Mathematical Knowledge Archives in *Theorema*

Camelia Rosenkranz, Bruno Buchberger, Tudor Jebelean

{Camelia.Rosenkranz, Bruno.Buchberger, Tudor.Jebelean}@risc.uni−linz.ac.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Schloss Hagenberg, A-4232 Hagenberg, Austria
{Camelia.Rosenkranz, Bruno.Buchberger,Tudor.Jebelean}@risc.uni−linz.ac.at

Abstract

Archives are implemented as an extension of *Theorema* for representing large bodies of mathematics. They provide various constructs for organizing knowledge bases in a natural way: breaking formulae across cells, grouping them in a hierarchical structure, attaching labels to subhierarchies, disambiguating symbols by the use of namespaces, importing symbols from other namespaces, addressing the domains of categories and functors as namespaces with variable operations. All constructs are logic-internal in the sense that they have a natural translation to higher-order logic so that 'mathematical knowledge management' can be treated by the object logic itself.

1 Introduction

An **archive** is a mathematical knowledge repository in *Theorema*. One can see it both as a *Theorema* language extension and as a user friendly interface to *Theorema*. It allows to input large collections of mathematics in style that tries to be close to that of the working mathematician: it allows to use natural notation, avoids redundancy and offers powerful tools: **labels** for building up hierarchical mathematical theories and **namespaces** for structuring concepts.

Our primary design principle was to provide these organizational constructs within predicate logic itself. Rather than using logic-external mechanisms like the label tools of [PiroiBuchberger04, Piroi04] or the metadata of [LibrechtMelis06] and [RudnickiTrybulec01], labels are therefore realized by propositional constants or terms and one refers to their contents by modus ponens. Likewise, namespaces are realized similar to the domains of categories and functors and not by logic-external content dictionaries as in [Kohlhase06].

Since archives provide a logic-internal representation of mathematics, we hope that organizational tasks on them can be achieved by the established methods of logic (e.g. modus ponens for label reference, as mentioned above). This leads us to the core of problems considered in the emerging field of mathematical knowledge management (MKM): how to build up repositories of mathematical knowledge and how to structure them from a practical perspective. We claim that archives can contribute to the main challenge of this field: to help mathematicians in their day-by-day research, where e.g. knowledge retrieval is needed for finding slightly different formulations of an existing lemma, doing simple proofs etc. In order to avoid extensive literature search, mathematicians still prefer reinventing lemmata and thus lose precious time with 'trivial' but tedious proofs that could be (at least partially) automated instead of spending days or months for finding the already existing proofs, hidden somewhere in the huge literature. We hope that this state of affairs will change in the near future and that archives will be a small step in this direction.

An idea similar in spirit, but using a proof-theoretic approach to MKM, is presented in [Hosn07] and [Hosn−Andersen05]. The library of proofs and theorems and the proof tactics are there integrated in the underlying proof logic, where scoping and tactics are represented by certain typed λ-calculus constructions. The point of emphasis there seems to be more on the migration between object and metalevels in mathematics. In *Theorema*

similar ideas have been studied from the perspective of reflection [GieseBuchberger07].

The biggest organized library of mathematics is the MML library of Mizar [RudnickiTrybulec01]. Other libraries of mathematics include HELM [Coen04] and the mathematical databases formalized in OMDoc [Kohlhase06]. The ongoing efforts for reworking the theory of special functions into an online database in the DLMF project [MillerYoussef02] are another instance of structuring a considerable portion of mathematics. Authoring tools like ActiveMath [MelisEtAl03] also contain portions of mathematics structured in a logic-external way.

The structure of the remaining paper is as follows. The present section serves as an overall motivation for digitized mathematics in general and formalized mathematics in *Theorema* in particular (Subsection 1.1). After motivating the usage of labels (Subsection 1.2), we introduce the notions of categories and functors (Subsection 1.3), and we explain the concept of namespaces (Subsection 1.4).

The notion of archive and basic commands for manipulating it are discussed in Section 2. We present various useful notational conventions for already existing *Theorema* constructs: conjunction (Subsection 2.1), universal quantifiers (Subsection 2.2), existential quantifiers (Subsection 2.3), substitution quantifiers (Subsection 2.4) and description quantifiers (Subsection 2.5).

In Section 3, we introduce labels and namespaces together with various concepts relating to them. Labels are attached to hierarchical blocks of formulae (Subsection 3.1), which may contain global and local symbols (Subsection 3.2) as well as symbols residing in a specified namespace (Subsection 3.3). Finally we work out the relations between namespaces and the domains of categories and functors (Subsection 3.4).

After a short Conclusion, we exibit a formalization example (Annex A1).

Some last remarks on terminology. We use double quotes for indicating literal references, e.g. “0” or “Algebra.nb” and single quotes for designating intuitive and non-precise notions, e.g. ‘organization’. We use the term “symbol” for the syntactic entities referring to functions, predicates or constants (i.e. “function constants”, “predicate constants” or “object constants”).

1.1 Digitized Mathematics with *Theorema*

Mathematical knowledge is increasing year by year at an exponential rate. Due to the huge, growing body of mathematical knowledge, the need for systematization, and accessibility increases with it. The necessity of structuring mathematics in a hierarchical knowledge body started and developed with mathematics itself. The axiomatic approach initiated at the beginning of the twentieth century marks a crucial step in this development; it led to fixing various important categories, e.g. the first axiomatization for rings was given in 1914 by Fraenkel [Fraenkel14] and for commutative rings in 1921 by E. Noether [Noether21]. Reals, integers, polynomials, etc. were initially studied as concrete domains, and it took some time for the abstract concepts to crystalize. The economy inherent in these axiomatized classes gave a great boost to the further development of mathematics in the twentieth century, bringing clarity and structure to its various branches.

With the advent of computers, the task of organizing mathematics took on a new face and it became more acute as the volume of mathematical knowledge increased. New techniques from computer science fields like data mining or semantic web gave valuable impulses, creating the field of MKM as mentioned above. Two crucial tasks in MKM are:

- **Mathematical knowledge retrieval**: The problem of finding information on a given notion in a given knowledge base turns out to be a highly nontrivial task. Since the essence of mathematical information lies in its logical structure rather than its linguistic appearance, textual search is not a sufficient solution. Equivalent formulae will not be discovered by a purely textual search engine. For example when one tries to prove

$$\frac{a+b}{2} \neq 0 \bigwedge c^2 + 1 \neq 0 \Rightarrow \frac{a+b}{2} * (c^2 + 1) \neq 0$$

one can retrieve the needed information if the knowledge base contains the formula:

$$\forall_x \forall_y \left((x * y = 0) \Rightarrow \bigvee \begin{Bmatrix} x = 0 \\ y = 0 \end{Bmatrix} \right)$$

But note that this retrieval needs the propositional tautology $(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$. Generally speaking, one needs inside the proof of a theorem, besides the given assumptions, also extra definitions, theorems and lemmata (perhaps occuring in the knowledge base in an equivalent form) for the proof to succeed. Nowadays this is done by adding the whole knowledge base to the proof assumptions, which is of course an overkill: calling a retrieval engine could ease and shorten the proof considerably. However, the search for such auxiliary information is still a complex task, especially when dealing with large databases. First steps in this direction are done in [AspertiEtAl07] where a complex, textual-based retrieval mechanism is called within an interactive prover called Matita. For more details about this approach, see the report [PiroiEtAl08, section on Matita]. We regard retrieval mechanisms as collections of simple inference rules used to obtain the formula needed or a semanticaly identical one by derivations on the knowledege base [Buchberger03]; we plan to present an implementation in the forthcoming report [Rosenkranz08].

- **Systematic theory exploration**: The buildup of mathematics is an involved process that can be carried out in various different ways [Buchberger99]; top−down versus bottom−up approaches can be distinguished [Buchberger04]. In the latter case, one studies systematically all interactions between the concepts occuring: Each time a new predicate or function is introduced, one investigates all interesting interactions with the old concepts. Thus one builds up a mathematical theory in layers, starting from elementary notions and axioms. In the case of a top-down approach, one starts out with a rudimentary knowledge base containing only axioms and some fundamental theorems. Trying to prove a conjecture 'higher up' in the knowledge hierarchy will typically lead to auxiliary lemmata whose proof will also guarantee the main conjecture. These lemmata produce in turn new auxiliary lemmata, generating a whole cascade of new mathematical formulae [Buchberger03a]. These can all be added to the initial knowledge base—if the recursive generation of lemmata is eventually grounded in direct proofs from the rudimentary knowledge base. In an ideal setup for generating mathematical knowledge, the top−down and the bottom−up approaches would be combined.

As mentioned above, we have created the language of archives within the *Theorema* system. Among the numerous formal frameworks nowadays available for assisting mathematicians, *Theorema* is particularly suited for supporting the entire process of mathematical work [BuchbergerEtAl06]:

- It offers two dimensional syntax for presenting logical formulae in a natural way that can easily be changed by the user without changing the internal abstract syntax. Proofs can be generated and presented in a style that comes close to that of mathematical textbooks.
- By using a variant of higher-order predicate logic, it allows to specify and use algorithms as an integral part of the language.
- Theory exploration is preferred over isolated theorem proving—which provides the right setup for integrating MKM components in the system.
- Every part of mathematics can be equipped with efficient specialized reasoners (e.g. cylindrical algebraic decomposition for elementary analysis). One would need a suitable reflection principle for embedding this approach into knowledge repositories. We do not address this issue in the present report, but see [GieseBuchberger07] for some first ideas.
- For specifying/manipulating model classes like the rings mentioned above, *Theorema* encourages the usage of so-called categories/functors (see Subsection 1.3). Note that these terms are meant in the sense

of [Buchberger08, Buchberger01] and not necessarily in the sense of Eilenberg and MacLane (although there will often be a natural correspondence).

- Apart from the newer developments described in this report, *Theorema* offers some MKM relevant features: Labels for individual formulae as well as nested environments for theories, propositions, definitions and the like with various tools for managing them [PiroiEtAl07]. One may view the language of archives as a continuation of this functionality, embedding the organizational tools into the object language.

Despite these attractive features of *Theorema*, some extensions seem to be necessary for coping with large bodies of mathematical knowledge. For example, it becomes very cumbersome that often one has to repeat essentially the same name three times: Once in the (informal) section heading, then in the label of the environ−ment, and finally in the formula label. (sometimes even a forth time, in the name of a symbol being defined or characterized). In the language of archives, we try to overcome this problem by using logic-internal labels.While this will be explained in detail later (see Subsection 3.1), let us now start with an informal exposition illustrating the basic ideas.

1.2 Organizing Mathematics in Chapters and Sections

Like most other books, mathematical textbooks are divided into chapters, sections, etc. and thus impose a kind of hierarchy on the formulae and concepts contained in them. For example a book on algebra, may contain a chapter on lattices, in turn containing a section on modular lattices, in which the concept of modularity for lattices is introduced. Hence it is natural to view the corresponding axioms as 'residing under the label' Lattices▪−ModularLattices and the symbol "is-modular" as belonging to a 'namespace associated' with this section. Using the language of archives, these informal ideas can be made precise: Labels will be introduced formally in Subsec−tion 3.1 and namespaces in Subsection 3.3.

