Organisational Tools for MKM in *Theorema*

Florina Piroi, Bruno Buchberger, Camelia Rosenkranz, Tudor Jebelean

`Florina.Piroi@oeaw.ac.at`
`{Bruno.Buchberger, Camelia.Rosenkranz, Tudor.Jebelean}@risc.uni-linz.ac.at`

Johann Radon Institute for Computational and Applied Mathematics (RICAM),
Austrian Academy of Sciences,
Altenbergerstrasse 69, A-4040 Linz, Austria

and

Research Institute for Symbolic Computation (RISC),
Johannes Kepler University, Linz,
Schloss Hagenberg, A-4232 Hagenberg, Austria

Abstract. In this paper we identify the organizational problems of mathematical knowledge management and describe tools that address some of these problems, namely the management of composite, hierarchical labels for formalized knowledge and instances of the problem of mathematical knowledge retrieval.

Acknowledgements This work was partially supported by the RISC PhD scholarship program of the government of Upper Austria, by the FWF (Austrian Science Foundation) Spezialforschungsbereich F013, project P1302, by the RICAM Institute, and by the EU "Calculemus" Project (HPRN-CT-2000-00102).

1 A View on MKM

In order to specify the exact goal of the tool described in this paper we first describe our view of the new research area Mathematical Knowledge Management (MKM) and its subareas. Roughly, this view was expressed by the second author in the preface of the first conference on MKM and the subsequent special issue of the journal AMAI, see [14] and [18]. Within this view of MKM, the current paper only deals with subarea c2., see below. For other, alternative, views of MKM see the introductions of recent papers on MKM, in particular the ones in [14, 4] and [18].

In our view, the aim of MKM is the computer-support (partially or fully automated) of all phases of the mathematical theories exploration:

- invention of mathematical concepts,
- invention and verification (proof) of mathematical propositions,
- invention of problems,

– invention and verification (correctness proofs) of algorithms that solve problems,

and the structured storage of concepts, propositions, problems, and algorithms in such a way that they can be easily accessed, used and applied at a later time.

MKM in this broad sense is, essentially, a logical activity: All formulae (axioms, definitions of concepts, propositions, problems, and algorithms) must be available in the coherent frame of a logical system, e.g. some version of predicate logic, and the main operation of MKM on these formulae is essentially formal reasoning (in particular formal proving), i.e. reasoning guided by explicit algorithmic rules.

The *Theorema* system is one of the systems whose emphasis is on this *logic* aspect of MKM, which we think is the fundamental aspect of future MKM. Some papers on the logical aspects of MKM within *Theorema* are [17, 9]. The question of computer-supported invention of mathematical knowledge within *Theorema* is treated in [12], the question of computer-supported algorithm synthesis within *Theorema* is treated in [10, 12] and [15]. The formal (computer-supported) reasoning aspect of MKM is *not* a goal of this paper.

On the surface of MKM, however, we are faced also with many additional *organisational problems*, which are important for the practical success of MKM:

a. *The translation of the vast amount of mathematical knowledge which is available only in printed form (in textbooks, journals, etc.) and which has to be brought into a form (e.g. LaTeX), in which it can be processed by computers:* This is the problem of "digitising" mathematical knowledge, see e.g. [39] for a survey on the existing projects in this area or [20]. The *Theorema* project is not engaged in this area of MKM.
b. *The translation of digitised mathematical knowledge, for example in the form of LaTeXfiles, into the form of formulae within some logical system, e.g. predicate logic, so that afterwards they can be processed by reasoning algorithms (in particular theorem proving assistants):* Many current projects are addressing this question, see e.g. MathML [40], OpenMath [19]. The *Theorema* project is not engaged in this area of MKM either. In fact, we think that most of the mathematical papers are logically not sufficiently consistent and explicit for automated extraction of their logical content. Therefore, in our own experiments on formalisation of mathematical theories, we prefer to build-up mathematical theories by radical re-formalisation from scratch. Such re-formalisations may well follow the general flow of presentation in an existing paper or textbook but the actual formulation of the formulae has to be done "by hand" or by formal reasoning tools.
c. *The organisation of big collections of formulae, which are already completely formalised within a logic system (e.g. predicate logic) in "hierarchies of theories":* At the moment, the largest such collection is Mizar [42]. Among other existing ones we mention MBase [29], the Formal Digital Library project [2], the NIST Digital Library of Mathematical Functions [32], Hypertextual Electronic Library of Mathematics [5], the libraries of the theorem provers Isabelle [34], PVS [33], IMPS [23], Coq [7].

Subproblem c., again, has two sub-aspects:

c1. The organisation of formalised mathematical knowledge by means of *mathematical / logical structuring* mechanisms like domains, functors, and categories. *Theorema* puts a particular emphasis on this aspect, see for example, [11]. However, this aspect is not treated in this paper either.

c2. The additional *assignment of various kinds of labels* to formulae and collections of formulae so that blocks of mathematical knowledge can be identified and combined in various ways without actually going into the "semantics" of the formulae.

In traditional mathematical texts labels are ubiquitous but a systematic management of labels is, normally, not considered to be important nor is it feasible. In contrast, in the build-up of completely formalised mathematical knowledge bases, the systematic design and processing of structured labels (i.e. individual labels like "(1)", "(2)" or "(associativity)" etc., hierarchical section headings, key words like "definition" and "theorem", names of files etc.) becomes vital for the automated structuring and re-structuring of collections of formulae as input to formal reasoning tools like provers, simplifiers, algorithm verifiers, model checkers etc.. Consequently, we need algorithmic tools that handle all types of labels and allow us to partition and combine, structure and re-structure mathematical knowledge bases.

To avoid misunderstandings, we emphasize that, in our view, labels do not intend to have any logical meaning or functionality. This is in contrast to the goal of "annotations", etc. as, for example, in [40, 19] and [28], which convey at least part of the semantics. In our view, the meaning of formulae is solely defined by their inclusion into the context of collection of other formulae (mathematical knowledge bases). Labels only help in addressing, referencing, selecting individual formulae in knowledge bases and in partitioning and re-combining (small and big) collections of formulae. Summarising, in the view of this paper, the functionality of labels is purely organisational and not logical. We also remark that the concept of labels has to be distinguished from the concept of "comments". The latter is only meant as meta-level guides for the human readers of mathematical knowledge bases.

