Mathematical Knowledge Management in Theorema

Bruno Buchberger

Research Institute for Symbolic Computation
Johannes Kepler University, A4040 Linz, Austria
buchberger@risc.uni-linz.ac.at

Abstract: We describe tools available and planned in the *Theorema* system for supporting the key issues of mathematical knowledge management: mathematical knowledge retrieval, mathematical knowledge composition, and the formal training of the users and creators of mathematical knowledge.

■ Introduction

In the call for papers for the Workshop on Mathematical Knowledge Management 2001, I tried to sketch the scope of "mathematical knowledge management". First, I proposed to parse the word as (mathematical knowledge) management, i.e. the (computer-supported) management of mathematical knowledge, and not as mathematical (knowledgement management), i.e. mathematical approaches to the general problem of knowledge management. Of course, mathematical approaches to the management of *mathematical* knowledge *are* in the scope of the workshop and so are mathematical approaches to the management of general knowledge if they have a significant bearing on the special case of managing mathematical knowledge. However, what we should exclude is (mathematical) approaches to the management of general knowledge that do not have a particular significance for the case of managing mathematical knowledge. Management of mathematical knowledge is challenging and, most probably, the most challenging case of knowledge management because mathematical knowledge has the highest degree of structural complexity. As a consequence, I believe that if, in the next few years, we will be able to make significant progress in developing techniques for the management of mathematical knowledge, these methods will be highly significant and applicable for the management of knowledge in all other fields because many of the other fieleds tend to be comparatively "flat", i.e. less structured.

Second, I tried to identify three main problems in the area of mathematical knowledge management:

- the problem of *retrieving* mathematical knowledge from our current and future mathematical knowledge space,
- the problem of *building up* future mathematical knowledge spaces, and
- the problem of *educating* a new generation of mathematicians who will be able to work more professionally with existing mathematical knowledge spaces and to create mathematical knowledge spaces of higher quality.

Each of the three problems has many subproblems some of which are listed in the call for papers of the workshop.

It is clear that, globally, our current status of mathematical knowledge management is unsatisfactory, see for example the QED Manifesto [QED95], which "deliberately avoids any authorship" because the ideas presented in this manifesto reflect the joint / disjoint opinions of quite some contemporary mathematicians. The way out is of course that we combine our efforts on various different levels and from quite different perspectives to improve the situation practically and theoretically.

In the present paper, I report on some of the ideas behind the *Theorema* project that, in my opinion, have bearing on the future of mathematical knowledge management. I structure this paper according to the three problems listed above. *Theorema* is a software system, programmed in Mathematica but not using any of the implicit knowledge of the Mathematica algorithm library, that aims at adding theorem-proving power to the current mathematical software systems, whose main emphasis is on providing algorithm libraries. In fact, *Theorema* is a philosophy *and* a system. The philosophy is the author's philosophy of doing mathematics as an interaction betwen intuitive exploration *and* formal verification. I developed this view over the years. It is reflected, for example, in my "Thinking, Speaking, Writing" course for PhD students [Buchberger 1992]. Hence, from the outset, *Theorema* was meant not only as an extension of the algorithm library of current mathematical systems but mainly as a tool for managing the process of doing mathematics, which also can be seen as the activity of "managing" mathematical knowledge.

■ Mathematical Knowledge Retrieval

■ The Role of Automated Theorem Proving

The problem of mathematical knowledge retrieval is the problem of finding information on a given mathematical notion, properties of a notion, solution of a mathematical problem, etc. in a given space of mathematical knowledge. The specific difficulty of this problem is that, in mathematics, textual search is not sufficient because the essence of mathematical information is contained in the logical structure and not so much in its linguistic appearance. For example, the two formulae

$$\mathrm{GB}[\mathrm{F}] \Longleftrightarrow \underset{\mathrm{f} \in \mathrm{I}[\mathrm{F}]}{\forall} \mathrm{R}[\mathrm{F}, \mathrm{f}]$$

and

$$\mathrm{GB}[\mathrm{F}] \Longleftrightarrow \underset{\mathrm{g} \in \mathrm{I}[\mathrm{F}]}{\forall} \mathrm{R}[\mathrm{F}, \mathrm{g}],$$

although textually different, from a mathematical point of view, are basically identical. Of course, textual information in the context of mathematical knowledge retrieval is *also* important (for example, it is surely useful to know that the above formula may occur in the context of Groebner bases theory and, thus, "Groebner bases" may serve a key word in the search) but it is not sufficient. It is clear that the two formuale in the example can easily be derived from each other by simple rules of logic. More generally, ease of logical derivability is a natural measure for "vicinity" of information in mathematical knowledge bases. Hence, the natural answer to the difficulty that textual vicinity is not an essential criterion in mathematical knowledge retrieval is that, essentially, information retrieval in mathematics is the question of whether or not a given formula is a *logical consequences* of mathematical facts stored in the given knowledge space. Hence, computer-support of deciding whether or not a formula F is a logical consequence of mathematical knowledge K or computer-support of generating all or some logical consequences F of a mathematical knowledge K is the main approach for improving the efficiency and reliability of mathematical knowledge retrieval. In other words, *computer-supported theorem proving* at various levels of sophistication will be the main future instrument for making progress in mathematical knowledge retrieval. I guess that this will lead to a renaissance of computer-supported theorem proving: Computer-supported theorem proving not as a dream about finding and proving nontrivial *new* mathematical facts but as a very practical and highly sophisticated tool for retrieving *known* mathematical facts in various disguise. In the next subsections, we describe the possible concrete contributions of *Theorema* to the mathematical knowledge *retrieval* problem by computer-supported proving.

■ The *Theorema* Language

First of all, *Theorema* provides a universal formal language (essentially, a version of higher-order predicate logic) in which all of mathematics can be expressed. The language consists of an internal standard language, which is basically a representation of predicate logic formulae as nested Mathematica expressions, and an external representation which we think comes very close to the "usual" two-dimensional notation of mathematical formulae in mathematical textbooks and publications. Of course, something like *the* usual notation of mathematics does not really exist. Thus, some special decisions for the external representations had to be taken in *Theorema*. However, there exist translators between the *Theorema* language and various other formal mathematical languages as, for example, the Mizar language (see, for example, [Rudnicki 1992]) or such translators are under construction.