The hierarchies connected to labels and namespaces are based on grouping formulae into nested blocks. While this idea will be made precise later (see Subsection 2.1), we try to give an intuitive understanding before−hand. In *Theorema*, the definition of a real vector space would look as follows:

Definition["RealVectorSpace", any[V],

$$\text{is-vecspace}[V] \Leftrightarrow \forall_{x,y,z\in V}\ \forall_{\lambda,\mu\in\mathbb{R}}\left(\bigwedge\left\{\begin{array}{c}\left(x +_V y\right) +_V z = x +_V \left(y +_V z\right)\\ x +_V 0_V = x\\ x +_V \left(-_V x\right) = 0_V\\ x +_V y = y +_V x\\ \lambda \cdot_V \left(x +_V y\right) = \lambda \cdot_V x +_V \lambda \cdot_V y\\ \left(\lambda +_{\mathbb{R}} \mu\right) \cdot_V x = \lambda \cdot_V x +_V \mu \cdot_V x\\ \left(\lambda *_{\mathbb{R}} \mu\right) \cdot_V x = \lambda \cdot_V \left(\mu \cdot_V x\right)\\ 1_{\mathbb{R}} \cdot_V x = x\end{array}\right.\right]$$

Underscripted operation symbols like $+_V$ are a shorthand notation for the corresponding curried versions like V[+]. The unary predicate is-vecspace is introduced for deciding whether some V is a vector space or not. Observe that such a vector space V is represented as a single object—called a **domain**—even though it contains the various constituents ϵ, +, 0, −, · ; they are 'connected' to V by underscripting.

Using blocks, this definition can be made more readable:

$$\forall_{V}$$

is-vecspace[V] $\Leftrightarrow$

$$\forall_{x,y,z \in V} \; \forall_{\lambda,\mu \in \mathbb{R}}$$

$$\left(x +_{V} y\right) +_{V} z = x +_{V} \left(y +_{V} z\right)$$

$$x +_{V} 0_{V} = x$$

$$x +_{V} \left(-_{V} x\right) = 0_{V}$$

$$x +_{V} y = y +_{V} x$$

$$\lambda \cdot_{V} \left(x +_{V} y\right) = \lambda \cdot_{V} x +_{V} \lambda \cdot_{V} y$$

$$\left(\lambda +_{\mathbb{R}} \mu\right) \cdot_{V} x = \lambda \cdot_{V} x +_{V} \mu \cdot_{V} x$$

$$\left(\lambda *_{\mathbb{R}} \mu\right) \cdot_{V} x = \lambda \cdot_{V} \left(\mu \cdot_{V} x\right)$$

$$1_{\mathbb{R}} \cdot_{V} x = x$$

Actually we can make it even a bit more readable if we open V as a namespace (see Subsection 3.4):

$$\forall_{V}$$

is-vecspace[V] $\Leftrightarrow$

$$V : \langle +, -, 0, \cdot \rangle$$

$$\forall_{x,y,z \in V} \; \forall_{\lambda,\mu \in \mathbb{R}}$$

$$(x + y) + z = x + (y + z)$$

$$x + 0 = x$$

$$x + (-x) = 0$$

$$x + y = y + x$$

$$\lambda \cdot (x + y) = \lambda \cdot x + \lambda \cdot y$$

$$\left(\lambda \underset{\mathbb{R}}{+} \mu\right) \cdot x = \lambda \cdot x + \mu \cdot x$$

$$\left(\lambda \underset{\mathbb{R}}{*} \mu\right) \cdot x = \lambda \cdot (\mu \cdot x)$$

$$\underset{\mathbb{R}}{1} \cdot x = x$$

Sometimes we not only want to group formulae into a block but also attach a name to it: This is realized by a label. For the example mentioned at the beginning of the Subsection, the hierarchy might contain the following skeleton:

Algebra $\rightleftharpoons$

LatticeTheory $\rightleftharpoons$

ModularLattices $\rightleftharpoons \underset{L}{\forall}$

is-modular[L] $\Leftrightarrow$

$L : \langle \epsilon, \sqcap, \sqcup \rangle$

$\underset{\epsilon[x,y,z]}{\forall} ((x \sqcup z = z) \Rightarrow (x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap z))$

The above example is actually taken out from Chapter XIV in [MacLaneBirkhoff67], which we have formalized as an archive (see the fragment in Annex A1). We would like to emphasize, however, that the ‘headings’ above (“Algebra”, “LatticeTheory” and “ModularLattices”) are not just logic-external ‘decorations’ but integral parts of the object language.

1.3 Categories and Functors

Mathematical theories are the building blocks of mathematics and thus also of archives. A mathematical theory is determined by a set of symbols (having a certain arity) and axioms characterizing them; the corresponding model class is known as a **category**. An example is given by the theory of real vector spaces axiomatized as in the definition given in Subsection 1.2.

There we represented the vector space as a single object, a domain; this is called the **packed** representation of the category. An alternative would be to use a quinary instead of a unary predicate:

form-vecspc[ϵ, +, 0, −, ·] $\Leftrightarrow$ …

Let us call this the **unpacked** representation of a category.

In a larger archive, one will generally avoid unpacked representations because they lead to a proliferation of symbols. For seeing this, consider computing in the matrix ring $\langle \mathbb{Z}[x]^{2\times 2}, +, *, \ldots \rangle$ the following example:

$$\begin{pmatrix} x^2-7 & 2x+4 \\ x-3 & x^2+3 \end{pmatrix} + \begin{pmatrix} x^2-3 & x-2 \\ x & x^2+4 \end{pmatrix} * \begin{pmatrix} x^3 & x^2+7 \\ 1-2x & 2x^2-3x \end{pmatrix}$$

In an unpacked representation, we would need three additions/multiplications: one for the matrix ring $\mathbb{Z}[x]^{2\times 2}$, another for the polynomial ring $\mathbb{Z}[x]$, and a third for the coefficient ring $\mathbb{Z}$. This becomes even worse if we want

to consider other coefficient rings like $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Q}(\sqrt{3})$, $\mathbb{C}$ and other constructions instead of the polynomials and matrices, e.g. power series. Allowing only one repetition of the constructions, we would already end up with 36 different symbols for addition/multiplication.

The constructions just mentioned are typical examples of **functors**. They construct new domains out of given ones. For example, the polynomial functor written informally as $\mathrm{Pol} : \mathrm{R} \mapsto \mathrm{R}[\mathrm{x}]$ sends the coefficient ring R to the ring of polynomials over R. Observe that this is already a preservation statement for the functor—it maps the category $\mathcal{Rng}$ of rings to itself, informally expressed by $\mathrm{Pol} : \mathcal{Rng} \to \mathcal{Rng}$. Such preservation properties are typical for functors F: If the input domain D satisfies a certain property P, its output domain F[D] satisfies some property Q. Viewing the properties as categories, this can also be expressed by saying that the functor F is a map between the categories P and Q. Similar things can be said about functors operating on several domains.

The usage of functors allows to realize the principle of **generic implementations**: In the above example, we do not need 36=4*3*3 different definitions ($\approx$ implementation) but only 7=4+3. The usage of such orthogonal implementations avoids code duplication and thus increases the usability of the underlying domains.

Note that in working with domains like rings, we have to distinguish between the 'whole ring' R and its 'carrier', which can either be defined by a membership predicate ϵ as in our previous examples (see Subsection 1.2) or by a carrier set. A similar distinction can be made for the other operations: They can either be realized as functions/predicate symbols of higher-order logic or as mappings/relations in the sense of set theory. While in this report, for simplicity, we will refrain from employing set theory in domains, we would like to emphasize that this is perfectly possible if desired.

Encoding domains as 'containers' for their operations as explained above goes back to [Buchberger08, Buchberger01], who refers to them as **operation objects**. We will also use this term, preferring the word "domain" only in conjunction with categories and functors, but we will see later that operation objects generalize naturally to the concept of namespaces (see Subsection 3.4). For example, the operation object corresponsing to a ring R encompasses the following components:

Operation object	R
Carrier	$\mathrm{R}[\epsilon]$
Addition	$\mathrm{R}[+]$
Zero	$\mathrm{R}[0]$
Negative	$\mathrm{R}[-]$
Multiplication	$\mathrm{R}[*]$
One	$\mathrm{R}[1]$

Operation objects allow for an elegant formulation of the preservation statements typically encountered in relation with categories and functors. Here is a simple example:

$$\text{is-group}[G] \wedge \text{is-group}[H] \Rightarrow \text{is-group}[G \times H]$$

We see again how important it is that an operation object packs all concepts into a single object: Using the unpacked representation, the formula above would look somewhat as follows:

$$\text{form-group}[\epsilon 1, +, 0, -] \wedge \text{form-group}[\epsilon 2, \oplus, \odot, \ominus] \Rightarrow \text{form-group}[\text{dir-prod-carrier}[\epsilon 1, \epsilon 2], \text{dir-prod-binary}[+, \oplus], \text{dir-prod-neutral}[0, \odot], \text{dir-prod-unary}[-, \ominus]]$$

This problem becomes more pronounced when dealing with rings or vector spaces (not to mention that one runs out of symbols for the operations needed).

Preservation theorems of the type above illustrate the other side of functors: While their computational aspect amounts to transporting algorithms (e.g. implementing the group operation of the direct product in terms of the given group operations), their proving aspect can be seen as transporting properties (e.g. the property of being a group in the example above). The resulting algorithms can of course be implemented in a programming lan−

guage (e.g. *Mathematica*), but for verifying their properties one needs a theorem prover; The *Theorema* system is an integrated environment providing both.

In *Theorema*, we can express categories and functors without extra language constructs, using only the higher-order predicate language of *Theorema*. For functors, *Theorema* offers a special notation; e.g. the direct product is expressed as follows:

Definition["DP", any[G, H],

$$G \times H = \text{Functor}[D, \text{any}[x, y],$$

$$\$ = \langle 0 : D \rangle$$

$$\epsilon[x] \Leftrightarrow \left(\text{is-tuple}[x] \bigwedge (\text{card}[x] = 2) \bigwedge_{G} \epsilon[x_1] \bigwedge_{H} \epsilon[x_2]\right)$$

$$x \underset{D}{+} y = \left\langle x_1 \underset{G}{+} y_1 , x_2 \underset{H}{+} y_2 \right\rangle$$

$$\underset{D}{0} = \left\langle \underset{G}{0}, \underset{H}{0} \right\rangle$$

$$\underset{D}{-} x = \left\langle \underset{G}{-} x_1 , \underset{H}{-} x_2 \right\rangle$$

]]

This notation does not take us out of higher-order logic; the expression "Functor[D, …]" is just an abbreviation for the description quantifier $\underset{D}{\ni}$ … yielding the desired operation object. Thus categories and functors are interpreted naturally within higher-order predicate logic, and we will show how the language of archives can support this interpretation (see Subsection 3.4).

