Formalization and answers by Wolfgang Windsteiger, Bruno Buchberger, and Markus Rosenkranz (`Firstname.Lastname@risc.uni-linz.ac.at`).

1 Formalization

We assume that all variables range over the *positive real numbers*, i.e. all formulae in the knowledge base should be true statements in the domain of positive real numbers.

1.1 Statement

Theorem["sqrt[2] irrational", $\neg$ rat[$\sqrt{2}$]]

1.2 Definitions

Definition["rational", any[r],
rat[r] $:\Leftrightarrow \underset{a,b}{\exists}$ (nat[a] $\wedge$ nat[b] $\wedge$ $r = \frac{a}{b} \wedge$ coprime[a, b])]

Definition["sqrt", any[x], $\sqrt{x} := \underset{y}{\ni!}$ ($y^2 = x$)]

1.3 The Proof

1.3.1 The Proof of the Main Theorem

Definition["rational", any[r],
rat[r] $:\Leftrightarrow \underset{a,b}{\exists}$ (nat[a] $\wedge$ nat[b] $\wedge$ $r = \frac{a}{b} \wedge$ coprime[a, b])]

Definition["sqrt", any[x], $\sqrt{x} := \underset{y}{\ni!}$ ($y^2 = x$)]

Lemma["coprime", any[a, b], with[nat[a] $\wedge$ nat[b]],
($2b^2 = a^2$) $\Rightarrow \neg$ coprime[a, b]]

Theorem["sqrt[2] irrational", $\neg$ rat[$\sqrt{2}$]]

Prove[Theorem["sqrt[2] irrational"], using $\rightarrow$ ⟨Lemma["coprime"],
Definition["rational"], Definition["sqrt"]⟩,
by $\rightarrow$ ElementaryReasoner, built-in $\rightarrow$ Built-in["Rational Numbers"],
ProverOptions $\rightarrow$ {SimplifyFormula $\rightarrow$ True, RWCombine $\rightarrow$ True}]

Prove:

(Theorem (sqrt[2] irrational)) $\neg$ rat[$\sqrt{2}$],

under the assumptions:

(Lemma (coprime)) $\underset{a,b}{\forall}$ (nat[a] $\wedge$ nat[b] $\Rightarrow$ (($2b^2 = a^2$) $\Rightarrow$ $\neg$ coprime[a, b])),

(Definition (rational)) $\underset{r}{\forall}$ rat[r] :$\Leftrightarrow$ $\underset{a,b}{\exists}$ ((nat[a] $\wedge$ nat[b])$\wedge$
($r = \frac{a}{b} \wedge$ coprime[a, b])),

(Definition (sqrt)) $\underset{x}{\forall}$ $\sqrt{x} := \underset{y}{\ni!}$ $y^2 = x$.

From (Definition (sqrt)) we can infer by expansion of the "such that"-quantifier

(1) $\underset{x,y}{\forall}$ ($\sqrt{x} = y \Leftrightarrow y^2 = x$).

We prove (Proposition (sqrt2)) by contradiction.
We assume

(3) rat[$\sqrt{2}$],

and show a contradiction.
Formula (3), by (Definition (rational)), implies:

(4) $\underset{a,b}{\exists}$ (coprime[a, b] $\wedge$ nat[a] $\wedge$ nat[b] $\wedge$ $\sqrt{2} = \frac{a}{b}$).

By (4) we can take appropriate values such that:

(5) $\text{coprime}[\text{a}_0, \text{b}_0] \wedge \text{nat}[\text{a}_0] \wedge \text{nat}[\text{b}_0] \wedge \sqrt{2} = \dfrac{\text{a}_0}{\text{b}_0}$.

By modus ponens, from (5.2), (5.3) and an appropriate instance of (Lemma (coprime)) follows:

(6) ${\text{2b}_0}^2 = {\text{a}_0}^2 \Rightarrow \neg\, \text{coprime}[\text{a}_0, \text{b}_0]$,

Formula (5.4), by (1), implies:

(7) $\left(\dfrac{\text{a}_0}{\text{b}_0}\right)^2 = 2$.

Using built-in simplification rules we can simplify the knowledge base: Formula (7) simplifies to

(8) ${\text{2b}_0}^2 = {\text{a}_0}^2$.