Here are a couple of formulae in the *Theorema* language:

$$\textbf{Definition}\Big[\text{"limit:"}, \text{any}[f], \text{with}[\, f : \mathbb{N} \to \mathbb{R}\,], \text{any}[a \in \mathbb{R}],$$
$$\text{limit}[f, a] \Longleftrightarrow \underset{\substack{\epsilon\in\mathbb{R}\\ \epsilon>0}}{\forall}\ \underset{N\in\mathbb{N}}{\exists}\ \underset{\substack{n\in\mathbb{N}\\ n\geq N}}{\forall} |f[\![x]\!] - a| < \epsilon\Big]$$

$$\textbf{Proposition}[\text{"limit of sum"}, \text{any}[\, f, g : \mathbb{N} \to \mathbb{R}\,], \text{any}[a, b \in \mathbb{R}],$$
$$(\text{limit}[f, a] \wedge \text{limit}[g, b]) \quad \Rightarrow \quad \text{limit}[f + g, a + b]]$$

$$\textbf{Definition}[\text{"+:"}, \text{any}[\, f, g : \mathbb{R} \to \mathbb{R}\,], \text{any}[x \in \mathbb{R}],$$
$$(f + g)[\![x]\!] = f[\![x]\!] + g[\![x]\!]]$$

$$\textbf{Lemma}[\text{"|+|"}, \text{any}[x, y, a, b, \delta, \epsilon \in \mathbb{R}],$$
$$(|(x + y) - (a + b)| < (\delta + \epsilon)) \quad \Longleftarrow \quad (|x - a| < \delta \wedge |y - b| < \epsilon)]$$

$$\textbf{Lemma}[\text{"max"}, \text{any}[m, M1, M2 \in \mathbb{R}],$$
$$m \geq \max[M1, M2] \quad \Rightarrow \quad (m \geq M1 \wedge m \geq M2)]$$

The actual *Theorema* expressions, as parts of higher-order predicate logic, like

$$\text{limit}[f, a] \Longleftrightarrow \underset{\substack{\epsilon\in\mathbb{R}\\ \epsilon>0}}{\forall}\ \underset{N\in\mathbb{N}}{\exists}\ \underset{\substack{n\in\mathbb{N}\\ n\geq N}}{\forall} |f[\![n]\!] - a| < \epsilon$$

can be wrapped in key-words like 'Definition', 'Proposition' etc. and we can also provide labels (strings in quotation marks) like "limit", "+:" etc. Thus, for retrieving formulae in knowledge bases by textual search, we can either use the textual information contained in the labels or, the names of the predicate and function constants. After (installing and) calling *Theorema* by entering

```
Needs["Theorema`"];
```

into a Mathematica input cell, *Theorema* definitions, propositions,etc.like the ones above can be entered into Mathematica input cells. After this, these definitions can be referenced by their labels.

The 'any' construct is used for declaring free variables. All identifiers and symbols that are neither explicitly declared as free variables nor explicitly quantified by quantifiers are considered to be constants. For example, in the above formulae, '+', '-', 'limit' etc. are constants.

Theorema quantifiers have two subscript boxes. The first one is for range descriptions like '$x \in \mathbb{R}$', the second one is for conditions like '$\epsilon>0$'. Logically, range descriptions and conditions are not really different. However, in the case formulae are used in computations, the distinction between range descriptions and conditions is very useful: In the algorithmic interpretation of a quantified formula like

$$\underset{\substack{a\in A \\ C[a]}}{\forall} F[a]$$

the range description can be used for enumerating, successively, all values in the range whereas the condition can then be used to single out those values for which the formula *F* should be evaluated. The distinction between range descriptions and conditions is also useful in the manipulation of typed variables, see below. The distinction can also be expressed for free variables: any[$a\in\mathbb{R}$]' is sa range description for the free variable 'a', 'with[$f : \mathbb{N} \to \mathbb{R}$]' expresses a condition on the free variable 'f'. Sometimes, the distinction depends on the intention of the user. For example, in certain situations it may be more desirable to conceive '$f : \mathbb{N} \to \mathbb{R}$' as a range description whereas in other situations the same formula may well be considered as a condition.

The notation 'f⟦x⟧' used in the above formulae denotes the application of the function *f* to the argument *x* in the sense of (Zermelo-Fraenkel) set theory, whereas 'f[x]' is function application in the sense of predicate logic.

The external notation of *Theorema* is extensible, i.e. the user can define arbitrary new two-dimensional symbols with slots for arguments of arbitrary size, which can be used as function and predicate constants. This feature of *Theorema*, which we call "logicographic symbols", is explained in detail in the companion paper [Buchberger, Nakagawa 2001]. Also, in *Theorema,* we have tools for generating variants of notions in a systematic way automatically, see also [Buchberger, Nakagawa 2001].

Definitions, propositions, etc. may also consist of several lines that can be labeled additionally. For example, here is a formulation of the merge-sort algorithm in *Theorema* conssting of three labeled formulae lines:

$$\textbf{Algorithm}\left[\text{"merge sort"}, \text{any}[X, Y, a, b, \bar{x}, \bar{y}], \begin{array}{ll} \text{mg}[\langle\rangle, Y] := Y & \text{"eg"} \\ \text{mg}[X, \langle\rangle] := X & \text{"ge"} \\ \text{mg}[\langle a, \bar{x}\rangle, \langle b, \bar{y}\rangle] := \begin{cases} a \smile \text{mg}[\langle\bar{x}\rangle, \langle b, \bar{y}\rangle] & \Leftarrow \quad a \geq b \\ b \smile \text{mg}[\langle a, \bar{x}\rangle, \langle\bar{y}\rangle] & \Leftarrow \quad \text{otherwise} \end{cases} & \text{"gg"} \end{array}\right]$$

In the third formula, we us sequence variables like '$\bar{x}$', etc, for which arbitrary finite sequences of terms can be substituted.

