JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

Retrieval and Structuring of Large Mathematical Knowledge Bases in Theorema

Dissertation zur Erlangung des akademischen Grades

Doktorin der technischen Wissenschaften

Angefertigt am Institut für Symbolisches Rechnen

Eingereicht von:

Camelia Rosenkranz

Begutachtung:

Erstbeurteiler: A. Univ.-Prof. Dr. Tudor Jebelean

Zweitbeurteiler: A. Univ.-Prof. Dr. Josef Küng

Linz, *Februar 2009*

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Linz, Februar 2009

Camelia Rosenkranz

Abstract

In this thesis we develop a *Theorema* approach to ***Mathematical Knowledge Management*** (MKM) for the working mathematician. Mathematical knowledge archives are introduced as an extension of *Theorema*, allowing the representation of large bodies of mathematics with their inherent structure. Various logic–internal constructs for building archives in a natural way are presented: breaking formulae across cells, grouping them in a hierarchy, attaching labels to subhierarchies, disambiguating symbols by namespaces, importing symbols from other namespaces, describing categories and functors via namespaces. All these constructs are logic–internal in the sense that they have a natural translation to higher–order logic so that various operations of MKM can be carried out in the object logic itself.

In addition to input/output commands, operations for restructuring archives and the translation to plain *Theorema* are implemented; the latter provides a rigorous semantics of the archive language. As an alternative to user input, archives can also be expanded by theory exploration mechanisms. ***Retrieval***, a central topic of MKM, is treated in connection with theorem proving and algorithm synthesis. Two main types of retrieval are distinguished: syntactic and semantic. While syntactic retrieval employs textual search techniques, matching, and unification between the target formula and the knowledge base, semantic retrieval obtains the target formula from the knowledge base by applying basic inference steps. The *Theorema* commands implementing various flavors of retrieval are presented. As a larger example of an archive, we have formalized a chapter on lattice theory taken from a standard algebra textbook.

Zusammenfassung

In der vorliegenden Dissertation wird ein *Theorema* Ansatz zum ***Mathematischen Wissensmanagement*** (MKM) für praktizierende Mathematiker entwickelt. Mathematische Wissensarchive werden als Erweiterung von *Theorema* eingeführt zur Darstellung von umfangreichen mathematischen Texten mit ihrer inhärenten Struktur. Wir präsentieren diverse logikinterne Konstrukte für den natürlichen Aufbau von Archiven: Umbrechen von Formeln über Zellgrenzen hinweg, ihre Gruppierung in Hierarchien, Anfügen von Labels an Subhierarchien, Disambiguierung von Symbolen durch Namespaces, Symbolimport von anderen Namespaces, Beschreibung von Kategorien und Funktoren mittels Namespaces. Diese Konstrukte sind allesamt logikintern in dem Sinne, dass sie eine natürliche Übersetzung in die Prädikatenlogik höherer Stufe haben, sodass diverse MKM–Operationen direkt in der Objektlogik ausgeführt werden können.

Zusätzlich zu Eingabe/Ausgabe–Befehlen sind Operationen zum Restrukturieren von Archiven und zur Übersetzung in reines *Theorema* implementiert; letzteres gibt der Archivsprache eine exakte Semantik. Alternativ zur Benutzereingabe können Archive durch Mechanismen der Theorie–Exploration expandiert werden. Die Wissensabfrage, das sogenannte ***Retrieval***, ist ein zentrales Thema in MKM und wird im Zusammenhang mit automatischem Beweisen und Algorithmensynthese behandelt. Wir unterscheiden zwei Haupttypen von Retrieval: syntaktisch und semantisch. Während das syntaktische Retrieval textuelle Suchtechniken, Matching und Unifikation von Zielformeln und Wissensbasis einsetzt, extrahiert das semantische Retrieval die Zielformel von der Wissensbasis mittels elementarer Inferenzschritte. Wir präsentieren die *Theorema*–Befehle, welche die diversen Varianten von Retrievals implementieren. Als ein größeres Archivbeispiel haben wir ein Kapitel über Verbandstheorie aus einem Standardlehrbuch der Algebra formalisiert.

Table of Contents

1 Introduction

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 crystallize. 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 ***Mathematical Knowledge Management*** (MKM), which we survey in Chapter 2. The field is analyzed along the three axes: language and representation—organization and buildup—access and search. We describe the various trends present in the current leading groups and projects of MKM. Systems with tools for processing mathematical formulae are investigated, emphasizing logical questions and concentrating on the aspects of buildup, structure and retrieval of mathematical knowledge. On this background we introduce those features of *Theorema* relevant for MKM and those used as building blocks for the language of archives. As an integrated system for proving, computing and solving in a natural–style paradigm, *Theorema* provides an optimal platform for building up structured mathematics in the manner of the working mathematician.