From (8) and (6) we obtain by modus ponens

(9) $\neg\, \text{coprime}[\text{a}_0, \text{b}_0]$.

Now, (9) and (5.1) are contradictory.

1.3.2 The Proof of the Auxiliary Lemma

The auxiliary Lemma "coprime" is a statement essentially about natural numbers. In the spirit of theory exploration, see [2], we assume this lemma to be proven during an (earlier) exploration of the notion "coprime" within the universe of natural numbers. In this section, we show this phase of exploration of the natural numbers.

Lemma["coprime", any$[a, b]$, $(2b^2 = a^2) \Rightarrow \neg\, \text{coprime}[a, b]$]

Definition["even", any$[a]$, is-even$[a]$ $:\Leftrightarrow \underset{m}{\exists}\ (a = 2m)$]

Proposition["even numbers", any$[a, b]$,
$(2b = a) \Rightarrow$ is-even$[a]$ "characteristic"]
is-even$[a^2] \Rightarrow$ is-even$[a]$ "even square"

Proposition["common factor", any$[a, b]$, $\neg\, \text{coprime}[2a, 2b]$]

Prove[Lemma["coprime"], using → ⟨Proposition["even numbers"],
Proposition["common factor"], Definition["even"]⟩,
by → ElementaryReasoner, built-in → Built-in["Natural Numbers"],
ProverOptions → {SimplifyFormula → True}, SearchDepth → 40]

Prove:

(Lemma (coprime)) $\underset{a,b}{\forall}((2b^2 = a^2) \Rightarrow \neg\ \text{coprime}[a, b])$,

under the assumptions:

(Proposition (even numbers): characteristic) $\underset{a,b}{\forall}((2b = a) \Rightarrow \text{is-even}[a])$,

(Proposition (even numbers): even square) $\underset{a}{\forall}(\text{is-even}[a^2] \Rightarrow \text{is-even}[a])$,

(Proposition (common factor)) $\underset{a,b}{\forall} \neg\ \text{coprime}[2a, 2b]$,

(Definition (even)) $\underset{a}{\forall}\ \text{is-even}[a] :\Leftrightarrow \underset{m}{\exists}(a = 2m)$.

We assume

(1) $2{b_0}^2 = {a_0}^2$,

and show

(2) $\neg\ \text{coprime}[a_0, b_0]$.

We prove (2) by contradiction.
We assume

(3) $\text{coprime}[a_0, b_0]$,

and show a contradiction.
Formula (1), by (Proposition (even numbers): characteristic), implies:

$\text{is-even}[{a_0}^2]$,

which, by (Proposition (even numbers): even square), implies:

is-even[a_0],

which, by (Definition (even)), implies:

(4) $\underset{m}{\exists}\,(\mathrm{a}_0 = 2m)$.

By (4) we can take appropriate values such that:

(5) $\mathrm{a}_0 = 2\mathrm{m}_0$.

Formula (3), by (5), implies:

(6) coprime[$2\mathrm{m}_0, \mathrm{b}_0$].

Formula (1), by (5), implies:

(7) ${\mathrm{2b}_0}^2 = (2\mathrm{m}_0)^2$.

Using built-in simplification rules we can simplify the knowledge base:
Formula (7) simplifies to

(8) ${\mathrm{2m}_0}^2 = {\mathrm{b}_0}^2$.

Formula (8), by (Proposition (even numbers): characteristic), implies:

is-even[${\mathrm{b}_0}^2$],

which, by (Proposition (even numbers): even square), implies:

is-even[b_0],

which, by (Definition (even)), implies:

(9) $\underset{m}{\exists}\,(\mathrm{b}_0 = 2m)$.

By (9) we can take appropriate values such that:

(10) $\mathrm{b}_0 = 2\mathrm{m}_1$.

Formula (6), by (10), implies:

(15) coprime[$2\mathrm{m}_0, 2\mathrm{m}_1$].

Now, (15) and (Proposition (common factor)) are contradictory.