An important purpose of the organisation of formal content is to facilitate *knowledge retrieval*. Namely, it should be easy for a user to find out whether a certain formula occurs in the knowledge base or it can be inferred from it in few, easy proving steps. In fact, the view of querying as (simple) proving has a much broader scope [8], and we are currently working on a more powerful retrieval algorithm that will realize this view. The notion of simplicity (querying as "simple" inferences) depends on the theory in which one works. For example, proving equalities in group theory can be regarded as simple by using the convergent rewrite system obtained from the Knuth-Bendix procedure; but proving equalities in number theory may of course be arbitrarily complex. In this paper, however, we will not enter into these questions.

This paper deals only with subproblem c2. and the simplest forms of knowledge retrieval. The plan of the paper is as follows: In the next section we give a quick review of the related work, then we describe the problem of managing labels and finding knowledge by an example. In section 4 we present the realization of the main routines of our tools. Section 5 contains some final comments. Parts of this paper are contained in the first author's thesis [37].

2 A Short Review of the Related Work

It seems that handling labels in the sense specified in this paper is not an explicit goal in the current MKM systems. However, work on annotation of (collections of) formulae and formula editing has an overlap with and relevance for the work presented in this paper. Therefore, in this subsection, we give a short review on the pertinent papers.

Starting from the display of formulae as graphics in internet document, W3C issued an MathML recommendation, [40], for displaying and communicating formulae by means different from images. Being an application of XML, MathML benefits of the existing tools that manipulate XML files. Though it does offer some semantics of the symbols in the mathematical formulae, the set of these symbols is too restricted when compared to those used by working mathematicians. To ameliorate this situation projects like OpenMath [19] and OMDoc [28] emerged. The OpenMath standard concentrates on representing mathematical expressions together with their semantics, allowing them to be exchanged between computer programs, stored in databases, or published on the world wide web. At a first glance, one can view OpenMath as extending the MathML capabilities by using "content dictionaries" where mathematical symbols are defined syntactically and semantically. OMDoc is an extension of OpenMath and MathML, adding capabilities of describing the mathematical context of the OpenMath objects used.

A drawback of the standards mentioned above is that the coherence of the different documents (e.g. content dictionaries) is not automatically checked. This has to be done by a human, a task that can be rather difficult because the representation formats are not human oriented. This representation confronts us with another issue: publishing mathematics using these representations is not attractive for the everyday mathematician. There is ongoing work to improve this state of facts, like the work described in [24] and the one in [27].

To our knowledge at the time of writing this paper, within most of the proving systems, the users are typing their documents in an Emacs-based editor or something similar, see for example, Mizar [42], HOL [25], CoQ [7], PVS [33]. 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 [5] and MBase [29]. (Helm includes part of the libraries of the proving assistants NuPRL [21] and Coq [7], MBase includes libraries from Omega [41] and TPS [3]). 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 [22] creates and manipulates OpenMath objects and within ActiveMath [31], jEditOQMath is a package of tools for editing and managing documents in OMDoc format [1].

In matters that regard searches in libraries of mathematical knowledge one notices a certain trend towards the usage of moderate versions of higher-order unification. Undergoing research tries to establish a reasonable compromise between the power of the search routines and their efficiency. We give a short overview of some systems that make use of search tools.

In [29] a future improvement to the MBase system with respect to retrieval is mentioned. The intention is to use inference procedures, like higher-order matching, for formula retrieval. We are not aware whether this addition to the system was done. The retrieval mechanism of the HELM system uses data mining techniques on the metadata of the formulae in the knowledge base to obtain a set of constraints on the metadata of the given pattern. Then, the knowledge base is filtered to obtain a set of candidate formulae on which the matching operation is iterated. Within the Mizar system there is ongoing work on developing a user-friendly query language for the Mizar Library [6]. In this frame, to retrieve a theorem the user combines a set of filters on certain metadata. The metadata, in fact a set of indices, is pre-computed automatically from the entire library. We also note here that Mizar uses labels to access formulae when checking proofs. The formulae are collected in files (called articles), numbers are given to them, but no grouping of formulae can be observed by the user.

3 Problem Description by an Example

Let us take a look at the screen shots in Figures 1 and 2 of the appendix. They present part of the contents of two Mathematica notebooks storing text and formulae. The formulae are in predicate logic, in the *Theorema* external syntax. The formulae and the text in these files are grouped under certain headings in sections, subsections, etc. Such files can either be the result of an automated process, or can be created by a human user, via Mathematica's and *Theorema*'s front end environments. Typing the documents is done in a WYSIWYG style: Both Mathematica and *Theorema* provide several tools and toolbars for creating documents and typing mathematical formulae in a user-friendly way.

The contents of the first file (Figure 1) describes the basic notions of the tuple theory. It gives the definitions of the tuple concept, the definitions of operations that can be performed on tuples (reversion, concatenation, etc), and propositions involving the defined concepts. The second file (Figure 2) stores axioms and definitions in the domain of natural numbers, where the natural numbers are expressed by the Peano axioms.

Note that this is only a small example. In practice we may have dozens of files, each with hundreds of formulae. Case studies that involve files with a large number of formulae are, for example, the algorithm synthesis [15] and the Gröbner rings [11] case studies. The didactic project using *Theorema*, CreaComp, is focused on producing large knowledge bases of formulae.

We can have the following different work scenarios:

- assume that we want to investigate some algorithms for sorting tuples of natural numbers. For this, we decide that we need knowledge about tuples and natural numbers. Since the two files already contain the formalisation of this knowledge, we would like to use it, combine it, extend it, and produce a third file containing the new knowledge (Figure 5).
- assume that we want to select some of the knowledge contained in some of the files and give it as an input to one of the automated reasoning tools of *Theorema*.
- assume we try to prove the correctness theorem for one of the sorting algorithms in the newly created file, and that the proof failed. Taking a closer look at the proof we notice that additional knowledge on certain symbols in the proof may make the proof successful. We can type in the needed knowledge (and give the proof another try) or we can search for it in the files with formalised knowledge we have available.