■ Building Up Structured Knowledge Bases in Theorema

For building up structured knowledge bases, *Theorema* provides the 'Theory' construct. For example, we could build up a knowledge base

Theory["absolute value",
  Lemma["|+|"]
  Lemma["|−|"]
  Lemma["|*|"]
  Lemma["|/|"]]

and a knowledge base

Theory["sequences",
  Definition["+"]
  Definition["−"]
  Definition["*"]
  Definition["/"]]

and then a knowledge base

Theory["example",
  Theory["absolute value"]
  Theory["sequences"]].

By the Theory construct we can compose arbitrarily deeply nested knowledge bases and refer to them by just one label and also select specific parts of knowledge bases. Typically, such knowledge bases are used as one of the essential parameter to the *Theorema* provers, solvers, and simplifiers.

■ Structuring by Functors

The Theory-construct is an "external" tool for structuring knowledge. In *Theorema*, we also provide the functor construct, which can be considered to be a predicate logic quantifier and, hence, is a logic-internal means for structuring mathematical knowledge. The *Theorema* functor construct is similar to the functor construct of ML but slightly more general. and allows to define processes by which new domains (carriers together with functions and predicates) are produced from arbitrary given domains. As an example, we show the functor 'pol' that takes a domain C of coefficients and a domain T of terms (power products) and constructs the domain P of polynomials over C and T represented as tuples of pairs of coefficients and power products ordered by the ordering available on the power products.

$$\textbf{Definition}\Big[\text{"pol"}, \text{any}[C, T],$$

$$\text{pol}[C, T] = \text{Functor}\Big[P, \text{any}[c, d, i, \overline{m}, \overline{n}, p, q, s, t],$$

$$\underline{\mathbb{s} = \langle 0 : P, 1 : P, + : P\times P \to P, + : P \to P, - : P\times P \to P, * : P\times P \to P\rangle}$$

$$\underset{P}{\in}[p] \Longleftrightarrow \left(\text{is-tuple}[p] \bigwedge \underset{i=1,\ldots,|p|}{\forall} \left(\text{is-pair}[p_i] \bigwedge \underset{C}{\in}[p_{i,1}] \bigwedge \underset{T}{\in}[p_{i,2}]\right)\right)$$

$$\underset{P}{0} = \langle\rangle$$

$$\underset{P}{1} = \left\langle\left\langle \underset{C}{1}, \underset{T}{1}\right\rangle\right\rangle$$

$$\langle\rangle \underset{P}{+} q = q$$

$$p \underset{P}{+} \langle\rangle = p$$

$$\langle\langle c, s\rangle, \overline{m}\rangle \underset{P}{+} \langle\langle d, t\rangle, \overline{n}\rangle = \Big\| \langle c, s\rangle \smile \left(\langle\overline{m}\rangle \underset{P}{+} \langle\langle d, t\rangle, \overline{n}\rangle\right) \Leftarrow s \underset{T}{>} t,$$

$$\langle d, t\rangle \smile \left(\langle\langle c, s\rangle, \overline{m}\rangle \underset{P}{+} \langle\overline{n}\rangle\right) \Leftarrow t \underset{T}{>} s,$$

$$\left(\left\langle c \underset{C}{+} d, s\right\rangle\right) \smile \left(\langle\overline{m}\rangle \underset{P}{+} \langle\overline{n}\rangle\right) \Leftarrow \left(c \neq \underset{C}{-} d\right),$$

$$\left(\langle\overline{m}\rangle \underset{P}{+} \langle\overline{n}\rangle\right) \Leftarrow \text{otherwise} \Big\|$$

$$\underset{P}{-}\langle\rangle = \langle\rangle$$

$$\underset{P}{-}\langle\langle c, s\rangle, \overline{m}\rangle = \left\langle \underset{C}{-} c, s\right\rangle \smile \left(\underset{P}{-}\langle\overline{m}\rangle\right)$$

$$p \underset{P}{-} q = p \underset{P}{+} \left(\underset{P}{-} q\right)$$

$$\langle\rangle \underset{P}{*} q = \langle\rangle$$

$$p \underset{P}{*} \langle\rangle = \langle\rangle$$

$$\langle\langle c, s\rangle, \overline{m}\rangle \underset{P}{*} \langle\langle d, t\rangle, \overline{n}\rangle = \left(\left\langle\left\langle c \underset{C}{*} d, s \underset{T}{*} t\right\rangle\right\rangle \underset{P}{+} \langle\langle c, s\rangle\rangle \underset{P}{*} \langle\overline{n}\rangle\right) \underset{P}{+} \langle\overline{m}\rangle \underset{P}{*} \langle\langle d, t\rangle, \overline{n}\rangle$$

$$\Big]\Big]$$

Note that, in addition to the ring operations (listed in the signature $\mathbb{s}$ of *P*), the functor also defines a unary predicate $\underset{P}{\in}$ that defines which objects belong to the polynomial domain generated by the functor.

Having introduced this functor, one can now apply it to particular coefficient and power product domains and produce, by one functor call, the corresponding polynomial domain in tuple representation. Let us assume, for example, that the two domains $\mathbb{Z}$ (integers) and $\mathbb{T}$ (trivariate power products as triples of natural numbers encoding the exponents at the three indeterminates) have already been defined (using appropriate parameter-less functors) in appropriate definitions Definition["integers"] and Definition["power products"]. Building up the corresponding polynomial domain $\mathbb{P}$ and computing in $\mathbb{P}$ would then proceed as follows:

Theory["F",
  Definition["integers"]
  Definition["power products"]
  Definition["pol"]]

Definition["D",
  $\mathbb{P} = \mathrm{pol}[\mathbb{Z}, \mathbb{T}]$]

Use[⟨Built–in["Quantifiers"], Built–in["Connectives"], Built–in["Numbers"], Built–in["Tuples"], Built–in["Sets"], Built–in["natural numbers"], Theory["F"], Definition["D"]⟩]