---

2 System

What is the home page of the system? `http://www.theorema.org/`

What are the books about the system? The essential theoretical ideas involved in the *Theorema* system can be found already in [5]. A comprehensive description of the basic design of a computer-system based on these ideas is then given in [1]. For a more elaborate exposition and for concrete design principles of the current *Theorema* system, we refer to the survey papers [3, 4].

Most of the system components are described in all detail in journal papers, conference proceedings articles, or technical reports. All downloadable material can be found on the *Theorema* homepage.

What is the logic of the system? The logic frame of *Theorema* is higher order predicate logic, which is extended by the language construct "sequence variables". Sequence variables are variables for which an arbitrary finite number of terms can be substituted. Sequence variables are a convenient construct for the formulation of algorithms in terms of pattern matching within logic. Thus, the *Theorema* language is also particularly suited for expressing logic algorithms like theorem provers etc. A logical study of sequence variables is given in [6].

In fact, the *Theorema* system is a (growing) collection of various general purpose and special theorem provers. The general purpose provers (like the first order predicate logic natural deduction prover) are valid only for special fragments of predicate logic (e.g. first order predicate logic). The special provers are valid only under the additional assumption that special knowledge is available that characterizes the underlying special theory. For example, the (various versions) of the induction prover assume that, for certain functions, an induction principle holds; the geometry prover based on the Gröbner bases method assumes that the universe of discourse is the field of complex numbers and the basic properties of complex numbers are available; the set theory prover assumes that the axioms and basic properties of set theory are valid. We do not yet have a prover for a general version of higher order logic. Of course, this approach to building a theorem proving system supposes that the correctness of special theorem provers is (automatically) proved w.r.t. to more general provers. So far, this research program is only partially carried

out. For example, Gröbner bases theory is proved correct in a fairly formal way. A complete formal proof within *Theorema* is planned for the near future.

What is the implementation architecture of the system? All *Theorema* 'reasoners' (provers, solvers, and simplifiers) are written in the programming language of Mathematica. *Theorema* does *not* use the Mathematica algorithm library or any implicit mathematical knowledge presupposed in Mathematica algorithms. The *Theorema* language is a version of higher order predicate logic. The currently available reasoners comprise *general purpose reasoners* (several methods for proving in first order predicate logic, a prover for equational reasoning, or a general simplification prover based on equality rewriting) and special purpose reasoners (like Collins' decision procedure for the theory of real closed fields, a special prover for geometry based on algebraic techniques like Gröbner bases, or the simplification prover for number domains used in the proofs shown in Section 1) that are valid only in particular theories. Various external provers like e.g. Otter can be accessed through *Theorema* by translating *Theorema* formulae to the specific input format for these provers and getting the results back. Reasoners available in Mathematica can also be accessed by *Theorema* upon explicit request by the user. The *Theorema* language includes a programming language as a natural sub-language.

Theorema provers have a modular structure, i.e. every *Theorema* prover is composed from smaller units, so-called 'special prover modules'. These prover modules are separate units, and can therefore be combined in arbitrary way. In the current status, the access to special prover modules is restricted to the system developers, but a mechanism for users to compose their own provers from available special prover modules is planned for future versions of the system. *Theorema* uses a general proof search procedure that maintains a global proof object. The proof object has a tree-structure where each node in the tree represents a *proof situation* made up basically from the proof goal and the current assumptions. In each step, the proof search procedure tries to apply a special prover module in order to simplify the current proof situation. A special prover module can be applied to a proof situation if one of its inference rules can be applied to the proof situation. Inference rules in the prover modules are implemented as Mathematica programs that take a proof situation as parameter and return a new proof situation. Applicability of an inference rule is tested by pattern matching on the parameters of the

Mathematica program against the current proof situation. When applying a special prover module to a proof situation the proof search procedure inserts the result of the first applicable rule as a new node into the global proof object. The proof search continues until a trivial proof situation (e.g. the goal is identical to one of the assumptions) appears on one branch of the tree, in which case the proof succeeds, or until the maximal search depth is exceeded on all branches, in which case the proof attempt fails.

