massachusetts, August 2000. ISBN 1–56881–145–4.
9. B. Buchberger, A. Craciun. Algorithm Synthesis by Lazy Thinking: Examples and Implementation in *Theorema*. In F. Kamareddine, editor, *Proceedings of the Mathematical Knowledge Management Workshop*, Edinburgh, Nov. 25, 2003, volume 93 of Electronic Notes in Computer Science, pages 24–59, Elsevier, February 2004. www.elsevier.com/locate/entcs. ISBN 044451290X.
10. B.Buchberger, D.Vasaru. *Theorema*: The Induction Prover for Lists. In B. Buchberger, T. Ida, D. Vasaru, editors, *Proceedings of the First International Theorema Workshop*, June 9–10, 1997, RISC, Hagenberg, Austria, RISC Report 97–20.
11. Call for Papers: First International Workshop on Mathematical Knowledge Management, RISC, A–4232 Schloss Hagenberg, September 24–26, 2001. http://www.risc.uni-linz.ac.at/institute/conferences/MKM2001/call_for_papers.html
12. Y. Coscoy. A Natural Language Explanation for Formal Proofs. In. C. Retoré, editor, *Logical Aspects of Computational Linguistics: First International Conference, LACL'96*. Nancy, France, September 1996. Selected Papers. Lecture Notes in Computer Science, volume 1328, pp. 149–167. Springer Verlag Heidelberg. ISSN: 0302–9743.

13. The Coq proof assistant. http://coq.inria.fr/coq-eng.html.
14. Helm: Hypertextual Electronic Library of Mathematics. http://helm.cs.unibo.it/.
15. HOL 4 homepage: http://hol.sourceforge.net/
16. A. M. Holland-Minkley, R. Barzilay, R. L. Constable. Verbalization of High-Level Formal Proofs. In *Proceedings of Sixteenth National Conference on Artificial Intelligence*, July 18–22, 1999, Orlando, Florida. pp. 277–284, 1999.
17. X. Huang, A. Fiedler. Presenting machine-found proofs. In M. McRobbie, J. Slaney, editors, *Proceedings of the 13th Conference on Automated Deduction*, New Brunswick, NJ, USA, Lecture Notes in Artificial Intelligence, volume 1104, pp. 221–225, Springer Verlag, 1996.
18. jEditOQMath Tools Package. http://www.activemath.org/projects/jEditOQMath/.
19. JOME: Java OpenMath Editor. http://mainline.essi.fr/wiki/bin/view/Jome/WebHome. Written by Laurent Dirat.
20. M. Kohlhase, A. Franke. MBase: Representing Knowledge and Context for the Integration of Mathematical Software Systems. *Journal of Symbolic Computation*, 23(4):365–402, 2001.
21. P. Libbrecht, E. Melis, C. Ullrich. The ActiveMath Learning Environment: System Description. In *Calculemus Workshop at IJCAR*, pages 173–177, 2001.
22. The Mizar System. http://mizar.uwb.edu.pl/system/. Developed at the University of Warsaw, directed by A. Trybulec.
23. F. Piroi. *Tools for Using Automated Provers in Mathematical Theory Exploration.* PhD Thesis. Research Institute for Symbolic Computation, Johannes Kepler University, Linz, Austria, August 2004. RISC Report 04–12.
24. F. Piroi, B. Buchberger. An Environment for Building Mathematical Knowledge Libraries. In A. Asperti, G. Bancerek, A. Trybulec, editors, *Proceedings of Mathematical Knowledge Management, Third International Conference, MKM 2004*, LNCS 3119, Springer Verlag.
25. D. Vasaru–Dupré. Automated Theorem Proving by Integrating Proving, Solving and Computing. RISC Institute, May 2000. RISC report 00–19.
26. W. Windsteiger. *A Set Theory Prover in Theorema: Implementation and Practical Applications.* PhD Thesis. Research Institute for Symbolic Computation, Johannes Kepler University, Linz, Austria, May 2001. RISC Report 01–03.
27. S. Wolfram. *The Mathematica Book*, Fifth Edition. Wolfram Media Inc., 2003.