1.4 Symbols with Multiple Meanings in Mathematics

In mathematics, some symbols are used in diffent contexts and for different purposes. Such ambiguous symbols are ubiquitos, e.g. "0", "rank", "is-normal" etc. Usually the only way to distinguish them is to look at them in their context, and use the interpretation intended by the mathematicians in a certain subfield. Of course, in the process of formalization this is a hurdle that has to be taken.

As a specific example, we consider now the symbol "0". By the legendary 'abuse of notation', it is used by the working mathematician for a variety of intuitively similar but logically distinct objects. Let us open up a typical algebra textbook, [MacLaneBirkhoff67, p. 360 or p. 276]: In the first example, the first two occurences of "0" refer to the zero module, the third to the zero element in a quotient module, the next two occurences to the natural number 0, the last before the exercises to the zero element in another module. The second example exibits "0" in two different meanings: the zero element of a field and the zero matrix in various shapes.

We distinguish two types of "0": **zero elements** (typically the neutral elements with respect to addition in various algebraic domains) and **zero domains** (typically the trivial domains of various algebraic categories). Using informal notation, here are some examples of zero elements:

- A zero vector (in a specific vector space):

$$(0, 0, 0) \in \mathbb{R}^3$$

- A zero function (in a specific function algebra):

$$(x \mapsto 0) \in C[0, 1]$$

- A zero matrix (in a specific matrix ring):

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^2 \times \mathbb{R}^2$$

- Zero knot, zero operator, etc.

The ambiguity between various zero elements can be resolved by using operation objects in a suitable category, as explained in Subsection 1.3. For the first example, we would assume a functor named VecSpc for building up finite-dimensional real vector spaces (parametrized by dimension), so VecSpc[n] denotes the vector space with carrier $\mathbb{R}^3$ and we can represent the zero element unambiguosly as:

$$\underset{\text{VecSpc}[3]}{0}$$

Also in the second example, we could take a functor named CntFnc for building up the real algebra of continuous functions on a closed interval (parametrized by its two end-points); Thus CntFnc[a, b] denotes the algebra of continuous function on [a, b], and the zero element can be represented as:

$$\underset{\text{CntFnc}[0,1]}{0}$$

In the third example, we start with a functor named MatRng for building up the matrix ring of a given dimension. Now MatRng[n] will denote the ring of $n \times n$ real matrices, with the zero element being:

$$\underset{\text{MatRng}[2]}{0}$$

In all three examples, we can also view the domains specified by the underscripts as compound namespaces constructed by the corresponding functors; this will be explained in Subsection 3.4.

On the other hand, the 'zero domains' are themselves operation objects, but with a trivial carrier (containing only one element—its zero element) and correspondingly trivial operations. Again all these zero domains would typically be written by the ubiquitous symbol "0". An example of such a domain is the zero group, realized by a canonical representation of its only element "0":

$$0 = \underset{G}{\ni} \underset{g,h}{\forall} \begin{cases} \underset{G}{\in}[g] \Leftrightarrow \left(g = \underset{G}{0}\right) \\ g \underset{G}{+} h = \underset{G}{0} \\ \underset{G}{-} g = \underset{G}{0} \end{cases}$$

Another example would be the zero space (over the reals):

$$0 = \underset{V}{\ni} \underset{v,w}{\forall} \begin{cases} \underset{V}{\in}[v] \Leftrightarrow \left(v = \underset{V}{0}\right) \\ v \underset{V}{+} w = \underset{V}{0} \\ \underset{V}{-} v = \underset{V}{0} \\ \underset{\lambda}{\forall} \lambda \underset{V}{\cdot} v = \underset{V}{0} \end{cases}$$

Of course there are numerous other examples like the zero monoid, the zero ring, the zero module, etc. In the language of archives (see Subsection 3.3), we resolve this type of ambiguity with the aid of atomic or hierarchical namespaces since each zero domain is defined uniquely in the category where it lives. So the zero group will be denoted by Groups▪0, formatted nicely as 0_{Groups}. If the theory of groups is built up in a more hierarchical fashion, the zero group might instead be written as Algebra▪GroupTheory▪Groups▪0, formatted as $0_{\text{Algebra▪GroupTheory▪Groups}}$. For details we refer to Subsection 3.4.

2 Coarse Structure of Archives

An archive is a single formula in an extension of the *Theorema* language (see Subsection 1.1). Only two new symbols are needed:

- For attaching labels we use "⇌".
- For declaring namespaces we use ":".

We will explain the usage of these two symbols in Section 3. In the present section, we discuss various notational conventions for already existing *Theorema* language constructs; they help to make large mathematical knowledge bases more readable and less redundant.

The user interface to an archive is a **structured notebook**, a *Mathematica* notebook written with the predefined "TheoremaFormalization" stylesheet. It will contain comments (represented by the cell styles "Author", "Formalizer", "Notes") and nested 'formal' cells (having the cell style "Formal X" with X a natural number from 1 to 9). The title of the notebook (having the cell style "Title") is also considered a formal cell. Of course comment cells do not influence the archive created and are meant only as a help for the reader.

In order to load an archive saved, say, under the filename "Algebra.nb" in the home directory, one can use the following command:

```
archive = LoadArchive["Algebra.nb"]
```

By this call, a file containing the box structure of the archive is parsed into a *Mathematica* expression stored in the variable "archive". The user can also specify a keyword (as a string), which is typically a label (see Subsection 3.1) occuring in the structured notebook. This will restrict parsing to the first cell group containing the keyword. As an example consider loading the 'subsection' entitled "BasicProperties" of the previous structured notebook:

```
archive = LoadArchive["Algebra.nb", "BasicProperties"]
```

Of course there is also a command "SaveArchive" for transforming a *Mathematica* expression representing an archive into its canonical box structure (typically not identical but equivalent to the original structured notebook).

Archives can be translated to plain *Theorema.* This translation involves a partial loss of structure but retains its logical content. In a sense, an archive is a logical formula plus organizational annotations: the annotations can be translated to logic but the resulting formula blurs the distinction between 'logic' and 'organization'. The translation command

```
ArchiveToList[archive]
```

returns a *Theorema* formula. In this report, we choose a slightly modified output of the translators: If the output *Theorema* formula is a conjunction, for readability reasons, we enlist only its conjuncts.

For details about how the translation takes place, see the remaining subsections in this and the following section. We view this translation as a convenient way of specifying a semantics for archives. Our ultimate goal is not the translator. On the contrary, we prefer to work directly with the archives: Exploiting their organizational annotations, we want to approach various tasks in MKM, in particular

- starting a retrieval on an archive,
- expanding an archive by a theory exploration.

But in this report we discuss only the statical aspects of an archive. For the dynamical aspects, we refer to our forthcoming report [Rosenkranz08].

2.1 Arranging Formulae in Blocks

In our archive language we assert the conjunction of several mathematical formulae by using the 'normal' *Theorema* $\wedge$ or by using blocks: A **block** is an "indentation level" in a hierarchy of nested (and hence indented) cells, denoting the conjunction of its parts. Thus a block consists of one or more formulae, possibly in turn containing other blocks (e.g. the scope of a quantifier—see the remaining subsections of this section). An equiva−lence or implication with a conjunction on its right-hand side can be broken after the "$\Leftrightarrow$" or "$\Rightarrow$" with the right hand side following as an indented block, as in the following example:

is-equivalence[D, R] $\Leftrightarrow$

  is-reflexive[D, R]

  is-symmetric[D, R]

  is-transitive[D, R]

Here we assumed for simplicity the domain "D" and the relation "R" to be constants.

The internal representation of blocks is realized by the "™Conjunction" connective. There is no semantic difference between the *Theorema* "$\wedge$" symbol and "™Conjunction", but for pragmatic reasons we decided to distinguish between the two. This distinction is comparable to the sequent calculus, where the point is to distin−guish between $\phi_1, \phi_2, \ldots, \phi_n \vdash \phi$ and $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n \vdash \phi$, but note that "™Conjunction" represents a nested rather than a flat list of formulae. Just as sequents can be exploited for building more efficient provers, we have the hope that the distinction between blocks and normal conjunction can be useful in a similar vein.

Blocks are very economic if the formulae are preceded by common quantifiers:

$$\forall_{\circ}\, \forall_{D} \left(\text{is-idempotent}[D, \circ] \Leftrightarrow \forall_{x \in D} (x \circ x = x) \right)$$

$$\forall_{\circ}\, \forall_{D} \left(\text{is-associative}[D, \circ] \rightleftharpoons \forall_{x,y,z \in D} ((x \circ y) \circ z = x \circ (y \circ z)) \right)$$

$$\forall_{\circ}\, \forall_{D} \left(\text{is-commutative}[D, \circ] \rightleftharpoons \forall_{x,y \in D} (x \circ y = y \circ x) \right)$$

In this case, one can pull out the common quantifiers, as explained in the next subsections.

2.2 The Universal Quantifier

The previous example can also be written in the following more readable way:

$$\underset{\circ}{\forall}\ \underset{D}{\forall}$$

$$\text{is-idempotent}[D, \circ] \Leftrightarrow \underset{x\in D}{\forall} (x\circ x = x)$$

$$\text{is-associative}[D, \circ] \rightleftharpoons \underset{x,y,z\in D}{\forall} ((x\circ y)\circ z = x\circ(y\circ z))$$

$$\text{is-commutative}[D, \circ] \rightleftharpoons \underset{x,y\in D}{\forall} (x\circ y = y\circ x)$$

Here the universal quantifier is the ‘normal’ universal quantifier from predicate logic, so that the translation to *Theorema* is straightforward and can be omitted here.

Seen from the MKM viewpoint, it is interesting to point out another side of the universal quantifier that has more of a ‘programming flavor’: It realizes the idea of **parametrization** in computer science. Consider the following definition of the ‘type’ of integer lists:

$$\underset{x,\overline{x}}{\forall}\ \underset{A}{\forall}$$

$$\text{is-list}[\langle x, \overline{x}\rangle] \Leftrightarrow ((x \in \mathbb{Z}) \wedge \text{is-list}[\langle \overline{x}\rangle])$$

$$\text{is-list}[\langle\rangle] \Leftrightarrow \text{True}$$

$$\underset{\overline{a}}{\nexists} (A = \langle \overline{a}\rangle) \Rightarrow (\text{is-list}[A] \Leftrightarrow \text{False})$$

Parametrizing the element type (changing the constant “$\mathbb{Z}$” to the variable “Z”), we obtain the type of polymor−phic lists:

$$\underset{Z}{\forall}\ \underset{x,\overline{x}}{\forall}\ \underset{A}{\forall}$$

$$\text{is-list}[Z][\langle x, \overline{x}\rangle] \Leftrightarrow ((x \in Z) \wedge \text{is-list}[Z][\langle \overline{x}\rangle])$$

$$\text{is-list}[Z][\langle\rangle] \Leftrightarrow \text{True}$$

$$\neg\ \underset{\overline{a}}{\exists} (A = \langle \overline{a}\rangle) \Rightarrow (\text{is-list}[Z][A] \Leftrightarrow \text{False})$$

Note that this could now be made into a functor that constructs the type of Z-lists out of a given element type Z. (For details about defining functors in archives see Subsection 3.4.) The idea of parametrization (‘making constants into variables’), usually wrapped into corresponding functors, is applied frequently also in mathemat−ics: For example, when the construction of $\mathbb{Q}$ from the integers $\mathbb{Z}$ is parametrized to arbitrary integral rings, one obtains the quotient field functor.