What does working with the system look like? The current version of *Theorema* is implemented as an extension package to Mathematica, thus, the standard way of working with *Theorema* is an interactive user-system-dialog in the well-known Mathematica notebook FrontEnd. The Mathematica notebook FrontEnd supports configurable two-dimensional mathematical notation both in input and output. The examples in Section 1 demonstrate a typical *Theorema* session. Two categories of commands are available for the user:

Organization of Knowledge: The *Theorema* Formal Text Language allows the user to enter arbitrary definitions, axioms, propositions, algorithms, etc. to the system and combine such formulae into theories in a nested way so that hierarchies of mathematical knowledge bases (theories) can be built up. All formulae may receive key words and labels for easy reference. Labels, however, carry no logical meaning. Declaration of free variables and conditions on these variables are specified using the keywords 'any' and 'with'. The individual formulae can be entered in a two-dimensional syntax very close to how formulae are written in mathematical textbooks. The actual input of arbitrary two-dimensional notation and special mathematical symbols is supported by the standard Mathematica notebook FrontEnd through input palettes and keyboard shortcuts. The input of *Theorema*-specific notation, like e.g. a formal text entity containing labelled formulae as used in Proposition "even numbers", is taken care of by additional input palettes. The definitions, theorems, lemmata, and propositions as they display in Section 1 illustrate some of the features of the Formal Text Language.

Note that the *Theorema* language also supports several notational variants for mathematical syntax in order to accommodate to the preferences of the user. As an example, we used the standard form of the

existential quantifier

$$\underset{a,b}{\exists}\,(\mathrm{nat}[a] \wedge \mathrm{nat}[b] \wedge r = \frac{a}{b} \wedge \mathrm{coprime}[a,b])$$

in Definition "rational" as given in Section 1.2. Supported notational variants, which would all be interpreted by all *Theorema* provers in exactly the same way, are:

$$\underset{\mathrm{nat}[a],\mathrm{nat}[b]}{\exists}\,(r = \frac{a}{b} \wedge \mathrm{coprime}[a,b]),$$

$$\underset{\mathrm{nat}[a,b]}{\exists}\,(r = \frac{a}{b} \wedge \mathrm{coprime}[a,b]),\ \text{or}$$

$$\underset{\substack{a,b \\ \mathrm{nat}[a]\wedge\mathrm{nat}[b]}}{\exists}\,(r = \frac{a}{b} \wedge \mathrm{coprime}[a,b]).$$

Mathematical Activities: Mathematical knowledge can then be processed in various ways using the *Theorema* User Language. Currently, the User Language supports 'proving', 'solving', and 'simplifying', see the commands `Prove[...]` in Section 1 for typical examples. These examples also exhibit, how knowledge specified in the Formal Text Language can be referred to in the User Language. Note also, that the keywords such as 'any' or 'with' are processed when formulae are passed to a prover: Compare the formulae as they are echoed at the beginning of the proof to how they appear in the Formal Text Language.

Proofs (or traces of solving or simplifying) are generated completely automatically and display very much in the form as they would be written in mathematical textbooks including intermediate explanatory text in natural language. Alternatively, *Theorema* also offers an interactive mode, where user-interaction *during proof generation* is supported. Mathematical formulae are displayed in two-dimensional syntax. The proofs appear in the *Theorema* system almost exactly as they are typeset in Section 1.

However, some of the features of the *Theorema* user interface cannot be modelled in the style of this paper:

- Proof branches are organized in hierarchically nested cells, which can be opened or closed by double-clicking the cell bracket on the right margin of the window. This allows the user to hide (or display) certain parts of a proof easily by mouse-click.
- Formula labels are active elements, such that clicking a formula reference in the running text shows the full formula in a separate pop-up window.
- Goal formulae, assumptions, labels, and explanatory text use different colors.

Also, there are means in *Theorema* to design and implement one's own syntax including the design of new mathematical symbols of arbitrary complexity ('logicographic symbols').

What is special about the system compared to other systems?