For each of the scenarios above we need means to access and search the desired knowledge within the given files. The most natural way to access formulae and groups of formulae is access by position in files, by section and subsection headings, keywords or labels. Searching in the given files for (groups of) formulae can be done either by position in files, by headings, etc., by labels (where regular expressions are accepted), or by formulating the structure of the formulae we hope to find. Since this is not yet possible in the current native Mathematica notebooks, we created such tools that help us in our working scenarios (see the next section).

4 Description of the MKM Tools

4.1 Starting Point for the Development of the Tools

The implementation of our tools is done in the frame of Mathematica [43] and *Theorema* [9, 17] which is built on top of Mathematica. As a mathematical editing environment, Mathematica offers a very good front end that gives the possibility of combining text, mathematical expressions, graphics, and code in the same document, called notebook. Another feature of Mathematica, which is used by *Theorema*, is two-dimensional syntax programming (see `MakeExpression` in [43], Section 2.9.17).

The tools for label management described in this paper, take as input Mathematica notebooks which contain comments in text cells and predicate logic formulae in input cells. The formulae are given in *Theorema* external syntax. Also, in these notebooks, labels of various kinds (section heading, key words like "definition", "theorem" etc., and individual labels) can be attached to formulae and groups of formulae. For this, we developed a particular Mathematica stylesheet. A Mathematica stylesheet is a special kind of notebook that defines the styles to be used in other notebooks ([43] Section 2.10). By our stylesheet keywords like “definition”, “theorem”, “property”, etc. can be attached to entire sections, subsections, etc. Formulae that occur under these headings will have attached the keyword given by the style of the headings. If desired, the user can also override these keywords by keywords at the level of individual formulae. With the help of this stylesheet, our can identify, select, and recombine formal parts of documents and use them, for example, as input to automated reasoners.

The MKM tools operate on libraries of *Theorema* notebooks. A library is a collection of Mathematica notebooks using the above stylesheet. By the tools described below it is guaranteed that the notebook labels are unique. Also, we provide an extra index file that lists all notebooks in the library and also describes the mutual inclusion of notebooks in the library. (In fact, in addition, the Mathematica package facility is used in the organisation of the notebooks in the library for speeding up reloading of notebooks. However, since this is only a technical detail which does not add to the organisational functionality of our tools, we do not refer to packages anymore in this paper).

In the following, we will describe the main tools which we designed and implemented for achieving the objectives specified in Section 3. For the convenience of the user these tools can be accessed also by a new *Theorema* toolbar, called ‘Library Utilities’ which, however, we do not describe in this paper.

4.2 Tool for Systematic Generation of Hierarchical Labels

An example of a *Theorema* notebook is given in Figure 1 of the appendix. It has a notebook title (“Basic Notions: Tuples”) and a notebook label (“BN:Tuples”). Formulae, in input cells, are grouped under section and subsection headings. Hierarchically grouping cells in sections, subsections and so on, is a feature of Mathematica notebooks. It is common practice that larger notebooks have chapters, sections and so on, each represented by groups of cells. The extent of these groups is indicated by a bracket on the right ([43]), which can also be conveniently used for optical contraction of sections to obtain an easy overview.

The headings of the cell groups, the notebook title, the notebook label, and labels of individual formulae are the components used for generating and attaching unique composite labels to all formulae and groups of formulae in the notebook. If the notebook label is not present in the document, a notebook label is generated from the notebook title such that every notebook in the notebook library has a unique identifier. For this reason, the notebook title is a mandatory element in the *Theorema* notebooks. User given notebook labels are checked against the list of existing notebook labels (extracted from the library index file). The user is notified when the notebook label is already used by another *Theorema* notebook.

From the notebook title, notebook label, section headings, etc. provided by the user our tool automatically generates composite labels for each section, subsection, etc., and individual formula in the notebook. These composite labels are generated in three variants which we call long, short, and decimal composite labels, respectively. (We remark here that any other information present in the notebook, for example comments posted in Mathematica "text cells", is skipped by the algorithmic process.)

In Figure 1 the user provided notebook and theory labels. The user also provided all of the headings in the notebook, among them "Relation on Tuples: Element". The generated labels are: "BN:Tuples.Axioms of Equality and Equality of Tuples" for the long label variant, "BN:Tuples.AxiEquEquTup.2" for the short label variant, and "BN:Tuples.2" for the decimal label variant (Figure 3).

The short variant of the label is obtained from the long variant by a simple string truncation algorithm, with some proviso for preserving uniqueness. The period in the above labels variants plays the role of a separator, displaying the composite structure of the labels generated.

In Mathematica, notebooks are represented as Mathematica expressions. The label generating routine takes as input the Mathematica expression of a *Theorema* notebook. The notebook expression has a recursive structure, reflecting the grouping and sub-grouping of cells in the notebook. Correspondingly the label generating routine proceeds recursively. The label variants are created by concatenation operations, where the operands are, depending on the label variant, the text of the heading, the short text of the headings (obtained by string truncation algorithms), and notebook counters. Eventual user-given labels are taken into account for the generation of the short and long label variants. Each of the labels is prepended the notebook label so that any label we look at, in the *Theorema* notebook, contains the notebook label as a substring. Because the notebook label is unique among notebook labels in the library, we are sure that the generated labels uniquely identify the formulae and groups of formulae within the library.

Figures 3 and 4 in the appendix show parts of the notebooks in Figures 1 and 2 after they have been processed by the label generating routine. The labels that can be seen right above the formulae and groups of formulae are the decimal label variants (Figure 3) and the long variants (Figure 4). The user is given the possibility to chose which label variant to see. From now on, when we use the word 'label' we refer to one of the three variants of a label.

Now, groups of formulae and individual formulae of *Theorema* notebooks can be referenced by composite labels, can be used for composing new *Theorema* notebooks and knowledge bases, can be given as input to formal reasoners. We will describe how these actions can be performed in the next section.