Note that, in addition to user–defined knowledge like Theory["F"] and Definition["D"], in *Theorema* one can also explicitly make available built-in computational knowledge (algorithms) from the underlying Mathematica and *Theorema* library by using the 'Built–in' construct. Now, for example, the call

$$\mathrm{Compute}\Big[\Big(\langle\langle 5, \langle 2, 3, 1\rangle\rangle, \langle 3, \langle 1, 6, 3\rangle\rangle\rangle \underset{\mathbb{P}}{*} \Big(\langle\langle 5, \langle 2, 3, 1\rangle\rangle, \langle 3, \langle 1, 6, 3\rangle\rangle\rangle \underset{\mathbb{P}}{*} \langle\langle 5, \langle 2, 3, 1\rangle\rangle, \langle 3, \langle 1, 6, 3\rangle\rangle\rangle\Big)\Big)\Big]$$

produces

$$\langle\langle 125, \langle 6, 9, 3\rangle\rangle, \langle 225, \langle 5, 12, 5\rangle\rangle, \langle 135, \langle 4, 15, 7\rangle\rangle, \langle 27, \langle 3, 18, 9\rangle\rangle\rangle$$

which, in the usual representation of trivariate polyonomials, is the result

$$125\,x1^6\,x2^9\,x3^3 + 225\,x1^5\,x2^{12}\,x3^5 + 135\,x1^4\,x2^{15}\,x3^7 + 27\,x1^3\,x2^{18}\,x3^9$$

of expanding

$$(5\,x1^2\,x2^3\,x3 + 3\,x1\,x2^6\,x3^3)^3.$$

■ Provers in Theorema

The core of the *Theorema* system is a growing library of povers (solvers, and simplifiers). We intend to implement, in the uniform language frame of *Theorema*, all major general and special theorem provers known in the literature plus a couple of new provers that have a particular emphasis on generating proofs that are "natural" (i.e. in the style of proofs generated by human mathematicians) and easy to read. Also, we developed and are developing a couple of simple formulae translators by which it is possible to call various existing and well-known theorem provers, as for example Otter, from within *Theorema*, i.e. for formulae and knowledge bases in the *Theorema* syntax.

In the frame of the theorem-proving based approach to mathematical knowledge retrieval, "retrieving" the proposition that the limit of sums is the sum of the limits, in *Theorema,* one has to formulate the proposition to be retrieved, for example, in the form of the Proposition["limit of sum"] above, and to try to prove the proposition from the available knowledge, for example the following theory

Theory["limit",
  Definition["limit:"]
  Definition["+:"]
  Lemma["|+|"]
  Lemma["max"]]

using some of the provers implemented in *Theorema,* for example the so called PCS prover, see [Buchberger et al. 2000]. This is a "special" prover in the sense that it is not valid for all of predicate logic but is valid w.r.t. the first-order theory of real numbers. (In fact, in the following call of the prover, we use slightly simpler versions of the formulae in the knowledge omitting some of the type information.) The corresponding *Theorema* prover call has the following form:

Prove[Proposition["limit of sum"], using → Theory["limit"], by → PCS]

After a couple of seconds, the following proof text (including the intermediate English explanatory text) will be produced completely automatically by the PCS prover:

Prove:

(Proposition (limit of sum)) $\underset{f,a,g,b}{\forall}(\mathrm{limit}[f, a] \wedge \mathrm{limit}[g, b] \Rightarrow \mathrm{limit}[f + g, a + b])$,

under the assumptions:

(Definition (limit:)) $\underset{f,a}{\forall}\left(\mathrm{limit}[f, a] \Leftrightarrow \underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n\geq N}}{\forall}(|f[n] - a| < \epsilon)\right)$,

(Definition (+:)) $\underset{f,g,x}{\forall}((f + g)[x] = f[x] + g[x])$,

(Lemma (|+|)) $\underset{x,y,a,b,\delta,\epsilon}{\forall}(|(x + y) - (a + b)| < \delta + \epsilon \Leftarrow (|x - a| < \delta \wedge |y - b| < \epsilon))$,

(Lemma (max)) $\underset{m,M1,M2}{\forall}(m \geq \max[M1, M2] \Rightarrow m \geq M1 \wedge m \geq M2)$.

We assume

(1) $\mathrm{limit}[\mathrm{f}_0, \mathrm{a}_0] \wedge \mathrm{limit}[\mathrm{g}_0, \mathrm{b}_0]$,

and show

(2) $\mathrm{limit}[\mathrm{f}_0 + \mathrm{g}_0, \mathrm{a}_0 + \mathrm{b}_0]$.

Formula (1.1), by (Definition (limit:)), implies:

(3) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n\geq N}}{\forall}(|\mathrm{f}_0[n] - \mathrm{a}_0| < \epsilon)$.

By (3), we can take an appropriate Skolem function such that

(4) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{\substack{n \\ n\geq \mathrm{N}_0[\epsilon]}}{\forall}(|\mathrm{f}_0[n] - \mathrm{a}_0| < \epsilon)$,

Formula (1.2), by (Definition (limit:)), implies:

(5) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n\geq N}}{\forall} (|g_0[n] - b_0| < \epsilon)$.

By (5), we can take an appropriate Skolem function such that

(6) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{\substack{n \\ n\geq N_1[\epsilon]}}{\forall} (|g_0[n] - b_0| < \epsilon)$,

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

(7) $\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{N}{\exists}\ \underset{\substack{n \\ n\geq N}}{\forall} (|(f_0 + g_0)[n] - (a_0 + b_0)| < \epsilon)$.

We assume

(8) $\epsilon_0 > 0$,

and show

(9) $\underset{N}{\exists}\ \underset{\substack{n \\ n\geq N}}{\forall} (|(f_0 + g_0)[n] - (a_0 + b_0)| < \epsilon_0)$.

We have to find N_2^* such that

(10) $\underset{n}{\forall} (n \geq N_2^* \Rightarrow |(f_0 + g_0)[n] - (a_0 + b_0)| < \epsilon_0)$.

Formula (10), using (Definition (+:)), is implied by:

(11) $\underset{n}{\forall} (n \geq N_2^* \Rightarrow |(f_0[n] + g_0[n]) - (a_0 + b_0)| < \epsilon_0)$.

Formula (11), using (Lemma (|+|)), is implied by:

(12) $\underset{\substack{\delta,\epsilon \\ \delta+\epsilon=\epsilon_0}}{\exists}\ \underset{n}{\forall} (n \geq N_2^* \Rightarrow |f_0[n] - a_0| < \delta \wedge |g_0[n] - b_0| < \epsilon)$.