- *Theorema* is both a logic language and a programming language. This means that, for example, within the same language and system, a formula that describes an algorithm can be proved correct and can then be executed on concrete input.
- The three fundamental mathematical activities proving, solving, and simplifying can be done in one uniform language and logic frame.
- *Theorema* is a multi-method system: instead of using one uniform proof method for all of mathematics *Theorema* provides sophisticated special provers for certain mathematical theories. These special provers are partly based on powerful computer algebra methods, which were the research focus of the working group in earlier years.
- *Theorema* has an attractive two-dimensional, extensible syntax.
- Most of the *Theorema* provers generate proofs in a natural, human-readable style with intermediate explanatory text and various tools that help to get various (contracted and expanded) views of proofs.
- *Theorema* has various structuring mechanisms for large mathematical knowledge bases, notably the recursive 'Theory' construct and a functor construct, which is similar to but more general than the functor construct of SML.

What are other versions of the system? There are no other versions of the system. The current version is free for download from the *Theorema* webpage `http://www.theorema.org/` under "software".

Who are the people behind the system? The development of *Theorema* has been initiated by Bruno Buchberger, who also implemented first prototypes in the mid 1990's and directs the project since then in cooperation with Tudor Jebelean and Wolfgang Windsteiger. Current senior *Theorema* researchers are, in addition, Temur Kutsia, Florina Piroi, and Markus Rosenkranz. Former *Theorema* PhD students, who contributed to the system, are Daniela Vasaru-Dupre, Mircea Marin, Koji Nakagawa, Judit Robu, and Elena Tomuta. For a complete (and always up-to-date) listing of persons involved in the *Theorema* project we refer to the *Theorema* webpage.

What are the main user communities of the system? There is a small community of alpha testers, mainly math researchers and math teachers. A major didactic case study in undergraduate math education at the Johannes Kepler University of Linz and the Polytechnic University Hagenberg is under way.

What large mathematical formalizations have been done in the system? Elementary analysis (with the typical epsilon/delta proofs), equivalence relations and partitions based on set theory, polynomial interpolation, the theory of lists with verified list algorithms like sorting, and the automated synthesis of the Gröbner bases algorithm.

What representation of the formalization has been put in this paper? The formalization in Section 1 should be self-explanatory. In fact, it is a main design principle of *Theorema* that formalizations using the *Theorema* language constructs should be self-explanatory and easy to read. The theorem is formulated as a statement over the positive real numbers. The first part shown in Section 1.3.1 contains the main proof that reduces the problem over the positive reals to a lemma over the natural numbers. Section 1.3.2 shows the second part that contains the proof of the auxiliary lemma over the naturals.

What needs to be explained about this specific proof? The proof of the theorem as shown in Section 1.3 is generated completely automatically within *Theorema*. The only user-interactions required are

- to choose an appropriate prover from the *Theorema* prover library (in the example the "Elementary Reasoner"),
- to specify appropriate prover options for the chosen prover, and
- to provide the auxiliary Lemma "coprime" necessary in the proof.

In the spirit of the layered approach of *Theorema*, the auxiliary lemma can then be proved in a separate proving session as shown in Section 1.3.2. Again, auxiliary knowledge needed in this proof (in the example the Propositions "even numbers" and "common factor") can be proved in separate phases of exploring a theory. However, we do not present the proofs of these propositions here, because when investigating the irrationality of $\sqrt{2}$ one would usually consider these propositions to be *known properties of natural numbers*. Phases of theory exploration can be structured bottom-up or top-down just like human mathematicians build up hierarchically structured mathematical knowledge.

Typically, different phases of theory exploration are characterized by using different proving techniques. In the case of *automated theory exploration* using *Theorema* this means that switching from one exploration phase to another would be reflected in changing the prover that is used in the Prove-calls. In this example, in fact, we use the Elementary Reasoner in both phases, but we allow the prover to access different portions of built-in computational knowledge in either phase.

The special feature of the Elementary Reasoner used for generating the proofs shown in Section 1 is the smooth integration of 'proving' and 'simplifying' ('computing') within one system. 'Simplifying', here, means 'simplifying expressions (to canonical form) based on the *algorithmic semantics* of the language'. The algorithmic semantics of the *Theorema* language consists of computation rules for the *algorithmic part of the language*[1], i.e. finite sets, finite tuples, quantifiers with a finite range, and basic arithmetic on numbers. Sets, tuples, and quantifiers are represented as special data structures

[1]In contrast, the semantics of the *non-algorithmic part of the language* is coded into inference rules that make up the *Theorema* special prover modules.

and the operations on these entities are implemented in the *Theorema* language semantics using the programming language of Mathematica. Only for arithmetic on natural numbers, integers, and rational numbers the *Theorema* semantics may access the arithmetic rules from the underlying Mathematica system.