2.3 The Existential Quantifier

Existential quantifiers can be used in archives just as universal ones, and one can also combine them into a quantifier prefix before a block of formulae. The following example is taken from projective geometry:

$$\underset{\substack{\text{is-point}[p,q]\\ p\neq q}}{\forall}\ \underset{\text{is-line}[l]}{\exists}$$

is-incident[p, l]

is-incident[q, l]

$$\underset{\text{is-line}[m]}{\forall}\ (\text{is-incident}[p, m] \wedge \text{is-incident}[q, m] \Rightarrow (m = l))$$

$$\underset{\substack{\text{is-line}[l,m]\\ l\neq m}}{\forall}\ \underset{\text{is-point}[p]}{\exists}$$

is-incident[p, l]

is-incident[p, m]

$$\underset{\text{is-point}[q]}{\forall}\ (\text{is-incident}[q, l] \wedge \text{is-incident}[q, m] \Rightarrow (q = p))$$

Note that in *Theorema* ranges can be described by unary predicates, like is-point and is-line in the above exam−ple; thus is-point[p,q] actually means is-point[p]∧is-point[q]. Just as for the universal quantifier, we would like to point to a role of the existential quantifier in programming: It realizes the idea of **modularization** in computer science. The following formula is a functional formulation of the quicksort algorithm from [AhoEtAl75, p.94]:

$$\underset{\text{left,right,pivot}}{\exists}$$

quick-sort[⟨⟩] = ⟨⟩

$$\underset{\text{is-non-empty-tuple}[X]}{\forall}\ (\text{quick-sort}[X] := \text{quick-sort}[\text{left}[X]] \asymp \langle \text{pivot}[X] \rangle \asymp \text{quick-sort}[\text{right}[X]])$$

$$\underset{\text{is-tuple}[X]}{\forall}$$

$$\underset{l,r}{\forall}\ ((l \in \text{left}[X] \wedge r \in \text{right}[X]) \Rightarrow l \leq \text{pivot}[X] \leq r)$$

$$(\text{left}[X] \asymp \langle \text{pivot}[X] \rangle \asymp \text{right}[X]) \approx X$$

$$|\text{left}[X]| < |X| \wedge |\text{right}[X]| < |X|$$

Informally, this passage could be formulated thus: ‘Quicksort first selects a pivot element and then splits the input list into a left and right part, calls the algorithm recursively on these parts and concatenates their outputs with the pivot in between; the algorithms for splitting into left and right parts may be chosen arbitrarily as long as all left elements preced all right elements, no elements are lost and the splits are smaller.’ In other words, the algorithms “left”, “pivot”, “right” are considered like local subroutines constrained by a suitable specification.

As one can see from this example, existential quantifiers can be used to introduce ‘local symbols’ in the sense of Subsection 3.2, i.e. they are a means of avoiding name clashes.

2.4 The Substitution Quantifier

As for the existential and universal quantifiers, the language of archives provides a multi-line variant of the substitution quantifier ↤, typically verbalised ‘let’ in prefix and ‘where’ in postfix usage. It is easiest to first consider an example (claiming correctness of Cardano’s formula):

$$\forall_{a,b,c} \; (x^3 + a * x^2 + b * x + c = 0)$$

$$\mapsfrom$$

$$x = -\frac{p}{3 * u} + u - \frac{a}{3}$$

$$\mapsfrom$$

$$p = b - \frac{a}{3}$$

$$q = c + \frac{2 * a^3 - 9 * a * b}{27}$$

$$u = \sqrt[3]{-\frac{q}{2} \pm \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}}$$

As can be seen from this example, substitution quantifiers can be nested into each other. They are used for avoiding multiple occurences of large terms, and/or for ease of readability, e.g. avoiding a large term in an index position.

Like in *Theorema*, the substitution quantifier can be used for an arbitrary expression Λ, that is either a term or a formula. The general form is as follows:

$$\Lambda$$

$$\mapsfrom$$

$$x_1 = \tau_1$$

$$\ldots$$

$$x_n = \tau_n$$

This corresponds to the plain *Theorema* formula where$[x_1 = \tau_1, \ldots, x_n = \tau_n, \Lambda]$, which evaluates to $\forall_{x_1 = \tau_1} \ldots \forall_{x_{1\,n} = \tau_n} \Lambda$.

2.5 The Description Quantifier

It often happens in mathematics that one cannot define a notion explicitly, so one has to fall back to an implicit definition. (Despite their obvious importance in applications, we do not address questions of existence and

uniqueness here.) The description quantifier provides a means of making such implicit definitions look like explicit ones, as in the following standard *Theorema* example:

Definition["Minimum", any[D], min[D] = $\mathop{\ni}_{x\in D} \mathop{\forall}_{y\in D} (y \neq x \Rightarrow y > x)$]

The intuitive meaning of the description $\mathop{\ni}_{x} \phi$ is some object satisfying the condition expressed by ϕ. If there are more such objects, any one of them is selected; if there are no such objects, an arbitrary element of the universe is chosen. The treatment of descriptions in proofs can be similar to that in HOL-Light [Harrison06, p. 92]; for more on semantical issues we refer to [Giese98, Chapter 5].
The language of archive provides a multi-line notation for this quantifier, analogous to the notation for universal and existential quantifiers (but now the resulting expression is a term rather than a formula). So the above example can be written as follows:

$\mathop{\forall}_{D}$

min[D] =

$\mathop{\ni}_{x\in D}$

$\mathop{\forall}_{y\in D} (y \neq x \Rightarrow x < y)$

It translates to the plain *Theorema* Definition above.

3 Refining Structure by Labels and Namespaces

3.1 Labelling Blocks of Formulae

As mentioned in Subsection 1.2, it is often useful to group formulae into various ‘Chapters’ and ‘Sections’. In the language of archives, one can achieve this by using atomic and hierarchical labels. **Labels** are special formu−lae asociated with blocks of formulae via the “⇌” symbol. In the example below, “BinaryRelations”, “BasicProp−erties” and “CompoundProperties” are labels:

BinaryRelations ⇌

BasicProperties ⇌ $\mathop{\forall}_{R}\mathop{\forall}_{D}$

is-reflexive[D, R] ⇔ $\mathop{\forall}_{x\in D}$ R[x, x]

is-symmetric[D, R] ⇔ $\mathop{\forall}_{x,y\in D}$ (R[x, y] ⇒ R[y, x])

…

CompoundProperties ⇌ $\mathop{\forall}_{R}\mathop{\forall}_{D}$

is-partial-order[D, R] ⇔

is-reflexive[D, R]

is-antisymmetric[D, R]

is-transitive[D, R]

is-quasi-order[D, R] ⇔ (is-reflexive[D, R] ∧ is-transitive[D, R])

…

…

Cell groups having an "⇌" in their first cell are called **packages**. The first cell in a package is its **head**, the remaining cells make up its **body.** A package body is always a block in the sense of Subsection 2.1. We distin−guish two types of labels. **Atomic** labels are propositional constants used for referring to blocks of formulae. **Hierarchical** labels serve the same purpose but are constructed as compound expressions built from several propositional constants.

Here is an example of atomic labels and their associated formulae taken from one of our formalized notebooks:

Algebra ⇌

GroupTheory ⇌

Magmas ⇌ $\underset{M}{\forall}$

is-magma[M] ⇔

$\underset{\in_M[x,y]}{\forall} \in_M\left[x \circ_M y\right]$

Semigroups ⇌ $\underset{S}{\forall}$

is-semigroup[S] ⇔

is-magma[S]

$\underset{\in_S[x,y,z]}{\forall} \left(\left(x \circ_S y\right) \circ_S z = x \circ_S \left(y \circ_S z\right)\right)$

…

LatticeTheory ⇌

Posets ⇌

…

Chains ⇌

…

This package of formulae is translated into plain *Theorema* as follows:

$$\text{Algebra} \Leftrightarrow \text{GroupTheory} \wedge \text{LatticeTheory}$$

$$\text{GroupTheory} \Leftrightarrow \text{Magmas} \wedge \text{Semigroups} \wedge \ldots$$

$$\text{Magmas} \Leftrightarrow \underset{M}{\forall}\left(\text{is-magma}[M] \Leftrightarrow \underset{\underset{M}{\in}[x,y]}{\forall} \underset{M}{\in}\left[x \underset{M}{\circ} y\right]\right)$$

$$\text{Semigroups} \Leftrightarrow \underset{S}{\forall}\left(\text{is-semigroup}[S] \Leftrightarrow \left(\text{is-magma}[S] \bigwedge \underset{\underset{S}{\in}[x,y,z]}{\forall} \left(\left(x \underset{S}{\circ} y\right) \underset{S}{\circ} z = x \underset{S}{\circ}\left(y \underset{S}{\circ} z\right)\right)\right)\right)$$

$$\text{LatticeTheory} \Leftrightarrow \text{Posets} \wedge \text{Chains} \wedge \ldots$$

$$\text{Posets} \Leftrightarrow \ldots$$

$$\text{Chains} \Leftrightarrow \ldots$$

In the example above, it was sufficient to refer to “Semigroups”, “Posets” etc. because it is clear that there are no other packages with these heads. Therefore it was appropriate to use an atomic rather than a hierarchical label.