We have to find δ_0^*, ϵ_1^*, and N_2^* such that

(13) $(\delta_0^* + \epsilon_1^* = \epsilon_0) \bigwedge \underset{n}{\forall} (n \geq N_2^* \Rightarrow |f_0[n] - a_0| < \delta_0^* \wedge |g_0[n] - b_0| < \epsilon_1^*)$.

Formula (13), using (6), is implied by:

$$(\delta_0^* + \epsilon_1^* = \epsilon_0) \bigwedge \underset{n}{\forall} (n \geq N_2^* \Rightarrow \epsilon_1^* > 0 \wedge n \geq N_1[\epsilon_1^*] \wedge |f_0[n] - a_0| < \delta_0^*),$$

which, using (4), is implied by:

$$(\delta_0^* + \epsilon_1^* = \epsilon_0) \bigwedge \underset{n}{\forall} (n \geq N_2^* \Rightarrow \delta_0^* > 0 \wedge \epsilon_1^* > 0 \wedge n \geq N_0[\delta_0^*] \wedge n \geq N_1[\epsilon_1^*]),$$

which, using (Lemma (max)), is implied by:

(14) $(\delta_0^* + \epsilon_1^* = \epsilon_0) \bigwedge \underset{n}{\forall} (n \geq N_2^* \Rightarrow \delta_0^* > 0 \wedge \epsilon_1^* > 0 \wedge n \geq \max[N_0[\delta_0^*], N_1[\epsilon_1^*]])$.

Formula (14) is implied by

(15) $(\delta_0^* + \epsilon_1^* = \epsilon_0) \bigwedge \delta_0^* > 0 \bigwedge \epsilon_1^* > 0 \bigwedge \underset{n}{\forall} (n \geq N_2^* \Rightarrow n \geq \max[N_0[\delta_0^*], N_1[\epsilon_1^*]])$.

Partially solving it, formula (15) is implied by

(16) $(\delta_0^* + \epsilon_1^* = \epsilon_0) \wedge \delta_0^* > 0 \wedge \epsilon_1^* > 0 \wedge (N_2^* = \max[N_0[\delta_0^*], N_1[\epsilon_1^*]])$.

Now,

$(\delta_0^* + \epsilon_1^* = \epsilon_0) \wedge \delta_0^* > 0 \wedge \epsilon_1^* > 0$

can be solved for δ_0^* and ϵ_1^* by a call to Collins cad–method yielding a sample solution

$\delta_0^* \leftarrow \frac{\epsilon_0}{2}$,

$\epsilon_1^* \leftarrow \frac{\epsilon_0}{2}$.

Furthermore, we can immediately solve

$N_2^* = \max[N_0[\delta_0^*], N_1[\epsilon_1^*]]$

for N_2^* by taking

$N_2^* \leftarrow \max[N_0[\frac{\epsilon_0}{2}], N_1[\frac{\epsilon_0}{2}]]$.

Hence formula (16) is solved, and we are done.

□

From the knowledge retrieval point of view the actual proof text is not interesting. The essential fact is that the proposition follows from the available knowledge and, hence, the proposition is retrieved. However, studying the actural proof text generated, one sees that the proof contains additional information on the index bounds for the sum of sequences $f + g$ under the assumption that index bounds for the input sequences f and g are known. Thus, "retrieving" knowledge by proving may also lead to "inventing" new knowledge, which may be worthwhile storing:

Proposition["limit of sum", any[f, g : ℕ → ℝ], any[a, b ∈ ℝ], any[N, M : ℝ → ℕ]

$$\left(\underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{\substack{n \\ n\geq M[\epsilon]}}{\forall} (|f[n] - a| < \epsilon) \bigwedge \underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{\substack{n \\ n\geq N[\epsilon]}}{\forall} (|g[n] - b| < \epsilon)\right) \Rightarrow \underset{\substack{\epsilon \\ \epsilon>0}}{\forall}\ \underset{\substack{n \\ n\geq \max[M[\frac{\epsilon}{2}], N[\frac{\epsilon}{2}]]}}{\forall} (|(f + g)[n] - (a + b)| < \epsilon)\Big]$$

■ Typed and Global Variables

Of course, typed and global variables are of great importance for the practice of formulating realistically big portions of mathematics. Without typed and global variables, it would be nearly impossible to express mathematical theories in a reasonable amount of space. Also, as in programming, typing helps to detect formalization errors in an early stage. In principle, one could of course indicate types explicitly in the formulae as range descriptions of quantified or free variables as we have seen in the examples above. However, for bigger mathematical texts it is much more convenient and efficient to use typed and global variables. On the other hand, explicit indication of types, often tends to blow up the number of lines in proofs without really adding interesting information. We describe, by an example, the mechanism in *Theorema* for working with typed and global variables. Let us suppose we have formulated the following axioms

$$\textbf{Axiom}\Big[\text{"rfl"},\ \forall_{x}\ x \sim x\Big]$$

$$\textbf{Axiom}\Big[\text{"sym"},\ \forall_{x,y}\ (x \sim y \Rightarrow y \sim x)\Big]$$

$$\textbf{Axiom}\Big[\text{"trans"},\ \forall_{x,y,z}\ (x \sim y \wedge y \sim z \Rightarrow x \sim z)\Big]$$

and the definition

Definition["cl", any[x],
 cl[x] = {y | y ~ x}].

Then the proposition

Proposition["cl=", any[x, y],
 x ~ y ⇒ (cl[x] = cl[y])]

can be proved by calling

Prove[Proposition["cl="],
 using → ⟨Axiom["rfl"], Axiom["sym"], Axiom["trans"], Definition["cl"]⟩, by → SetTheoryPCSProver].