4.3 Tool for Including Formal Parts of Notebooks into Other Notebooks

When we compose a new *Theorema* notebook we may want to use existing formalised knowledge, stored in other *Theorema* notebooks. For this we implemented an 'Include' command with the following structure

$$\texttt{Include}[\texttt{Label}_1, \texttt{Label}_2, \ \ldots\ , \ \texttt{Label}_n, \ \mathit{Options}].$$

The command understands two options: "IncludeStyle" (which can take the values "Dynamic" or "Static") and "KeepStructure" (with values "True" or

"False"). Based on the labels given as parameters, either links to the knowledge the labels refer to (' IncludeStyle ⟶ "Dynamic" ') are created, or the knowledge is copied into the current notebook (' IncludeStyle ⟶ "Static" ') in which case new labels will be generated for them. Furthermore in the latter case, by the option "KeepStructure", we can chose to keep or not the grouping structure which the knowledge had in the *Theorema* notebook from where it is copied. In both cases, in the *Theorema* notebook being composed, corresponding cells are inserted after the ones that contain the 'Include' command (Figure 5). If the knowledge was included dynamically, a click on any of the links in the newly created cells will open the respective *Theorema* notebook where the included (groups of) formulae are stored, and show them to us. If the static inclusion was used, a click on any of the links will open a sub-notebook which contains the respective copied (groups of) formulae. Figure 6 shows such a sub-notebook. Here we can modify the formulae as we desire, to correspond to the theory we are composing. As a remark, when dynamic inclusion is used, any given value to the "KeepStructure" option is ignored. The dynamic inclusion is useful for minimising duplication of formalised mathematical content, while static inclusion gives us the possibility to concentrate knowledge dispersed in various notebooks in the library in one *Theorema* notebook and use this notebook independent of the library.

4.4 Tool for Finding Knowledge in the Library

Knowledge finding (retrieval) comes in several flavours. We focus here on tools that support only three types of knowledge retrieval. The first one is retrieval by label which performs a textual search on the labels of the formulae. Another tool tackles purely syntactic queries that allow the retrieving of formulae containing certain given constant symbols. Finally, the most elaborate, in this paper, answers queries using matching techniques.

Find By Label: With this tool, we can search for knowledge when we know only parts of the label that references it. The command that does the search has the following form:

$$\texttt{LabelLookup}[\ \mathit{input}\ , \{\texttt{Label}_1, \texttt{Label}_2,\ \dots\ ,\ \texttt{Label}_n\}].$$

where *input* is a string that can use regular expression string patterns. For example, searching for knowledge that has "associative" as part of its label, in the *Theorema* notebook with the label "BN:Tuples", is done with the following command:

$$\texttt{LabelLookup}[\ *\texttt{associative}\ ,\ \{\text{“}\texttt{BN : Tuples}\text{”}\}].$$

The given result is a list of formulae that have "associative" as a postfix of their label (for at least one of the label's variants).

Find By Constant Symbols: Given an input that contains certain (object, function, predicate) constant symbols we want to find formulae in our library of *Theorema* notebooks that contain those symbols. The synopsis of the command that does this is

$$\texttt{SymbolLookup}[\ \textit{input}\ , \{\texttt{Label}_1, \texttt{Label}_2,\ \ldots\ ,\ \texttt{Label}_n\},\ \texttt{FilterType} \longrightarrow \textit{value}]$$

where *input* can be either a list of symbols or a list of formulae from where the tool will extract the constant symbols. Mixed lists are also allowed.

The tool will look-up, in the formal knowledge referred by the list of the given labels, all the formulae that contain the constant symbols. If *value* is "All" each of the returned formulae contains all of the searched constant symbols. If *value* is "Any" formulae that contain at least one of the given constant symbols will be part of the returned result. Logical connectives and quantifiers are not taken into account as constant symbols by this tool.

As an example, looking up in the *Theorema* notebook with the label "BN:Tuples" for formulae that contain both 'is-empty-tuple' and '$\smile$' (tuple prepend) we use

$$\texttt{SymbolLookup}[\{\texttt{is-empty-tuple},\ \smile\}, \{\text{"}\texttt{BN : Tuples}\text{"}\},\ \texttt{FilterType} \rightarrow \text{"}\texttt{All}\text{"}]$$

and the returned result is an empty list, meaning that no formula, in the searched group of formulae, contains both these symbols. To find the formulae that contain any of these symbols we use the option value "Any" and the tool returns three formulae that contain the symbol 'is-empty-tuple' and eight formulae that contain the tuple prepend symbol, $\smile$, one of them is given below, where overbarred letters represent sequence variables:

$$\begin{array}{l}\texttt{Proposition}[\text{"BN : Tuples.6.2.5"}] \\ \forall_{\overline{\mathtt{x}},\mathtt{a},\overline{\mathtt{y}}}((\mathtt{a} \smile \langle\overline{\mathtt{x}}\rangle) \asymp \langle\overline{\mathtt{y}}\rangle = \mathtt{a} \smile (\langle\overline{\mathtt{x}}\rangle \asymp \langle\overline{\mathtt{y}}\rangle)).\end{array}$$

Find By Formula Structure: The tool we are to describe now uses the Mathematica notion of "pattern" ([43], Section 2.3), which gives convenient means for describing subsets of expressions.

Patterns are a versatile and powerful way of differentiating and grouping formulae by various requirements, like for example, similarities in formula structures. With the help of patterns, we can express the kind of formulae we are looking for. The command that implements this tool has the form

$$\texttt{FormulaLookup}[\textit{formula}, \{\texttt{Label}_1, \texttt{Label}_2,\ \ldots\ ,\ \texttt{Label}_n\}]$$

where *formula* is a higher-order formula that may contain Mathematica patterns. When executed, the command will look-up in the content referred by the labels $\texttt{Label}_1$ to $\texttt{Label}_n$ for formulae that match the pattern expressed by *formula*. If a formula has a subformula that matches *formula*, it is also returned as a search result. For example, if we want to search all the formulae in the theory "BN:Tuples" that use the concatenation operation, $\asymp$, on the left-hand side of the equal sign we can express this as

$$\texttt{FormulaLookup}[\ \_ \asymp \_\ =\ \_\ ,\ \{\text{"}\texttt{BN : Tuples}\text{"}\}].$$

One of the formulae in the returned search result is

$$\texttt{Proposition}[\text{“}\texttt{BN : Tuples.7.2}\text{”}]$$
$$\forall_{\overline{\mathtt{x}},\overline{\mathtt{y}}}(\langle\overline{\mathtt{x}}\rangle \asymp \langle\overline{\mathtt{y}}\rangle = \langle\overline{\mathtt{x}},\overline{\mathtt{y}}\rangle).$$

Among the formulae in the search result we will find the proposition labelled “BN:Tuples.6.2.5”, which we found also by using the `SymbolLookup` tool.