In other cases, a disambiguation is necessary. For example if the user wants to preserve a similar structure in all her packages: She wants to investigate algebra, distinguishing between the basic and advanced theory. For each important notion, she wants to write definitions and theorems under a head with the ad-hoc name “Definitions” and “Theorems”, respectively. A fragment of such an archive looks like this:

$$\text{Posets} \rightleftharpoons \underset{P}{\forall}$$

$$\blacksquare\text{Basics} \rightleftharpoons$$

$$\blacksquare\text{Definitions} \rightleftharpoons$$

$$\text{is-poset}[P] \Leftrightarrow$$

$$\underset{\underset{P}{\in}[x,y,z]}{\forall}$$

$$x \underset{P}{\leq} x$$

$$\left(x \underset{P}{\leq} y \bigwedge y \underset{P}{\leq} x\right) \Rightarrow x = y$$

$$\left(x \underset{P}{\leq} y \bigwedge y \underset{P}{\leq} z\right) \Rightarrow x \underset{P}{\leq} z$$

$$\underset{x,y}{\forall}$$

$$\underset{\text{converse}[P]}{\in}[x] \Leftrightarrow \underset{P}{\in}[x]$$

$$x \underset{converse[P]}{\leq} y \Leftrightarrow y \underset{P}{\leq} x$$

…

▪Theorems ⇌

is-poset[P] ⇒ is-poset[converse[P]]

…

▪Advanced ⇌

▪Definitions ⇌ $\underset{C}{\forall}$

$$is\text{-}conn\text{-}chain[C, P] \Leftrightarrow \left(is\text{-}tuple[C] \bigwedge_{i=1,\ldots,|C|} \underset{P}{\in}[C_i] \bigwedge_{i=1,\ldots,|C|-1} covers\left[\underset{P}{\leq}, C_i, C_{i+1}\right]\right)$$

$$endpoint[C] = C_{|C|}$$

▪Theorems ⇌

is-lattice[P] ∧ is-finite[P] ⇒

$$\underset{C,D}{\forall} ((is\text{-}conn\text{-}chain[C, P] \wedge is\text{-}conn\text{-}chain[D, P] \wedge (endpoint[C] = endpoint[D])) \Rightarrow (|C| = |D|))$$

…

Groups ⇌

▪Basics ⇌

▪Definitions ⇌ …

▪Theorems ⇌ …

▪Advanced ⇌

▪Definitions ⇌ …

▪Theorems ⇌ …

In this example, it is obvious, that by using only atomic labels, it would not be clear e.g. which “Definitions” are meant, so the usage of hierarchical labels is appropriate in this case: They take into account the hierarchy of labels in packages including the current one. Using the archive above, one would refer to the advanced theorems of poset theory by the hierarchical label “Posets▪Advanced▪Theorems”. A translation to *Theorema* is as follows:

Posets ⇔ Posets▪Basics ∧ Posets▪Advanced

Posets▪Basics ⇔ Posets▪Basics▪Definitions ∧ Posets▪Basics▪Theorems

Posets▪Basics▪Definitions ⇔

$$\forall_{P}\left(\left(\text{is-poset}[P] \Leftrightarrow \forall_{\epsilon_{P}[x,y,z]}\left(x \leq_{P} x \bigwedge\left(\left(x \leq_{P} y \bigwedge y \leq_{P} x\right) \Rightarrow (x = y)\right) \bigwedge\left(\left(x \leq_{P} y \bigwedge y \leq_{P} z\right) \Rightarrow x \leq_{P} z\right)\right)\right) \bigwedge \right.$$

$$\left. \forall_{x,y}\left(\epsilon_{\text{converse}[P]}[x] \Leftrightarrow \epsilon_{P}[x] \bigwedge x \leq_{\text{converse}[P]} y \Leftrightarrow y \leq_{P} x\right) \bigwedge \ldots\right)$$

Posets▪Basics▪Theorems ⇔ $\forall_{P}$ (is-poset[P] ⇒ is-poset[converse[P]] ∧ …)

Posets▪Advanced ⇔ Posets▪Advanced▪Definitions ∧ Posets▪Advanced▪Theorems

Posets▪Advanced▪Definitions ⇔

$$\forall_{P}\forall_{C}\left(\text{is-conn-chain}[C, P] \Leftrightarrow \left(\text{is-tuple}[C] \bigwedge \forall_{i=1,\ldots,|C|} \epsilon_{P}[C_i] \bigwedge \forall_{i=1,\ldots,|C|-1} \text{covers}\left[\leq_{P}, C_i, C_{i+1}\right]\right) \bigwedge \right.$$

$$\left. (\text{endpoint}[C] = C_{|C|}) \bigwedge \ldots\right)$$

Posets▪Advanced▪Theorems ⇌

$$\forall_{P}\left(\left(\text{is-lattice}[P] \wedge \text{is-finite}[P] \Rightarrow \forall_{C,D}\left((\text{is-conn-chain}[C, P] \wedge \text{is-conn-chain}[D, P] \wedge \right.\right.\right.$$

$$\left.\left.\left.(\text{endpoint}[C] = \text{endpoint}[D])\right) \Rightarrow (|C| = |D|)\right)\right) \bigwedge \ldots\right)$$

Groups ⇔ Groups▪Basics ∧ Groups▪Advanced

Groups▪Basics ⇔ Groups▪Basics▪Definitions ∧ Groups▪Basics▪Theorems

Groups▪Basics▪Definitions ⇔ …

Groups▪Basics▪Theorems ⇔ …

Groups▪Advanced ⇔ Groups▪Advanced▪Definitions ∧ Groups▪Advanced▪Theorems

Groups▪Advanced▪Definitions ⇔ …

Groups▪Advanced▪Theorems ⇔ …

Note that if one uses an atomic label inside a hierarchy of labels, the atomic one will be treated as if the surrounding hierarchy were not present. An atomic label will be considered in building up the names for the hierarchical labels underneath (e.g. “Posets” or “Groups” in the example above).

A final remark about combining quantifiers with labels. As explained in Subsection 2.2, quantifiers may be prefixed to blocks; this remains true for those blocks that form the body of a package. An instance of this usage can be seen in the previous example after the label “Posets▪Advanced▪Definitions” and similar places. Quantifiers appearing in package heads higher up in the hierarchy are distributed to all formulae in the blocks underneath; this is what happened to the quantifier on “P” in the package labelled “Posets”.

3.2 Global and Local Symbols

As we saw in Subsection 1.4, we often have ambiguous symbols in mathematics. In the language of archives, we offer two ways of resolving ambiguities. Altogether, there are three types of symbols:

- **Global symbols:** These are practical for symbols that are highly important in the whole of mathematics so that one does not want to refer to them via any prefixed namespace (see the example below). Of course such symbols should be used with care, since they bring up the danger of name clashes.
- **Local symbols**: Such symbols are visible only in the subhierarchy of blocks below their point of introduction, hence they avoid name clashes with symbols used in a parallel block. But they can be used only if they are not needed outside the block where they are introduced.
- **Symbols in a namespace**: Another strategy of disambiguation proceeds by prefixing symbols with a 'path' providing the necessary context information. Due to the analogy with certain programming languages we call such a path a namespace (see the next subsection). Of course, the price for this disambiguation is that one has to specify the namespace explicitly when referring to such a symbol; we will provide a shortcut notation for that purpose.

As we have already seen in Subsection 2.3, local symbols can be simulated in our chosen language by the aid of an existential quantifier. Furthermore, we can add a hierarchy of labels for referring to a block containing a local symbol (omitting the passage from before for saving space):

Tuples ⇌

Sorting ⇌

BubbleSort ⇌ …

MergeSort ⇌ …

QuickSort ⇌

$\exists_{\text{left,right}}$

… [see Subsection 2.3]

PartialCorrectness ⇌ $\forall_{\text{is-tuple[X]}}$

is-sorted[quick-sort[X]]

quick-sort[X] ≈ X

In this example, the names "left" and "right" for the auxiliary algorithms used in the package "QuickSort" could also occur in "BubbleSort" and "MergeSort" with different meanings (different properties).

Despite the usefulness of local or namespace-bound symbols, some symbols need to be global: While they cannot be local because they are needed everywhere, one also does not want to clutter the archive with countless occurences of the same namespace. Typical examples are the basic notions of set theory, for example $\in$, $\subseteq$, $\emptyset$. In ZFC set theory, these symbols could be introduced as follows:

ZFC ⇌

▪Axioms ⇌

Extensionality ⇌

$$\underset{x,y}{\forall}\left(\underset{z}{\forall}\left((z \in x) \Leftrightarrow (z \in y)\right) \Rightarrow (x = y)\right)$$

…

Regularity ⇌

$$\underset{x}{\forall}\left(\underset{y}{\exists}\, y \in x \Rightarrow \underset{y}{\exists}\left(y \in x \bigwedge \underset{z}{\nexists}\,(z \in x \wedge z \in y)\right)\right)$$

▪Definitions ⇌

Subset ⇌

$$\underset{x}{\forall}\,(x \subseteq y) \Leftrightarrow \underset{z}{\forall}\,(z \in x \Rightarrow z \in y)$$

EmptySet ⇌

$$\underset{x}{\forall}\,(x \notin \emptyset)$$

…

Arguably, also the natural numbers can be considered global symbols. They are used very often in mathematics, e.g. as indices of sequences, dimensions of vector spaces or degrees of polynomials. But of course this is ultimately a question of one's goal and taste.

3.3 Symbols Bound to a Namespace

As explained in the previous subsection, there is a need of disambiguating mathematical symbols. In many cases, we need certain symbols in many places throughout the archive, so we cannot use local ones. Since we should also be cautios not to use too many global symbols, we will often use the third type of symbols, those bound to a **namespace**. This is realized in a fashion analogous to the operation objects of Subsection 1.3: Namespaces are function symbols wrapping symbols in order to distinguish their different meanings. For example, the package

BinRel : ⟨is-transitive⟩ ⇌

$$\underset{\sim}{\forall}\,\text{is-transitive}[\sim] \Leftrightarrow \underset{x,y,z}{\forall}\,(x \sim y \wedge y \sim z \Rightarrow x \sim z)$$

…

is translated to

$$\text{BinRel} \Leftrightarrow \left(\underset{\sim}{\forall}\left(\underset{\text{BinRel}}{\text{is-transitive}}[\sim] \Leftrightarrow \underset{x,y,z}{\forall}\,(x \sim y \wedge y \sim z \Rightarrow x \sim z)\right) \bigwedge \ldots\right)$$

so that "is-transitive" will be distinguished from other occurences of the symbol "is-transitive", e.g. the follow−
$\underset{\text{BinRel}}{}$ing set theoretic notion:

SetTh : ⟨is-transitive⟩ ⇌

$\underset{z}{\forall}$ is-transitive[z] ⇔ $\underset{x,y}{\forall}$ (x ∈ y ∧ y ∈ z ⇒ x ∈ z)

which is of course translated to

$$\text{SetTh} \Leftrightarrow \underset{z}{\forall}\left(\underset{\text{SetTh}}{\text{is-transitive}}[z] \Leftrightarrow \underset{x,y}{\forall}\ (x \in y \wedge y \in z \Rightarrow x \in z)\right)$$

In the above example, we have employed the namespaces "BinRel" and "SetTh", in both cases on the symbol "is-transitive". Note that the identifiers "BinRel" and "SetTh" are interpreted as labels as well as namespaces (thus we have overloaded these identifiers—see below for some comments on this issue).

In general, we can bind a sequence of symbols σ_1, …, σ_n to the namespace associated with a label L, its so-called **home namespace,** in a block of formulae:

L : ⟨σ_1, …σ_n⟩ ⇌

ϕ_1

…

ϕ_m

This is translated to *Theorema* as $L \Leftrightarrow (\phi_1 \wedge \ldots \wedge \phi_m)_{\sigma_1 \leftarrow L[\sigma_1],\ldots,\sigma_n \leftarrow L[\sigma_n]}$. In other words, the block is as always interpreted as a conjunction, but with the specified symbols being replaced by their 'wrapped' correlates L[σ_1], …, L[σ_n]; we then say that σ_1, …, σ_n are bound to the namespace L. Such packages will be called **wrapped**, as opposed to the **plain** ones of Subsection 3.1.