It is clear that, having done this proof, we also know that

$$\textbf{Proposition}\Big[\text{"cl=: global"}, \text{any}[\sim],\ \bigwedge\begin{cases}\forall_{x}\ x \sim x \\ \forall_{x,y}\ (x \sim y \Rightarrow y \sim x) \\ \forall_{x,y,z}\ (x \sim y \wedge y \sim z \Rightarrow x \sim z) \\ \forall_{x}\ \text{cl}[x] = \{y \mid y \sim x\}\end{cases} \Rightarrow \forall_{x,y}\ (x \sim y \Rightarrow (\text{cl}[x] = \text{cl}[y]))\Big]$$

and also

$$\textbf{Proposition}\Big[\text{"cl=: global and typed", any}[\sim, U],$$

$$\bigwedge\begin{cases}\underset{x\in U}{\forall}\ x\sim x\\ \underset{x,y\in U}{\forall}\ (x\sim y\Rightarrow y\sim x)\\ \underset{x,y,z\in U}{\forall}\ (x\sim y\wedge y\sim z\Rightarrow x\sim z)\\ \underset{x\in U}{\forall}\ \mathrm{cl}[x]=\{y\in U\mid y\sim x\}\end{cases}\Rightarrow\underset{x,y\in U}{\forall}\ (x\sim y\Rightarrow(\mathrm{cl}[x]=\mathrm{cl}[y]))\Big].$$

From the point of view of building up mathematical knowledge bases, the globalized and the globalized and typed propositions are of course more useful because they can be applied later in more general contexts. On the other hand, a proof of the globalized and the globalized and typed propositions would be less concise and will contain a lot of uninteresting additional proof lines although the essence of the proof is basically the same. Hence, for building up formal knowledge bases, it is a good idea to prove propositions using global constants and untyped variables and to globalize and type them afterwards when storing them into the knowledge base. For this, we (will) provide various tools in *Theorema*. For example, by

$$\textbf{Proposition}\Big[\text{"cl=: global and typed", any}[\sim, U], \text{type}[U],$$

$$\bigwedge\begin{cases}\text{Axiom["rfl"]}\\ \text{Axiom["sym"]}\\ \text{Axiom["trans"]}\\ \text{Definition["cl"]}\end{cases}\Rightarrow\underset{x,y}{\forall}\ (x\sim y\Rightarrow(\mathrm{cl}[x]=\mathrm{cl}[y]))\Big]$$

the global and typed variant of the proposition can be generated automatically.

■ Building Up Structured Mathematical Knowledge Bases

■ Structuring Mathematical Knowledge Needs Creativity

Bourbakism is an important and well-known attempt to compile "all of mathematics" within one formal system, namely first-order Zermelo-Fraenkel set theory. The individual volumes of the Bourbaki book series then contain kind of standard presentations (definitions, theorems, proofs) of the important areas of mathematics. In a sense, one might believe that all current activities in mathematics should just be an expansion of the knowledge structured in the Bourbaki volumes and this is what the Bourbakists may have hoped for. This is, however, not what happens. Most authors of mathematical text books do not care how the topic of their books is structured in the respective Bourbaki volume. Rather, they start from their own "vision" of the field and then go into the presentation of more details or new results. Also, it is a sad fact that the Bourbaki view of mathematics does not include the crucial notion of algorithm and hence, within Bourbakism, there is no natural room for important results on the algorithmic side of mathematics. Thus, Bourbakistic mathematics is too narrow as a frame for future formal mathematical knowledge bases.

Rather, future computer-supported frames for mathematical knowledge management must provide a formal system that is broad enough to encompass both the nonalgorithmic and algorithmic aspect of mathematics and it must give the freedom to authors to structure their area of mathematical interest in their own way. Actually, the more we understand how the routine part of proving can be computer-supported the more it turns out that a decent portion of mathematical creativity goes into *structuring* mathematics rather than into the actual investigation of which conclusions can be drawn from which assumption. For example, generalizing the method of Groebner bases to more and more domains (rings, modules), the actual creative work does not any more go so much into repeating the correctness arguments under weaker assumptions (the arguments are basically known and repeating them under weaker assumptions is necessary but not very demanding) but into finding out in which layers the basic concepts have to be introduced in such a way that we can master more and more domains by basically the same concepts and methods. Stated differently, we believe that there is still room, for instance, for the (n+1)st book on linear algebra although some people may believe that the field is sufficiently studied because the value of a new book on linear algebra may well lie in the way of looking at the field and structuring it in a new way rather than in the amount of new material contained in it.

Hence, we are advocating the construction of formal (logic and software) systems that allow to integrate the nonalgorithmic and algorithmic aspect of mathematics and allow to build up areas of mathematics in various different and competing ways. *Theorema* tries to meet these two requirements. First, it is perfectly possible and practical to formulate algorithms in the frame of *Theorema*. In fact, equational logic and quantifiers with bounded ranges form an elegant and practical programming language within the general predicate logic *Theorema* language. For example, the following formulae from the theory of sorting contain both *algorithms* and *knowledge* about the algorithms and about ingredient notions. In fact, all these formulae can be considered as formulating true statements but some of them can also be considered as describing how to compute certain objects (by applying some of the predicate logic inference rules as computational rules, i.e. as "programming language interpreter"):

$$\textbf{Algorithm}\Big[\text{"sorting by merging", any}[X],\quad \text{stmg}[X] := \begin{cases} X & \Leftarrow \ |X| \leq 1 \\ \text{mg}[\text{stmg}[\text{lsp}[X]], \text{stmg}[\text{rsp}[X]]] & \Leftarrow \ \text{otherwise} \end{cases}\Big]$$

$$\textbf{Algorithm}\Big[\text{"merging", any}[X, Y, a, b, \bar{x}, \bar{y}],$$
$$\text{mg}[\langle\rangle, Y] := Y$$
$$\text{mg}[X, \langle\rangle] := X$$
$$\text{mg}[\langle a, \bar{x}\rangle, \langle b, \bar{y}\rangle] := \begin{cases} a \smile \text{mg}[\langle\bar{x}\rangle, \langle b, \bar{y}\rangle] & \Leftarrow \ a \geq b \\ b \smile \text{mg}[\langle a, \bar{x}\rangle, \langle\bar{y}\rangle] & \Leftarrow \ \text{otherwise} \end{cases}\Big]$$

Lemma["merging sorted tuples yields a sorted tuple" , any[X, Y],
$(\text{ist}[X] \wedge \text{ist}[Y]) \Longrightarrow \text{ist}[\text{mg}[X, Y]]$]