In the formula pattern above, the underscore symbol stands for a single arbitrary Mathematica expression. Two underscores stand for a sequence of at least one expression, while three underscores stand for any sequences of expressions, even empty ones.

Semantic Find: Probably the most sophisticated form of mathematical knowledge retrieval and also the most specific to mathematics, is the semantical based retrieval, in which the syntax (e.g. the name of the searched item) plays little or no role.

One typical example of semantical retrieval is the search for a *function* fulfiling certain properties. This type of search occurs in algorithm synthesis [16], but also in the context of proving. For instance, in the context of the synthesis of the sorting algorithm one needs to find function $F^?$ (where the small question mark denotes that the symbol F is to be found) satisfying

$$\forall_t \texttt{is-trivial-tuple}[t] \Rightarrow \texttt{is-sorted-version}[t, F^?[t]],$$

and one already has in the knowledge base:

$$\forall_t \texttt{is-trivial-tuple}[t] \Rightarrow \texttt{is-sorted-version}[t, t].$$

Our system implements two methods for finding unknown functions: (a) search by signature and (b) term construction. For space reasons, we will describe how these methods proceed, without giving examples of command calls, as we did in the sections above:

(a) *Search by signature:* consists in extracting, from the given formula, the signature of unknown function, and selecting from the knowledge base all functions which have the same signature. (By preprocessing of the knowledge, one already has a table of signatures). All selected functions are *candidates*, and one tries for each one in turn to prove the required property. If successful, then the desired function is found. This method can find the required function only if it already exists in the knowledge base

(b) *Term construction:* This method can find existing functions, but also functions which have not been constructed yet (as e.g. in the example with is-sorted-version). In general, the method accepts a quantified formula as the “goal”, which formula contains an occurrence of an unknown function, say $F^?[x_1, ..., x_k]$. The goal (without loss of generality, we may assume it in prenex form) is first transformed by replacing the unknown function “call” with a new existentially quantified variable v_F , where the new existential quantifier is inserted exactly after the last quantifier corresponding to $x_1, ..., x_k$. After that, the goal is transformed by introducing Skolem functions for the universally quantified variables (each Skolem function having as arguments the existential quantied variables occurring before the respective universal variable), and finally by removing all quantifiers. In the example mentioned above, the goal becomes:

$$\texttt{is-trivial-tuple}[t_0] \Rightarrow \texttt{is-sorted-version}[t_0[], v_F].$$

The corresponding assumption in the knowledge base will also be transformed, but by usual Skolemization:

$$\texttt{is-trivial-tuple}[t] \Rightarrow \texttt{is-sorted-version}[t, t].$$

By unification between the goal and the assumption, one obtains the term $t_0[]$ as the substitution for v_F, and by reverting to the original variables, one obtains $F[t] = t$.

4.5 Tool for Using Selected Formal Parts of Notebooks

The various reasoners (provers, simplifiers, and solvers) of *Theorema* can be called by instructions of the following structure

$$\texttt{Reason}[\texttt{Goal}, \texttt{using} \rightarrow \texttt{KnowledgeBase}, \texttt{by} \rightarrow \texttt{ReasoningMethod}],$$

where 'Reason' is '`Prove`', '`Compute`', or '`Solve`'; '`KnowledgeBase`' is given by $\langle \texttt{Label}_1, \ldots, \texttt{Label}_n \rangle$ and $\texttt{Label}_1, \ldots, \texttt{Label}_n$ are composite labels of collections of formulae in the notebook library.

For example, to prove the correctness theorem for the merge-sort algorithm (see Figure 5) we use the instruction

$$\begin{aligned}\texttt{Prove}[\ & \text{“}\texttt{SortTuples.9.2}\text{”}, \texttt{using} \rightarrow \langle \text{“}\texttt{SortTuples.2}\text{”}, \\ & \text{“}\texttt{NN : Basic.Defining} \leq \texttt{w.r.tsuccessor}, \texttt{Properties}\text{”}\rangle, \\ & \texttt{by} \rightarrow \texttt{TupleEqIndProver}]\end{aligned}$$

where '`TupleEqIndProver`' is a *Theorema* prover that combines rewriting and induction over tuples.

Alternatively, we also implemented the following '`Theory`' construct:

$$\texttt{Theory}[\texttt{Label}, \langle \texttt{Label}_1, \ldots, \texttt{Label}_n \rangle]$$

which is something like a temporary assignment of a new label to specified collections of formulae so that we can formulate calls to reasoners also in the following form

$$\texttt{Reason}[\texttt{Goal}, \texttt{using} \rightarrow \texttt{Label}, \texttt{by} \rightarrow \texttt{ReasoningMethod}],$$

For example, the above `Prove` call can be formulated in the following way:

$$\begin{aligned}\texttt{Theory}\ & [\text{“}\texttt{Merge sort Alg. Def. and} \leq \texttt{Prop.}\text{”}, \langle \text{“}\texttt{SortTuples.2}\text{”}, \\ & \text{“}\texttt{NN : Basic.Defining} \leq \texttt{w.r.tsuccessor}, \texttt{Properties}\text{”}\rangle]; \\ \texttt{Prove}[\ & \text{“}\texttt{SortTuples.9.2}\text{”}, \texttt{using} \rightarrow \langle \text{“}\texttt{Merge sort Alg. Def. and} \leq \texttt{Prop.}\text{”}, \\ & \texttt{by} \rightarrow \texttt{TupleEqIndProver}]\end{aligned}$$

The proof of the merge-sort correctness theorem, above, is likely to fail because of missing knowledge about tuples (we gave the prover only the definition of the merge-sort algorithm and auxiliary functions in it, and some knowledge about less-equal, $\leq$). We have now two choices: we can give the knowledge to the prover explicitely, by adding the label that references it to the `using` list; or we can tell the proof search mechanism of *Theorema* to look up this knowledge in the library. For our example, this can be done as follows:

$$\texttt{Prove}[\text{“}\texttt{SortTuples.9.2}\text{”}, \texttt{using} \rightarrow \langle\text{“}\texttt{Merge sort Alg. Def. and} \leq \texttt{Prop.}\text{”}, \\ \texttt{withLibSupport} \rightarrow \{\text{“}\texttt{BN : Tuples}\text{”}\}, \texttt{by} \rightarrow \texttt{TupleEqIndProver}]$$

which has the consequence that, before the proof is started, the constant symbols in the formula “`SortTuples.9.2`” are extracted, then a search by symbol look-up in the “BN:Tuples” *Theorema* notebook is done. The symbol look-up uses the tool described in the previous section, with the option `FilterType` $\rightarrow$ “`Any`”. The formulae returned are, then, added to the prover's knowledge base.