In order to refer to a symbol from a **foreign namespace** (i.e. a namespace different from the current home namespace), one normally would have to use its full name (symbol with the namespace underneath). For improv−ing readability, this can be abbreviated by 'opening' the foreign namespace for 'importing' the needed symbols. Imagine one builds up the theory of real numbers (with ℝ being the universe) and wants to state that certain relations are transitive in the sense defined above. In this case one could write:

Reals : ⟨…, sin, cos⟩ ⇌

…

$\underset{x}{\forall}$ ($\sin[x]^2 + \cos[x]^2 = 1$)

…

RealRelations ⇌ $\underset{x}{\forall}$

BinRel : ⟨is-transitive⟩

is-transitive[<]

$$\underset{a,b}{\forall}\ (a \sqsubset b \Leftrightarrow |a - x| < |b - x|) \Rightarrow \text{is-transitive}[\ \sqsubset\ ]$$

The general situation is as follows: The declaration $N : \langle \sigma_1, \ldots \sigma_n \rangle$ has the effect that each formula ϕ within its block is translated to $\phi_{\sigma_1 \leftarrow N[\sigma_1], \ldots, \sigma_n \leftarrow N[\sigma_n]}$. We note that the notation for wrapped packages

$L : \langle \sigma_1, \ldots \sigma_n \rangle \rightleftharpoons$

ϕ_1

$\ldots$

ϕ_m

is actually a shortcut for the following plain package combined with a namespace declaration:

$L \rightleftharpoons$

$L : \langle \sigma_1, \ldots \sigma_n \rangle$

ϕ_1

$\ldots$

ϕ_m

So home namespaces and foreign namespaces are not distinguished from the viewpoint of (pure) logic, but the distinction may still be very useful for formula retrieval and related MKM tasks. We will have to say more about this in our forthcoming report [RosenkranzEtAL08].

As mentioned above, namespaces are realized in a similar fashion as the operation objects of categories and functors. In fact, **operation objects** are identical to namespaces from a semantical point of view. The difference is more of a psychological nature: Operation objects are typically conceived as the domains residing in a certain category or constructed by a certain functor. As an example consider the following formula defining the cate−gory of semigroups:

$\underset{S}{\forall}$

Semigroup[S] $\Leftrightarrow$

$S : \langle \epsilon, * \rangle$

$\underset{x,y}{\forall}$

$\epsilon[x * y]$

$(x * y) * z = x * (y * z)$

Using the same mechanism as explained above, this is translated to:

$$\underset{S}{\forall}\left(\text{Semigroup}[S] \Leftrightarrow \underset{x,y}{\forall}\left(\underset{S}{\epsilon}\left[x \underset{S}{*} y\right] \bigwedge \left(x \underset{S}{*} y\right) \underset{S}{*} z = x \underset{S}{*} \left(y \underset{S}{*} z\right)\right)\right)$$

But observe that here we have used a variable rather than a constant for opening a namespace: The symbols ϵ and $*$ are bound to the variable namespace S.

If one uses hierarchical labels (see Subsection 3.1), it is very natural to build up a parallel **hierarchy of namespaces** for binding the miscellaneous symbols introduced in them. In fact, this is what happens automatically since the names for the home namespaces always coincide with the corresponding labels—no matter whether they are atomic and hierarchical. The above fragment from the theory of relations could naturally occur inside a surrounding hierarchy:

Algebra ⇌

▪Relations ⇌

▪UnRel : ⟨…⟩ ⇌ …

▪BinRel : ⟨is-transitive⟩ ⇌

$\underset{\sim}{\forall}$ is-transitive[~] ⇔ $\underset{x,y,z}{\forall}$ (x ~ y ∧ y ~ z ⇒ x ~ z)

…

The translation to *Theorema* reads as follows:

Algebra ⇔ (Algebra▪Relations ∧ …)

Algebra▪Relations ⇔ (Algebra▪Relations▪UnRel ∧ Algebra▪Relations▪BinRel ∧ …)

Algebra▪Relations▪UnRel ⇔ …

$$\text{Algebra▪Relations▪BinRel} \Leftrightarrow \left(\underset{\sim}{\forall}\left(\underset{\text{Algebra▪Relations▪BinRel}}{\text{is-transitive}}[\sim] \Leftrightarrow \underset{x,y,z}{\forall}(x \sim y \wedge y \sim z \Rightarrow x \sim z)\right)\bigwedge \ldots\right)$$

Finally, we would like to remark that hierarchical labels are internally realized by namespaces: A nested package like

L ⇌

▪M ⇌

ϕ

is regarded as a shortcut for:

L : ⟨M⟩ ⇌

M ⇌

ϕ

Observe that its translation to *Theorema* will be $\left(\text{L} \Leftrightarrow \underset{\text{L}}{\text{M}}\right)\bigwedge\left(\underset{\text{L}}{\text{M}} \Leftrightarrow \phi\right)$. As explained in Subsection 1.3, the notation $\underset{\text{L}}{\text{M}}$ stands for the internal representation L[M]; this is also the actual meaning of the notation L▪M. In

general, a hierarchical label L1▪L2▪…▪Ln is internally represented as L1[L2]…[Ln]. As noted above, we over−load the identifiers used for labels and namespaces: If L in the above example is used for binding symbols, it will occur with three different meanings that could be resolved by considering their types—Bool as a label, Bool→Bool as a 'wrapper' around the label M, and Bool→T as a namespace for binding a symbol of type T. Since these ambiguities do not create any problems in dealing with archives, we will not develop this issue any further.

3.4 Defining Categories and Functors via Namespaces

As already mentioned in Subsection 1.3, namespaces also provide a handy notation for domains, functors and categories; here a domain is seen as a special case of a wrapped package, e.g. defining the (multiplicatively written) semigroup of naturals in terms of the global symbols $\mathbb{N}$, + and $\in$ can be realized thus (note the differ−ence between $\in$ and ϵ):

$$\text{NaturalSemigroup} : \langle \epsilon,\ * \rangle \rightleftharpoons \underset{x,y}{\forall}$$

$$\epsilon[x] \Rightarrow x \in \mathbb{N}$$

$$x * y = x + y$$

Such definitions are also called **introduction functors** since they introduce a domain without other domains as arguments (as opposed to 'normal' functors like the direct product defined below). Note however that introduc−tion functors may have **parameters**, i.e. arguments that do not represent domains; an example would be the n-dimensional real vector spaces (where $n \in \mathbb{N}$ is a parameter).

A more degenerate example is given by the zero group mentioned in Subsection 1.4 (note the difference between O and 0):

$$\text{GroupTheory} : \langle O \rangle \rightleftharpoons$$

$$\underset{g,h}{\forall}$$

$$\underset{O}{\epsilon}[g] \Leftrightarrow \left(g = \underset{O}{0}\right)$$

$$g \underset{O}{+} h = \underset{O}{0}$$

$$\underset{O}{-} g = \underset{O}{0}$$

Note that this construction is trivially algorithmic since everything reduces to the canonical form $\underset{O}{0}$.

A very common example of a **bivariate functor** is the direct product introduced in Subsection 1.3, which could be written directly in a package (assuming that is-tuple, card and the tuple selector are globally defined):

$$\text{DP} : \langle \times \rangle \rightleftharpoons \underset{G,H}{\forall}$$

$$G \times H = \underset{D}{\ni} \underset{x,y}{\forall}$$

$$D : \langle \epsilon,\ +,\ 0,\ - \rangle$$

$$\epsilon[x] \Leftrightarrow \left(\text{is-tuple}[x] \bigwedge (\text{card}[x] = 2) \bigwedge \underset{G}{\epsilon}[x_1] \bigwedge \underset{H}{\epsilon}[x_2]\right)$$

$$x + y = \left\langle x_1 \underset{G}{+} y_1,\ x_2 \underset{H}{+} y_2 \right\rangle$$

$$0 = \left\langle \underset{G}{0},\ \underset{H}{0} \right\rangle$$

$$-x = \left\langle \underset{G}{-} x_1,\ \underset{H}{-} x_2 \right\rangle$$

Here is an equivalent definition of the same functor, which is more in the spirit of archives and arguably more elegant:

$$DP : \langle \times \rangle \rightleftharpoons \underset{G,H}{\forall}$$

$$G \times H : \langle \epsilon,\ +,\ 0,\ - \rangle$$

$$\underset{x,y}{\forall}$$

$$\epsilon[x] \Leftrightarrow \text{is-tuple}[x] \wedge \text{card}[x] = 2 \bigwedge \underset{G}{\epsilon}[x_1] \bigwedge \underset{H}{\epsilon}[x_2]$$

$$x + y = \left\langle x_1 \underset{G}{+} y_1,\ x_2 \underset{H}{+} y_2 \right\rangle$$

$$0 = \left\langle \underset{G}{0},\ \underset{H}{0} \right\rangle$$

$$-x = \left\langle \underset{G}{-} x_1,\ \underset{H}{-} x_2 \right\rangle$$

Note that here we have for the first time explicitly used a compound term for denoting a namespace, but the translation proceeds as usual and results in the following *Theorema* formulae:

$$DP \Leftrightarrow$$

$$\underset{G,H}{\forall} \underset{x,y}{\forall} \left(\left(\underset{\underset{DP}{G \times H}}{\epsilon}[x] \Leftrightarrow \left(\text{is-tuple}[x] \bigwedge (\text{card}[x] = 2) \bigwedge \underset{G}{\epsilon}[x_1] \bigwedge \underset{H}{\epsilon}[x_2]\right)\right) \bigwedge \left(\left(x \underset{\underset{DP}{G \times H}}{+} y \right) = \left\langle x_1 \underset{G}{+} y_1,\ x_2 \underset{H}{+} y_2 \right\rangle \right) \bigwedge \right.$$

$$\left. \left(\underset{\underset{DP}{G \times H}}{0} = \left\langle \underset{G}{0},\ \underset{H}{0} \right\rangle \right) \bigwedge \left(\underset{\underset{DP}{G \times H}}{-} x = \left\langle \underset{G}{-} x_1,\ \underset{H}{-} x_2 \right\rangle \right) \right)$$

Observe that the formula above (except for the label DP) is also what the *Theorema* function “FlattenKB”, usually applied before starting a proof or a computation, would have made out of the earlier definition using the description quantifier. In fact, the general usage of ɜ is nonconstructive, so it is typically restricted to very specific settings where it can be eliminated (like *Theorema* does in ‘explicit’ definitions). Hence it is only natural to avoid it. Nevertheless, the user may still use it if she insists.