Theorem["correctness of sorting by merging" , any[X],
istv[stmg[X], X]]

Definition["is sorted version", any[X, Y],
$\text{istv}[X, Y] \Longleftrightarrow (\text{ist}[X] \wedge \text{ipm}[X, Y])$]

Definition["is permuted version", any[X, Y],
$\text{ipm}[X, Y] \Longleftrightarrow \ldots$]

Definition["is sorted", any[X],
$\text{ist}[X] \Longleftrightarrow \ldots$]

Second, we provide a frame for the formulation of any part of mathematics in the way a particular author wants to do this. Since knowledge bases are always explicit, variants of propositions in two different formulations of the same mathematical area may well differ from each other but ambiguities can always be resolved because the respective knowledge bases are explicitly formulated. In other words, as a goal for mathematical knowledge management, we do not believe that there should be or will be something like "the knowledge base of mathematics" on which every mathematician should base his future work but there will be a huge number of different, hierarchically organized knowledge bases from which mathematicians can draw well-defined and explicitly specified portions and to which they can add their individual pieces.

■ How Much Knowledge Should be Stored

The question of how many and which of the potentially infinitely many true statements in a mathematical theory should be stored in knowledge bases and how many and which statements are left to be derived any time when a retrieval request is issued seems to be a crucial, if not the crucial question in future theorem-proving based mathematical knowledge bases.

My personal answer to this question is as follows: I propose the paradigm of "computer-supported theory exploration" over the paradigm of "computer-supported isolated theorem proving", see [Buchberger 2000] for more details. Roughly, this paradigm says that, instead of trying the computer-supported proof of individual theorems, one should strive for the computer-supported exploration of entire mathematical theories. In this context, a theory exploration situation is characterized by the following ingredients:

- a collection of known concepts (function and predicate constants)
- "complete" knowledge about the known concepts
- a new concept
- a collection of statements (axioms; in particular, definitions) that describe the connection of the new concept with the known concepts
- a collection of goal statements that describe the connection of the new concept with the known concepts "completely".

Here, the notion of "complete" knowledge is crucial and, in fact, is also the answer to the question how much of the potentially infinite knowledge on given notions should be stored and which knowledge should be retrieved by logical derivation each time it is needed. Roughly, we say that (finite) knowledge is complete if all other logical consequences can be proved by "simple" proofs. More concretely, we believe that "simple" proving basically is "proving" by applying rewrite rules. Of course, this characterization of "completeness" and "simple" is not a mathematical definition (although we believe it could be made into a precise definition). Rather, it is a heuristic recommendation. Let us give an example: We consider, as "completely explored" theory, the theory of arithmetical operations on real numbers together with a theory containing the definition of sequences on real numbers and arithmetical operations on such sequences. Now we introduce the new notion of "limit" by the definition above and we explore various properties of this new notion, for example, the above Proposition["limit of sums"] and similiar propositions. It should be clear that it only makes sense to explore (by proving) and store the "first few interactions" of the new notion limit with the known notions (for example the arithmetical operations on sequences, like addition) and that it does not make sense to explore and store propositions like

$$(\text{limit}[f, a] \land \text{limit}[g, b] \land \text{limit}[h, c]) \Longrightarrow \text{limit}[f * (g + h), a * (b + c)]$$

because this proposition can be "retrieved" (i.e. proved by simple rewrite proving) from the more elementary propositions already proved. (Note that, for the proof of these more elementary propositions that, basically, describe the "first interactions" of the new notion with the known notions, rewrite rule proving is *not* sufficient. Rather, full predicate logic proving is necessary, see the *Theorema*-generated proof above. In fact, these proofs are not so easy and, thus, it is reasonable to store these first few propositions rather than re-proving them every time a retrieval request is issued.)

In the frame of this view of on "storing versus retrieving by proving", "complete knowledge", and "simple proving" another principle seems to be important for future computer-supported mathematical systems: "Lifting knowledge into the status of special inferencing". We again explain this principle (whose details are also described in [Buchberger 2000]) for the above example: After "complete" exploration of the notion limit, i.e. after having proved all the elementary interactions of the new notion 'limit' with the known operations on sequences, retrieving more complex propositions on 'limit' by rewrite rule proving using the elementary interactions as rewrite rule knowledge can and should be replaced by a special inference rule (an "algorithm for computing limits") that retrieves (proves) the more complex propositions by applying the algorithm *without* using the knowledge base . Of course, the correctness of the limit algorithm must be proved and this proof is again a (nontrivial) problem for automated theorem provers. Note, however, that the correctness proof for the limit algorithm is not a proof that can carried out in the theory of 'limit' but a proof on the theory of predicate logic formulae! Thus, building up future mathematical knowledge bases is not only a questions of the hierachical composition of knowledge bases but also a question of building up layers of special inference techniques. This is a view hardly addressed in the literature. In *Theorema*, first attempts have been made to apply inductive provers not only for proving properties of notions on inductively defined structure like the integers or finite tuples but also for proving the correctness statements of special inference rules defined on the inductively defined domain of predicate logic formulae.

■ Formal Training and Mathematical Knowledge Management

Formal mathematics supported by formal math systems is a must for a decisive step forward in the essentail role mathematics should and will play as the core of the technology-based aspect of society. (This does not mean that I believe that society can be based on technology alone. However, surely, the technology-based aspect of the development of society should be as professional as possible and formal mathematics is the tool for making technology professional. Neither does this mean that I believe that invention in mathematics can be produced "automatically" by formal systems. However, formal, computer-supported mathematical systems form the tool on which invention - by human exploration - can be based and made more efficient, flexible, reliable and less repetitive.)

However, creating and using formal mathematical systems can only be done by mathematicians whose formal abilities are much more developed than what we encounter today. Frankly, I believe that, worldwide, the formal abilities of average mathematicians are in a deplorable state. Many mathematicians are even proud that they don't know anything about logic, computers and algorithms. Some believe that what they are doing is so ingenious that one can not and should speak *about* what we are doing in mathematics and how we are doing mathematics. Rather, they believe, one just should *do* mathematics and not care about thinking *about* how we do matheamtics.