In the example, the option `built-in→Built-in["Rational Numbers"]` allows the Elementary Reasoner explicitly to use built-in computation rules for operations on rational numbers. Arithmetic on numbers is, in fact, the only case, where *Theorema* silently relies on the mathematical algorithm library available from Mathematica[2]. On explicit user request, however, the interface allows this prover to even access special simplification algorithms from Mathematica when performing computational simplification. Specifying the prover option `SimplifyFormula→True` (default value is `False`) tells the prover to post-process any formula obtained from a computation by Mathematica's `FullSimplify` function. `FullSimplify` is a black-box simplifier for Mathematica expressions, which uses powerful simplification rules, in particular for arithmetic expressions. Moreover, the prover performs additional simplification of equalities involving arithmetic expressions, such that certain equalities are turned into equalities that are more likely to be usable for rewriting. In our context, the correctness of the built-in simplifier is based exclusively on the field axioms (for proofs over the reals) and the ring axioms (for proofs over the naturals[3]), respectively. In other words, the 'hidden knowledge' used by the simplifier are only the field or ring axioms[3], respectively. All other knowledge, e.g. Lemma "coprime", Proposition "even numbers", and Proposition "common factor", must be mentioned explicitly in the call of the prover.

A combination of these capabilities is used in the main proof when simplifying formula

$$(7) \quad \left(\frac{a_0}{b_0}\right)^2 = 2$$

to

$$(8) \quad 2{b_0}^2 = {a_0}^2$$

[2]Apart from that, Mathematica is used just as a programming environment!

[3]To be precise: the simplifier for the naturals uses the ring axioms without the axiom ensuring the existence of additive inverses.

and in the proof of the auxiliary lemma when simplifying

(7) $2{b_0}^2 = (2m_0)^2$

to

(8) $2{m_0}^2 = {b_0}^2$.

Definition "sqrt" uses a convenient language construct available in *Theorema*: The 'the-unique'-quantifier $\underset{y}{\ni!}\ P_y$ denotes 'the unique y satisfying P_y'. The expression $f[x] := \underset{y}{\ni!}\ P_{x,y}$ is a means to express an implicit definition for the function f, for which, of course, we need to verify $\underset{x}{\forall}\ \underset{y}{\exists!}\ P_{x,y}$ beforehand. In the concrete example of Definition "sqrt", we assume that $\underset{x}{\forall}\ \underset{y}{\exists!}\ y^2 = x$ holds over the positive real numbers. Based on this convention the prover applies an inference rule for the 'the-unique'-quantifier in order to rewrite Definition "sqrt" in the main proof into

(2) $\underset{x,y}{\forall}\ (\sqrt{x} = y \Leftrightarrow y^2 = x)$.

All other steps in the proofs are basic predicate logic and therefore require no further explanation.

References

[1] B. Buchberger. Symbolic Computation: Computer Algebra and Logic. In K. Schulz, editor, *FroCoS: Frontiers of Combined Systems*, 1996.

[2] B. Buchberger. Algorithm Supported Mathematical Theory Exploration. In B. Buchberger and John Campbell, editors, *Proceedings of AISC 2004 (7th International Conference on Artificial Intelligence and Symbolic Computation*, volume 3249 of *Springer Lecture Notes in Artificial Intelligence*, RISC, Johannes Kepler University, Austria, September 22-24 2004. Springer Berlin Heidelberg. ISSN 0302-9743, ISBN 3-540-23212-5.

[3] 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.

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

[5] B. Buchberger and F. Lichtenberger. *Mathematik für Informatiker I.* Springer Verlag, 2nd edition, 1981.

[6] T. Kutsia and B. Buchberger. Predicate Logic with Sequence Variables and Sequence Function Symbols. In Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec, editors, *Proceedings of the 3rd International Conference on Mathematical Knowledge Management, MKM'04*, volume 3119 of *Lecture Notes in Computer Science*, pages 205–219, Bialowieza, Poland, Sep 19–21 2004. Springer Verlag.