Namespace declarations also facilitate the specification of categories, as we have seen in the example of semigroups given in Subsection 3.3. In mathematics, categories are generally built up by gradual refinement—monoids, groups, abelian groups, rings, etc. This can be compared to the idea of **inheritance** in computer science. Consider the following archive version of a fragment of this refinement chain:

$\underset{R}{\forall}$ Ring[R] $\Leftrightarrow$

R : $\langle \epsilon, +, 0, -, *, 1 \rangle$

AbelianGroup[R]

Monoid[$\lceil \epsilon \mapsto \epsilon,\ \circ \mapsto\ *, 1 \mapsto\ 1 \rceil$]

Distributive[R]

$\underset{G}{\forall}$ AbelianGroup[G] $\Leftrightarrow$

G : $\langle \epsilon, +, 0, - \rangle$

Group[$\epsilon \mapsto \epsilon,\ * \mapsto\ +, 1 \mapsto\ 0, \square^{-1} \mapsto\ -$]

$\underset{\epsilon[x,y]}{\forall}$ (x + y = y + x)

$\underset{G}{\forall}$ Group[G] $\Leftrightarrow$

G : $\langle \epsilon, *, 1, \square^{-1} \rangle$

Monoid[$\lceil \epsilon \mapsto \epsilon, \circ \mapsto\ *, 1 \mapsto\ 1 \rceil$]

$\underset{\epsilon[x]}{\forall}\ \underset{\epsilon[y]}{\forall}\ \underset{\epsilon[z]}{\forall}$

$\epsilon[x^{-1}]$

$x * x^{-1} = 1$

$\underset{D}{\forall}$ Distributive[D] $\Leftrightarrow$

D : $\langle \epsilon, +, * \rangle$

$\underset{\epsilon[x,y,z]}{\forall}$ $(x + y) * z = x * z + y * z$

$\underset{M}{\forall}$ Monoid[M] $\Leftrightarrow$

M : $\langle \epsilon, \circ, 1 \rangle$

$\underset{\epsilon[x,y,z]}{\forall}$

$\epsilon[x \circ y]$

$\epsilon[1]$

$(x \circ y) \circ z = x \circ (y \circ z)$

In the formulae above, it is sometimes necessary to translate between certain symbols (e.g. between additive and multiplicative group notation). This is realized by using **theory interpretations** written as $\lceil \sigma_1 \mapsto \tau_1, \ldots, \sigma_n \mapsto \tau_n \rceil$. The intuitive meaning of this construct is quite clear: It is the finitely supported function that maps the symbols $\sigma_1, \ldots, \sigma_n$ to $\tau_1, \ldots, \tau_n$ and leaves all other inputs unchanged (this last requirement is only made for definiteness and could be omitted). More precisely, it could be defined as the following lambda expression:

$$\underset{\sigma}{\lambda} \begin{cases} \tau_1 & \Leftarrow & \sigma = \sigma_1 \\ \vdots & \Leftarrow & \vdots \\ \tau_n & \Leftarrow & \sigma = \sigma_n \\ \sigma & \Leftarrow & \text{True} \end{cases}$$

Its purpose is to 'build' a new operation object with appropriate operations; we will indicate its usage below. For the *Theorema* translation, the lambda expressions for the theory interpretations are retained but the usual replacement for the bound symbols in a namespace are carried out only in the right-hand sides of the lambda expression. Thus the formulae above will become:

$$\underset{R}{\forall}\left(\text{Ring}[R] \Leftrightarrow \left(\text{AbelianGroup}[R] \bigwedge \text{Monoid}\left[\left\lceil \epsilon \mapsto \underset{R}{\epsilon},\ \circ \mapsto \underset{R}{*},\ 1 \mapsto \underset{R}{1} \right\rceil\right] \bigwedge \text{Distributive}[R]\right)\right)$$

$$\underset{G}{\forall}\left(\text{AbGrp}[G] \Leftrightarrow \text{Group}\left[\left\lceil \epsilon \mapsto \underset{G}{\epsilon},\ * \mapsto \underset{G}{+},\ 1 \mapsto \underset{G}{0},\ \square^{-1} \mapsto \underset{G}{-} \right\rceil\right] \bigwedge \underset{\underset{G}{\epsilon}[x,y]}{\forall}\left(x \underset{G}{+} y = y \underset{G}{+} x\right)\right)$$

$$\underset{G}{\forall}\left(\text{Group}[G] \Leftrightarrow \text{Monoid}\left[\left\lceil \epsilon \mapsto \underset{G}{\epsilon},\ \circ \mapsto \underset{G}{*},\ 1 \mapsto \underset{G}{1} \right\rceil\right] \bigwedge \underset{\underset{G}{\epsilon}[x]}{\forall} \underset{\underset{G}{\epsilon}[y]}{\forall} \underset{\underset{G}{\epsilon}[z]}{\forall}\left(\epsilon[x^{-1}] \bigwedge \left(x \underset{G}{*} x^{-1} = \underset{G}{1}\right)\right)\right)$$

$$\underset{D}{\forall}\left(\text{Distributive}[D] \Leftrightarrow \underset{\underset{D}{\epsilon}[x,y,z]}{\forall}\left(\left(x \underset{D}{+} y\right) \underset{D}{*} z = x \underset{D}{*} z \underset{D}{+} y \underset{D}{*} z\right)\right)$$

$$\underset{M}{\forall}\left(\text{Monoid}[M] \Leftrightarrow \underset{\underset{M}{\epsilon}[x,y,z]}{\forall}\left(\underset{M}{\epsilon}\left[x \underset{M}{\circ} y\right] \bigwedge \underset{M}{\epsilon}[1] \bigwedge \left(x \underset{M}{\circ} y\right) \underset{M}{\circ} z = x \underset{M}{\circ}\left(y \underset{M}{\circ} z\right)\right)\right)$$

The actual meaning of theory interpretations becomes clear only when we consider their behavior in proofs (or MKM tasks as addressed at the beginning of Section 2). So assume a proof situation in which Group[Q] occurs in the assumptions. By instantiation and modus ponens on the Group definition above we obtain (besides the invertibility axiom):

$$\text{Monoid}\left[\left\lceil \epsilon \mapsto \underset{Q}{\epsilon},\ \circ \mapsto \underset{Q}{*},\ 1 \mapsto \underset{Q}{1} \right\rceil\right]$$

Writing M for $\left\lceil \epsilon \mapsto \underset{Q}{\epsilon},\ \circ \mapsto \underset{Q}{*} \right\rceil$, the definition of Monoid further yields:

$$\forall_{\epsilon_M[x,y,z]} \left(\epsilon_M[x \circ_M y] \bigwedge \epsilon_M[1_M] \bigwedge (x \circ_M y) \circ_M z = x \circ_M (y \circ_M z) \right)$$

Since symbols like ϵ_M are internally represented as M[ϵ] and the like, the rule of β-reduction and case distinction on the lambda expression above finally leads to:

$$\forall_{\epsilon_Q[x,y,z]} \left(\epsilon_Q[x *_Q y] \bigwedge \epsilon_Q[1] \bigwedge (x *_Q y) *_Q z = x *_Q (y *_Q z) \right)$$

Let us remark that the application of β-reduction and case distinction can be combined into a single computation step that can be intuitively understood as applying finitely supported functions on arguments.

4 Conclusion

In this paper we have introduced the notions of labels and namespaces, which permit a structured representation of a mathematical knowledge base; we call such a knowledge bases an archive. We have motivated the need for archives with various examples. Based on the *Theorema* language, archives offer constructs for splitting formu−lae in multiple cells, with quantifier ranging over whole cell groups and labels for attaching a name to the groups. This makes archives very readable and particularly suitable for large bodies of mathematical knowledge.

Symbols can be bound to a namespace, which is typically associated with the label attached to the cell group containing the symbols. Namespaces provide a unified approach for two important issues: On the one hand, the domains used for categories and functors; on the other hand, the intuitive usage of 'contexts' for resolving ambiguos symbols.

A decisive feature of archives is that all its language constructs, in particular the symbols "⇌" for attaching labels and ":" for declaring namespaces, are logic-internal: they extend *Theorema* in such a way that there is a natural translation back into higher-order logic.

We have studied archives from a statical perspective, analyzing their syntax, their expressive power and their relation to *Theorema*. The next step is to analyze their dynamical aspects, i.e. various operations on archives like merging, rearranging, searching, proving. Regarding the latter, a trivial solution would be to flatten the tree structure and feed the resulting assumption lists into a *Theorema* prover. But using archives—in their hierarchi−cal form—one may expect to reduce considerably the search space, both for mathematical knowledge retrieval and for proving. In the forthcoming report [Rosenkranz08], we will investigate the dynamical aspects of archives.

References

[AhoEtAl75] Alfred A. Aho, John E. Hopcroft and Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms*,Addison-Wes−ley, October 1975

[AspertiEtAl07] A. Asperti, C. Sacerdoti Coen, E. T. and S. Zacchiroli. User Interaction with the Matita Proof Assistant, in *Journal of Automated Reasoning*, Special Issue on User Interfaces for Theorem Proving, vol. 39/2, p. 109 − 139, 2007.

[Buchberger08] Bruno Buchberger, *Groebner Bases in Theorema Using Functors*, Proceedings of SCC'08, J.C. Faugere and D. Wang (ed.), p. 1—15, LMIB Beihang University Press, 2008.

[BuchbergerEtAl06] Bruno Buchberger, Adrian Craciun, Tudor Jebelean, Laura Kovacs, Temur Kutsia, Koji Nakagawa, Florina Piroi, Nikolaj Popov, Judit Robu, Markus Rosenkranz, Wolfgang Windsteiger, *Theorema: Towards Computer−Aided Mathematical Theory Exploration*, Journal of Applied Logic 4(4), pp. 470−504. 2006. ISSN 1570−8683.

[Buchberger04] Buchberger, Algorithm Supported Mathematical Theory Exploration: A Personal View and Stragegy, *Proceedings of AISC 2004 (7th International Conference on Artificial Intelligence and Symbolic Computation)*, RISC, Johannes Kepler University, Austria, Ed: B. Buchberger, B. and Campbell, J. , appeared as LNAI 3249, p. 236−250, Springer, Berlin−Heidelberg, 2004.

[Buchberger03] Bruno Buchberger, *Algorithm Retrieval: Concept Clarification and Case Study in Theorema*, SFB Report No. 2003−44, Johannes Kepler University Linz, Spezialforschungsbereich F013, October 2003.

[Buchberger03a] Bruno Buchberger, Algorithm Synthesis by Lazy Thinking: Examples and Implementation in Theorema, *Proceedings of the Mathematical Knowledge Management Workshop 2003*, Electronic Notes in Theoretical Computer Science, vol. 93, 24−59, 2003.

[Buchberger01] Bruno Buchberger, *Groebner Rings and Modules*, Proceedings of SYNASC 2001, S. Maruster, B. Buchberger, V. Negru and T.Jebelean (ed.), p.22—25, Timisoara, Romania, 2001.

[Buchberger99] Bruno Buchberger, Theory Exploration Versus Theorem Proving, *Proceedings of the Calculemus '99 Workshop*, Elsevier, 1999, Electronic Notes in Theoretical Computer Science, volume 23−3.

[Buchberger96] Bruno Buchberger, *Functor Programming in Mathematica*, talk at the Australian National University, Feb. 8, 1996.

[Buchberger96a] Bruno Buchberger, *Functors for Mathematics (The Theorema project)*, talk, Germany,1996.