In fact, if one analyzes mathematical texts in publications and textbooks, it soon becomes apparent that it would be absolutely silly and counter-productive to make the mathematical knowledge contained in these publications and textbooks available in computer-supported mathematical knowledge bases: Storing false, contradictory, badly formulated, badly structured "knowlegdge" in the mathematical knowledge bases of the future would just result in vast collections of inconsistent and confusing junk. Of course, most times (but not always), there is some truth behind what mathematicians write in their publications and textbooks but the truth can only be revealed if most of the text is re-written, re-structured, read "in the correct way", polished, made explicit with respect to "methods" that are only implicitly contained in the text, put into explicit contexts of definitions and notation used, etc. For example, if in a text on Hilbert spaces, the author defines a function *F* on operators *K* by saying that

$$\mathrm{F}[\mathrm{K}\,.\,\mathrm{K}^*] \;:=\; \sum_{\mathrm{i}=1}^{\infty} \mathrm{f}[\sigma_\mathrm{i}{}^2],$$

where the σ_i are the eigenvalues of *K* and *f* is some known function, then an alarm bell must ring: For formal reasons, *F* can *not* be defined this way because the argument of *F* is a term in *K* and, hence, uniqueness of *F* is not guaranteed except we knew something about the *K*s that are involved in this definition. This "definition" could, hence, result in a contradiction. Also, the σ_i are apparently defined outside the definition, the dependence of *F* on *K* is not clear, etc. Of course, most times, if one reads the pages before (or after) such a definition in such a text, with lots of guess work and good will, the reader may have a chance to detect what the author really means or wanted to say and he may well be able to establish one variant of reading the text in such a way that it is logically consistent and mathematically interesting. However, for this, the reader must sometimes do more than the author.

There exists a myth among mathematicians saying that it would be impossible to write mathematical papers in such a way that everything is explicit and formally correct because then papers would become intolerably long and cumbersome to read. I decisively oppose this myth, i.e. I think that this myth is just not true. There exists another myth among mathematicians stating, as a reasonable goal for future computer-supported math systems, that these systems must become so "intelligent" as to be able to "understand" current mathematical papers even if they are written in the usual, questionable syle. I think this is an "ill-posed" goal. Why should we spend lots of effort on making unintelligent, wrong, unstructured, badly written papers being understood?

I believe that there is only one way out of this waste of effort and permanent confusion-generation in current mathematical research, teaching, and publishing: Mathematicians must devote some of their time to improve their own formal training. Since, most probably, this request will not be appreciated and accepted by a decent percentage of the current generation of mathematicians, serious effort must be put into the formal training of the next generation of mathematicians. As a consequence, in the present and future math and computer science curricula, right at the beginning of the study and then permanently in deeper and deeper layers, sufficient time must be set aside and special courses and training opportunities must be established for the training of students in the formal aspects of mathematics. I believe this effort and time is well spent. Instead of pressing more and more contents into math and computer science curricula I think it is much wiser to spend some little, but well-panned, portion of the math education into formal training. The result will be much more efficiency and ability for studying mathematics; ability to learn and understand mathematical contents much more quickly; improved research potential; more ability to analyze problems and to solve them; faster reading of mathematical literature; ability to polish, improve, and fix what we read; improved allertness of how to use mathematical results; better ability to communicate with potential users of mathematics; etc.

As a consequence, future computer-supported mathematical systems should also be a tool for training the formal abilities of math and computer science students. In the *Theorema* project, therefore, we also address the pedagogical problem of designing and implementing the *Theorema* software system in such a way that it can be used for the formal training of students (and graduates). For this, we are conducting a couple of pedagogical experiments using *Theorema* both in the education of math and computer science freshmen, see [Buchberger 1998], as well as in the first year of the international PhD program for symbolic computation at RISC, see [Buchberger 1992]. The results are not yet satisfactory but promising. They are not satisfactory because we learn from the experiments that a lot of features are still missing in the *Theorema* for making it really attractive for a systematic formal training of math and computer science students. The results are promising because we observe that already the necessity to formulate mathematical contents (concepts, propositions, methods) within a well-defined formal language (a version of predicate logic) has a purifying and clarifying effect on the side of the students. Also, as far as the *Theorema* provers are already working, provers that generate proofs in a "natural" style including explanatory text can be used as personal tutors for training the proving potential of students. Also, students can use the system for building up their own consistent mathematical knowledge bases of what they learned in the various math courses for their future professional life. These knowledge bases contain both mathematical facts (propositions) and methods (algorithms) in a uniform and executable form.

■ References

[Buchberger 1992] B. Buchberger. *Thinking, Speaking, Writing*. Lecture Notes, RISC, Johannes Kepler University, Linz, Austria, 1992.

[Buchberger 1998] B. Buchberger. *Introduction to Mathematics: Combining the Nonalgorithmic and Algorithmic Aspect.* Lecture Notes, Polytechnique University, Hagenberg, Austria.

[Buchberger 1998] B. Buchberger. Theory Exploration with Theorema. *Proceedings of the SYNASC (Symbolic and Numeric Algorithms in Scientific Computing) Conference* (T. Jebelean ed.), University of Timisoara, Rumania, Oct. 4-6, 2000.

[Buchberger et al. 2000]. B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, W. Windsteiger, 2000. The Theorema Project: A Progress Report. In: *8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, St. Andrews, Scotland, August 6-7, M. Kerber and M. Kohlhase (eds.), available from Fachbereich Informatik, Universität des Saarlandes, pp. 100-115.

[Buchberger, Nakagawa 2001]. B. Buchberger, K. Nakagawa. Two Tools for Mathematical Knowledge Management in *Theorema*. Proceedings of the *Workshop on Mathematical Knowledge Management,* Sep. 24-26, 2001, RISC, Austria.

[MKM 2001] Workshop on Mathematical Knowledge Management 2001 at RISC, Hagenberg, Austria, Sep. 24-26, 2001. Call for papers.

[QED 1995] The QED Manifesto. http://www.rbjones.com/rbjpub/logic/qedres00.htm.

[Rudnicki 1992]. Piotr Rudnicki. An overview on the Mizar project. In *Proceedings of the Workshop on Types and Proofs as Programs,* 1992, pp. 311-332.