5 Conclusion

We designed and implemented simple tools that allow to reference and search in specific parts of collections of mathematical knowledge bases organised in libraries of *Theorema* notebooks. These tools generate systematically composite hierarchical labels for all the sections, subsections, etc. and the individual formulae of *Theorema* notebooks from section headings and individual labels of formulae in the original notebooks provided by the user. With these tools one then can quickly compose specific new notebooks and knowledge bases (as input to the formal reasoners of the *Theorema* system) using the composite hierarchical labels. Furthermore, the syntactic and semantic search tools give users the possibility to retrieve mathematical knowledge either by syntactic or by semantic means.

These tools can also be used for knowledge bases in other systems by translation of the formulae formats between systems. Translators to and from *Theorema* collections of formulae are implemented for several deduction systems [30]. More such translators are under way and will be added to the system depending on the available man power. Alternatively, one could design and implement similar labelling management tools directly in other systems.

Typical users of the described tools are “working mathematicians” who want to build up and explore mathematical theories within the *Theorema* system.

Seemingly, label management is an easy subject of mathematical knowledge management. However, we believe that, in fact, systematic and ecient label management is quite significant for the user-friendliness of future mathematical knowledge management systems and needs systematic treatment. For different purposes within MKM the choice of labels must meet different criteria. For example, for human readers of mathematical knowledge bases (e.g. in the form of *Theorema* notebooks) long textual labels may be preferable whereas in the presentation of proofs short version of labels are desirable in order not to disrupt the flow of the proof presentation. In the extreme case, if the proof presentation style of 'Focus Windows' is used (see [38]), one even does not need labels in proof presentations. However, at the same time, labels as references for organising new knowledge bases from given ones, for example as input to provers, are very important. Thus, flexible label management tools on top of the logic tools of MKM systems are necessary.

References

1. jEditOQMath tools package.
2. S. Allen, M. Bickford, R. Constable, R. Eaton, C. Kreitz, and L. Lorigo. FDL: A prototype formal digital library, 2002. `http://www.nuprl.org/FDLproject/02cucs-fdl.html`.
3. P.B. Andrews, M. Bishop, and C.E. Brown. System description: TPS: A theorem proving system for type theory. In D. McAllester, editor, *Automated Deduction - CADE-17*, volume 1831 of *LNAI*, pages 164–169, Pittsburgh, PA, 2000. Springer–Verlag.
4. A. Asperti, B. Buchberger, and J. H. Davenport, editors. *Mathematical Knowledge Management*, volume 2594 of *Lecture Notes in Computer Science*. Springer - Verlag Berlin, 2003.
5. A. Asperti, L. Padovani, C. Sacerdoti Coen, F. Guidi, and I. Schena. Mathematical knowledge management in HELM. *Annals of Mathematics and Artificial Intelligence*, 38(1):27–46, 2003.
6. G. Bancerek and P. Rudnicki. Information retrieval in MML. In [4], pages 119–132.
7. Y. Bertot and P. Castèran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.
8. B. Buchberger. Mathematical Knowledge Management using Theorema. In [14].
9. B. Buchberger. Theorema: A Proving System Based on Mathematica. *The Mathematica Journal*, 8(2):247–252, 2001.
10. B. Buchberger. Algorithm invention and verification by Lazy Thinking. In [35], pages 2–26, 2003.
11. B. Buchberger. Groebner Rings in Theorema: A Case Study in Functors and Categories. Technical report, SFB Report No. 2003-49, Johannes Kepler University Linz, Spezialforschungsbereich F013, November 2003.
12. B. Buchberger. Algorithm-supported mathematical theory exploration: A personal view and strategy. In [13], pages 236–250, 2004.
13. B. Buchberger and J. A. Campbell, editors. *Proc. of the 7th Int. Conference on Artificial Intelligence and Symbolic Computation, AISC'04*, volume 3249 of *LNAI*, Hagenberg, Austria, 2004. Springer.
14. B. Buchberger and O. Caprotti, editors. *First International Workshop on Mathematical Knowledge Management (MKM 2001)*, 24-26 Sept 2001.
15. B. Buchberger and A. Crăciun. Algorithm synthesis by Lazy Thinking: Examples and implementation in *Theorema*. In F. Kamareddine, editor, *Proc. of the Mathematical Knowledge Management Workshop, Edinburgh*, volume 93 of *Electronic Notes on Theoretical Computer Science*, pages 24–59. Elsevier, 2003.
16. B. Buchberger and A. Crăciun. Algorithm synthesis by Lazy Thinking: Using problem schemes. In [36], pages 90–106, 2004.