[Coen04] Claudio Sacerdoti Coen, *Knowledge Management of Formal Mathematics and Interactive Proving*, PhD Thesis, University of Bologna, Italy, 2004.

[Harrison06] John Harrison, HOL Light: A Tutorial Introduction. *Proceedings of the First International Conference on Formal Methods in Computer−Aided Design* (FMCAD'96), Springer LNCS 1166, 1996. Updated and expanded version (2006) available at http://www.cl.cam.ac.uk/~jrh13/hol−light/tutorial_220.pdf.

[Hosn07] Kamal Aboul−Hosn. *A Proof−Theoretic Approach to Mathematical Knowledge Management*, Ph.D. Thesis, Cornell University, January 2007.
http://kamal.aboulhosn.org/phdthesis.pdf

[HosnAndersen05] Kamal Aboul−Hosn and Terese Damhøj Andersen, A Proof−Theoretic Approach to Hierarchical Math Library Organization, *Proceedings of the 4th International Mathematical Knowledge Management Conference*, pages 1−16, Bremen, October 2005.
http://kamal.aboulhosn.org/MKM05.pdf

[Fraenkel14] Adolf Fraenkel, Über die Teiler der Null und die Zerlegung von Ringen, *Journal für die reine und angewandte Mathematik*, A. L. Crelle (ed.), vol. 145, 1914.

[GieseBuchberger07] Martin Giese, Bruno Buchberger. Towards Practical Reflection for Formal Mathematics, extended abstract. In *Proceedings of Austria−Japan Workshop on Symbolic Computation and Software Verification*, RISC Technical report no. 07−09, July 1 2007. pages 30−34.

[Giese98] Martin Giese, *Integriertes automatisches und interaktives Beweisen: Die Kalkülebene*, Master Thesis, Fakultät für Informatik, Universität Karlsruhe, Germany, 1998.

[Noether21] Emmy Noether, *The Theory of Ideals in Ring Domains*, Mathematische Annalen, 83, 24-66, 1921.

[MacLaneBirkhoff67] Saunders MacLane and Garrett Birkhoff, *Algebra*, MacMillan, New York, 1967.

[Kohlhase06] Michael Kohlhase, *An Open Markup Format for Mathematical Documents (Version 1.2)*, LNAI 4180. Springer Verlag, Heidelberg, 2006.

[LibrechtMelis06] Paul Librecht and Erica Melis, *Methods for Access and Retrieval of Mathematical Content in ActiveMath*, Proceedings of ICMS−2006, Nobuki Takayama, Andreas Iglesias and Jaime Gutierrez (ed)., LNCS 4151, Springer, 2006.

[MelisEtAl03] E. Melis, E., J. Büdenbender, G. Goguadze, P. Libbrecht and C. Ulrich, Knowledge Representation and Management in ActiveMath, In *Annals of Mathematics and Artificial Intelligence*, vol. 38, p. 47−64, 2003.

[MillerYoussef02] B. Miller and A. Youssef, Technical Aspects of the Digital Library of Mathematical Functions, *Ann. Math. Artificial Intelligence*, 2002.

[Pitts03] Andrew M. Pitts, *Nominal Logic, a First Order Theory of Names and Bindings*, Information and Computation 186 (2003), 165−193, Elsevier, USA.

[PiroiEtAl08] Florina Piroi, Bruno Buchberger and Camelia Rosenkranz. *Mathematical Journals as Reasoning Agents: Literature Review*. Technical report no. 08−05 in RISC Report Series, University of Linz, Austria. March 2008.

[PiroiEtAl07] Florina Piroi, Bruno Buchberger, Camelia Rosenkranz, Tudor Jebelean. *Organisational Tools for MKM in Theorema.* Technical report no. 07−11 in RISC Report Series, University of Linz, Austria. 2007.

[Piroi04] Florina Piroi. *Tools for Using Automated Provers in Mathematical Theory Exploration.* PhD Thesis, University of Linz, Austria. August 2004.
Available as Technical report no. 04−12 in RISC Report Series, University of Linz, Austria.

[PiroiBuchberger04] Florina Piroi and Bruno Buchberger, *An Environment for Building Mathematical Knowledge Libraries*, Proceedings of the Workshop on Computer−Supported Mathematical Theory Development, Second International Joint Conference (IJCAR), Wolf−gang Windsteiger and Christoph Benzmueller (ed.), pp. 19−29. 4−8 July 2004. Cork, Ireland, ISBN 3−902276−04−5.

[Rosenkranz08] Camelia Rosenkranz, *Basic Operations On Mathematical Knowledge Archives*, forthcoming RISC report.

[RudnickiTrybulec01] P. Rudnicki and A. Trybulec, Mathematical Knowledge Management in Mizar, Proceedings of MKM 2001, Ed: B. Buchberger and O. Caprotti, 2001.

A1 Formalization Example: Lattice Theory (Fragment)

BinaryRelations $\rightleftharpoons$

BasicProperties $\rightleftharpoons$

$\underset{R}{\forall}$

$R : \langle \epsilon,\ \longrightarrow \rangle$

is-reflexive[R] $\Leftrightarrow \underset{\epsilon[x]}{\forall}\ x \longrightarrow x$

is-irreflexive[R] $\Leftrightarrow \underset{\epsilon[x]}{\forall}\ \neg\ (x \longrightarrow x)$

is-symmetric[R] $\Leftrightarrow \underset{\epsilon[x,y]}{\forall}\ (x \longrightarrow y \Rightarrow y \longrightarrow x)$

is-asymmetric[R] $\Leftrightarrow \underset{\epsilon[x,y]}{\forall}\ (x \longrightarrow y \Rightarrow \neg\ y \longrightarrow x)$

is-antisymmetric[R] $\Leftrightarrow \underset{\epsilon[x,y]}{\forall}\ (x \longrightarrow y \wedge y \longrightarrow x \Rightarrow (x = y))$

is-trichotomic[R] $\Leftrightarrow \underset{\epsilon[x,y]}{\forall}\ (x \longrightarrow y \vee (x = y) \vee y \longrightarrow x)$

is-dichotomic[R] $\Leftrightarrow \underset{\epsilon[x,y]}{\forall}\ (x \longrightarrow y \vee y \longrightarrow x)$

is-transitive[R] $\Leftrightarrow \underset{\epsilon[x,y,z]}{\forall}\ (x \longrightarrow y \wedge y \longrightarrow z \Rightarrow x \longrightarrow z)$

$\underset{R}{\forall}\ \underset{a}{\forall}$

$R : \langle \epsilon,\ \geq \rangle$

…

is-minimal[a, R] $\Leftrightarrow \underset{\epsilon[x]}{\forall}$

$\epsilon[a]$

$(x \neq a) \Rightarrow \neg\ (a \geq x)$

$$\text{is-maximal}[a, R] \Leftrightarrow \underset{\epsilon[x]}{\forall}$$

$$\epsilon[a]$$

$$(x \neq a) \Rightarrow \neg\,(x \geq a)$$

$$\ldots$$

$$\text{CompoundProperties} \rightleftharpoons \underset{R}{\forall}$$

$$\text{is-quasi-order}[R] \Leftrightarrow$$

$$\text{is-reflexive}[R]$$

$$\text{is-transitive}[R]$$

$$\text{is-partial-order}[R] \Leftrightarrow$$

$$\text{is-reflexive}[R]$$

$$\text{is-antisymmetric}[R]$$

$$\text{is-transitive}[R]$$

$$\text{is-strict-partial-order}[R] \Leftrightarrow$$

$$\text{is-asymmetric}[R]$$

$$\text{is-transitive}[R]$$

$$\text{is-equivalence}[R] \Leftrightarrow$$

$$\text{is-reflexive}[R]$$

$$\text{is-symmetric}[R]$$

$$\text{is-transitive}[R]$$

$$\text{Operations} \rightleftharpoons \underset{R}{\forall}$$

$$R : \langle \epsilon,\ \longrightarrow \rangle$$

$$\text{strict}[R] = \underset{S}{\ni}\ \underset{x,y}{\forall}$$

$$\underset{S}{\epsilon}[x, y] \Leftrightarrow \epsilon[x, y]$$

$$x \underset{S}{\longrightarrow} y \Leftrightarrow (x \longrightarrow y \wedge x \neq y)$$

$$\text{converse}[R] = \underset{S}{\ni}\ \underset{x,y}{\forall}$$

$\underset{S}{\epsilon}[x, y] \Leftrightarrow \epsilon[x, y]$

$x \underset{S}{\longrightarrow} y \Leftrightarrow y \longrightarrow x$

strict-converse[R] = $\underset{S}{\ni}\ \underset{x,y}{\forall}$

$\underset{S}{\epsilon}[x, y] \Leftrightarrow \epsilon[x, y]$

$x \underset{S}{\longrightarrow} y \Leftrightarrow \left(y \underset{R}{\longrightarrow} x \bigwedge x \neq y\right)$

neg[R] = $\underset{S}{\ni}\ \underset{x,y}{\forall}$

$\underset{S}{\epsilon}[x, y] \Leftrightarrow \epsilon[x, y]$

$x \underset{S}{\longrightarrow} y \Leftrightarrow \neg\ x \underset{R}{\longrightarrow} y$

…

…

Posets $\rightleftharpoons$

Definitions $\rightleftharpoons \underset{P}{\forall}$

$P : \langle \epsilon,\ \sqsubseteq \rangle$

is-poset[P] $\Leftrightarrow$

is-partial-order[$\lceil \longrightarrow\ \mapsto \sqsubseteq \rceil$]

Covering[P] $\Leftrightarrow \underset{\epsilon[a,b]}{\forall}$

covers[a, b] $\Leftrightarrow$

$a \sqsubset b$

$\neg\ \underset{\underset{P}{\epsilon}[x]}{\exists}\ ((a \sqsubset x) \wedge (x \sqsubset b))$

$\nLeftarrow$

$\sqsubset\ =$ strict[$\sqsubseteq$]

is-poset-with-variants[P] $\Leftrightarrow$

$P : \langle \sqsubseteq,\ \sqsubset,\ \sqsupseteq,\ \sqsupset \rangle$

is-poset[P]

$\sqsubset$ = strict[$\sqsubseteq$]

$\sqsupseteq$ = converse[$\sqsubseteq$]

$\sqsupset$ = strict-converse[$\sqsubseteq$]

is-poset[P] $\Rightarrow$

is-poset$\left[\left\lceil \underset{P}{\in},\ \sqsubseteq\ \mapsto \text{neg}\left[\ \underset{P}{\sqsubseteq}\ \right]\right\rceil\right]$

is-poset[$\langle$P, $\sqsubseteq\rangle$] $\Rightarrow$ $\underset{X\subseteq P}{\forall}\ \underset{m\in X}{\exists}$ is-minimal[m, X][$\sqsubseteq$]

is-poset[P] $\Rightarrow$

$\underset{X}{\forall}$

is-subposet[X, P] $\Rightarrow$

$\underset{\in[m]}{\exists}$ is-minimal[m, X]

$\underset{\in[m]}{\exists}$ is-maximal[m, X]