17. B. Buchberger, C. Dupré, T. Jebelean, F. Kriftner, K. Nakagawa, D. Văsaru, and W. Windsteiger. The *Theorema* project: A progress report. In [26], pages 98–113, 2000.
18. B. Buchberger, G. Gonnet, and M. Hazewinkel, editors. *Mathematical Knowledge Management*, volume 38 of *Special Issue of Annals of Mathematics and Artificial Intelligence*. Kluwer Academic Publishers, Dordrecht, Netherlands, May 2003.
19. O. Caprotti and D. Carlisle. OpenMath and MathML: Semantic mark up for mathematics. *ACM Crossroads, Special Issue on Markup Languages*, 6(2), 1999. ACM Press.
20. K. Chan and D. Yeung. Mathematical expression recognition: A survey. *International Journal on Document Analysis and Recognition*, 3(1):3–15, August 2000.
21. R. Constable. *Implementing Mathematics Using the Nuprl Proof Development System*. Prentice Hall, 1986.
22. Laurent Dirat. JOME: Java openmath editor.
23. W. M. Farmer, J. D. Guttman, and F. J. Thayer. IMPS: An Interactive Mathematical Proof System. *J. Automated Reasoning*, 11(2):213–248, 1993.
24. G. Goguadze and A. González Palomo. Adapting mainstream editors for semantic authoring of mathematics. Presented at the Mathematical Knowledge Management Symposium, November 2003.
25. M. Gordon and T. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic*. Cambridge University Press, 1993.
26. M. Kerber and M. Kohlhase, editors. *Proc. of Calculemus'2000*, St. Andrews, UK, 2000.
27. A. Kohlhase and M. Kohlhase. Cpoint: Dissolving the author's dilemma. In A. Asperti, G. Bancerek, and A. Trybulec, editors, *Proceedings of Mathematical Knowledge Management, Third International Conference, MKM 2004*, volume 3119 of *Lecture Notes in Computer Science*, pages 175–189. Springer, 2004.
28. M. Kohlhase. OMDoc: An infrastructure for OpenMath content dictionary information. *ACM SIGSAM Bulletin*, 34(2):43–48, 2000.
29. M. Kohlhase and A. Franke. MBase: Representing knowledge and context for the integration of mathematical software systems. *Journal of Symbolic Computation*, 23(4):365–402, 2001.
30. T. Kutsia and K. Nakagawa. An Interface between Theorema and External Automated Deduction Systems. In S. Linton and R. Sebastiani, editors, *Proceedings of 9th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus'01)*, pages 178–182, Siena, Italy, Jun 21–23 2001.
31. P. Libbrecht, E. Melis, and C. Ullrich. The ActiveMath learning environment. System description. In *Calculemus Workshop at IJCAR*, pages 173–177, 2001.
32. D. W. Lozier. NIST digital library of mathematical functions. pages 105–119.
33. S. Owre, J. Rushby, N. Shankar, and D. Stringer-Calvert. PVS: An experience report. In D. Hutter, W. Stephan, P. Traverso, and M. Ullman, editors, *Applied Formal Methods FM–Trends '98*, volume 1641 of *LNCS*, pages 338–345. Springer–Verlag, Germany, 2004.
34. L. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.
35. D. Petcu, V. Negru, D. Zaharie, and T. Jebelean, editors. *Proc. of SYNASC'03, 5th Int. Workshop on Symbolic and Numeric Algorithms for Scientific Computing*, Timişoara, Romania, 2003. Mirton.
36. D. Petcu, V. Negru, D. Zaharie, and T. Jebelean, editors. *Proc. of SYNASC'04, 6th Int. Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Timişoara, Romania, 2004. Mirton.

37. F. Piroi. *Tools for Using Automated Provers in Mathematical Theory Exploration*. PhD thesis, RISC, Johannes Kepler University, Linz, Austria, August 2004.
38. F. Piroi and B. Buchberger. Focus windows: A new technique for proof presentation. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors, *Artificial Intelligence, Automated Reasoning and Symbolic Computation. Proceedings of Joint AISC'2002 - Calculemus'2002 Conference*, volume 2385 of *LNAI*, pages 290–304, Marseille, France, July 2002. Springer Verlag.
39. S. Rockey. Mathematics digitization, at Cornell University, Mathematics Library. `http://www.library.cornell.edu/math/digitalization.php`.
40. P. Sandhu. *The MathML Handbook*. Charles River Media, 2002.
41. J. Siekmann and C. Benzmüller. Omega: Computer supported mathematics. In G. Palm S. Biundo, T. Frühwirth, editor, *KI 2004: Advances in Artificial Intelligence: 27th Annual German Conference on AI*, volume 3228 of *LNAI*, pages 3–28, Ulm, Germany, 2004.
42. A. Trybulec and H. A. Blair. Computer aided reasoning. In R. Parikh, editor, *Proc. of the conference Logic of Programs*, volume 193 of *LNCS*, pages 406–412. Springer, 1985.
43. S. Wolfram. *The Mathematica Book*. Wolfram Media Inc., 5th edition, 2003.

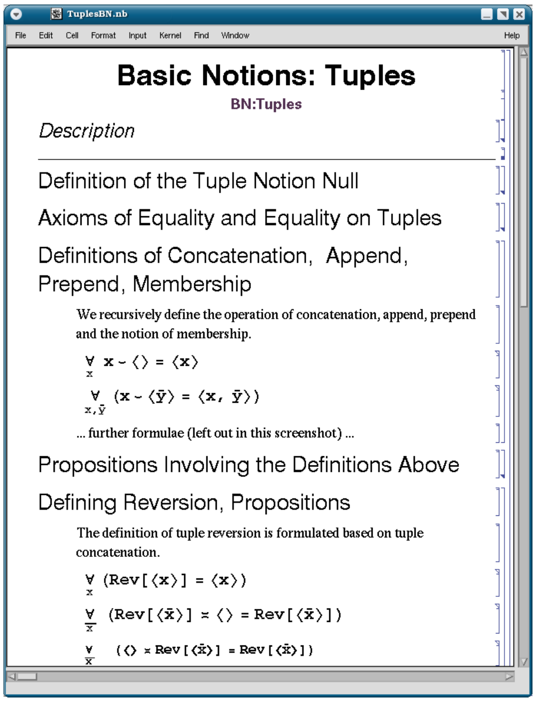

Figure 1.

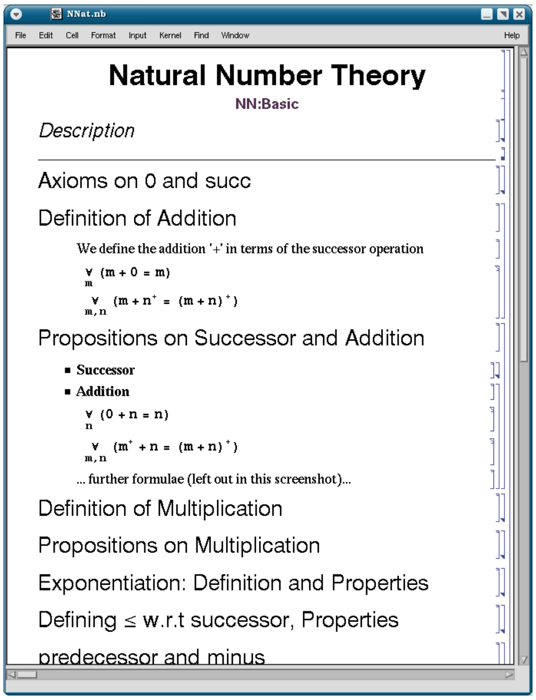

Figure 2.

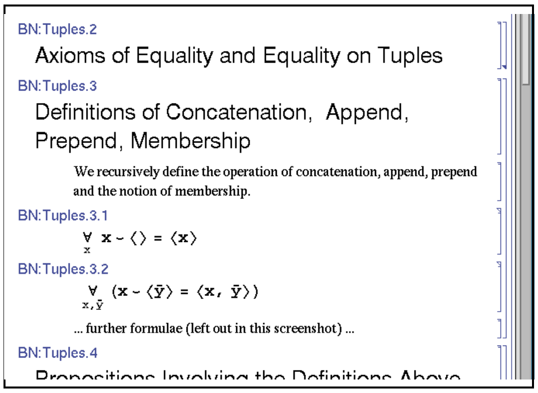

Figure 3.

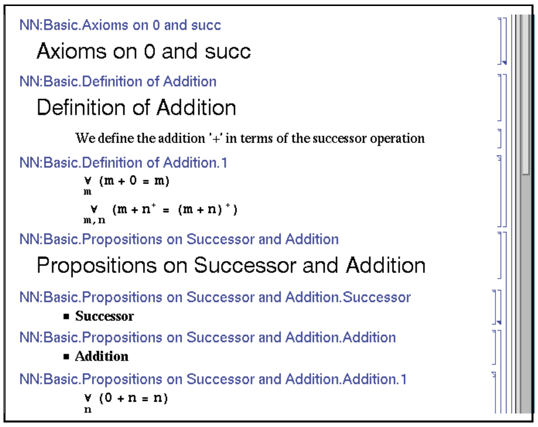

Figure 4.

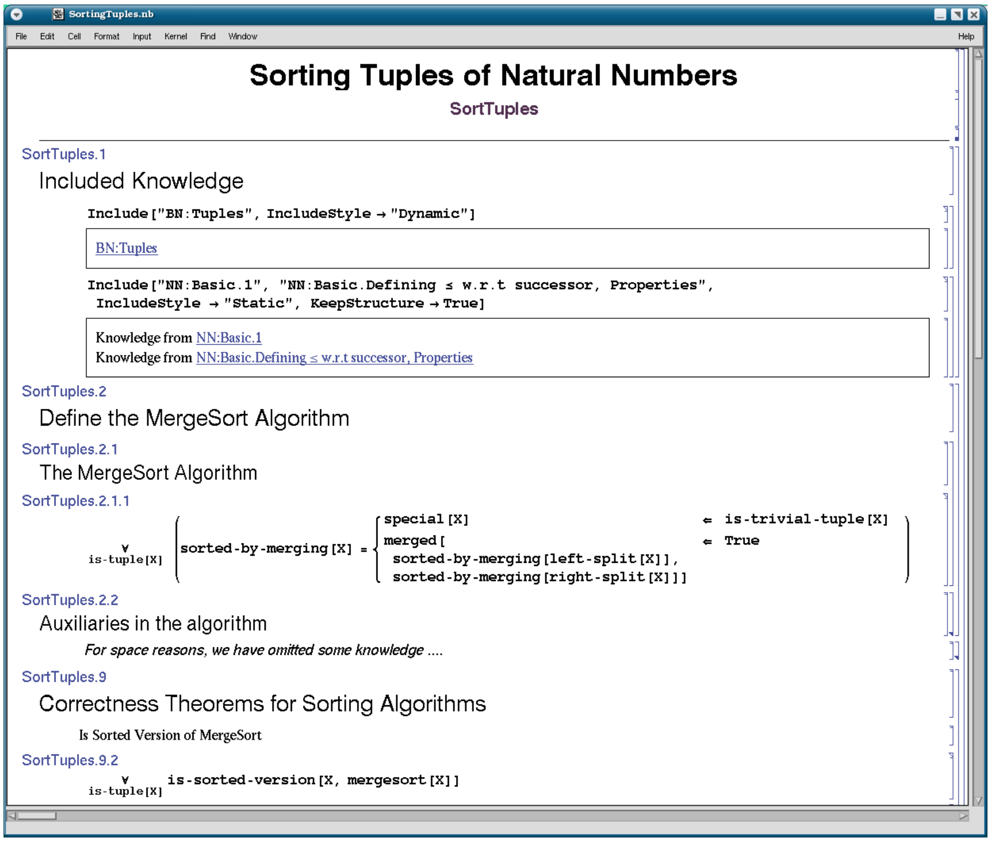

Figure 5.

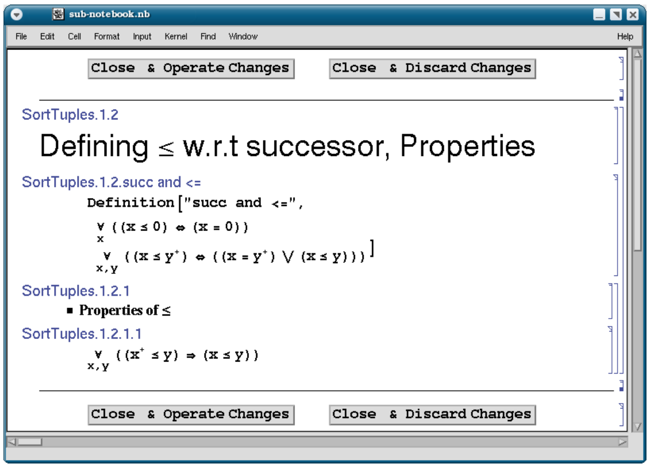

Figure 6.