Following the logic–oriented MKM approach of Theorema, we develop a logic–internal perspective on mathematical knowledge. The ***focus of this thesis*** is to formalize intuitive organizational constructs like labels and namespaces within the object logic. The resulting language—specified by an appropriate grammar—allows the representation and organization of complex mathematical texts in a unified framework. The dynamic aspects of this language are realized by various MKM operations. In particular, retrieval—formulated as a calculus of basic inference rules—is seen in its natural connection to proving and solving.

We approach some of the challenges inherent in organizing large bodies of mathematics by what we have called ***mathematical knowledge archives***—structured mathematical knowledge bases within the *Theorema* system. The language of archives is treated extensively in Chapter 3. All its constructs are logic–internal in the sense that they are part of a slight extension of the higher–order predicate logic of *Theorema* (henceforth referred to as “plain predicate logic”), and they have a natural translation to plain predicate logic.

After pointing out some of the typical difficulties encountered in formalizing real–life mathematical texts, we present various constructs for building archives that overcome these difficulties. The first group of language constructs are merely ***notational shortcuts*** that are useful for overcoming the limitations imposed by the cell boundaries in *Mathematica:* Formulae can be grouped into nested

blocks and spread across different cells. The point is that a hierarchy of nested *Mathematica* cells allows to view archives at different levels of abstraction by expanding or collapsing suitable cell groups.

The language of archives extends plain predicate logic by essentially only two symbols, one of which is the "equi" symbol ⇌. It is used for attaching labels to subhierarchies of formulae, but in such a way that the ***labels*** are a part of the object language, namely propositional constants denoting the truth value of the underlying formulae (interpreted as a conjunction). Since the subhierarchies beneath a label may themselves contain further labels, they can be viewed as a formal counterpart to the segmentation of a book in chapters, sections, etc.

The other symbol extending plain predicate logic is the colon : used for declaring what we have called ***namespaces***. As in some programming languages, the purpose of a namespace is to bundle symbols for disambiguating them from identical symbols residing in a different namespace. As with the labels, an important point is that namespaces are a part of the object language, essentially higher–order functions applied in curried notation. If symbols from another namespace are needed in some portion of an archive, one can "import" them by opening the required namespace before the point of their first usage. Sometimes it is practical to fuse a namespace containing various symbols with a label encompassing them. These so–called "home namespaces" are particularly useful for definitions and axioms, where the home namespace holds exactly the symbols to be defined or characterized; this can be an important speedup for retrieval engines searching for useful formulae involving given symbols.

Namespaces were designed to be fully compatible with the operation objects of ***categories and functors*** in the sense of [Buchberger08, Buchberger01] rather than of Eilenberg and MacLane. Categories provide an elegant tool for organizing the above–mentioned axiomatization process with respect to the symbols it involves: Their operation objects are simply namespaces bundling the operations (functions / predicates / constants). Functors capture the intuitive notion of constructions used throughout mathematics; they transform a given operation object into another, typically giving rise to a mapping between two categories. For example, the construction of polynomials is a functor from the category of unital commutative rings into itself.

After describing how archives look like and how they can be used for representing structured mathematical repositories, we turn in Chapter 4 to various important ***operations on archives*** and their implementation. We introduce so–called mathbooks—the user interfaces to archives—by their formal grammar. By ***parsing*** their box structure, we obtain "concise" archives, i.e. archives with a richer language, that permits natural descriptions and avoids redundancies. This concise form of an archive is then expanded into a "verbose" form, by distributing quantifiers and normalizing labels. After presenting the grammar of "verbose" archives, we describe the reverse process of formatting an archive as a mathbook as well as some other simple input/output operations on archives.

Several operations for combining archives are implemented. These commands differ in their treatment of global symbols: identifying them or wrapping them in separate namespaces or as a mixture of these. For motivating the different choices, a representative mathematical example is exhibited for each. Mechanisms for inserting and extracting parts of an archive are presented with simple examples from computer science. Besides these ***simple structural operations*** on archives, the current implementation provides also more sophisticated ones like theory exploration and knowledge retrieval. While the topic of retrieval is deferred to Chapter 5 for a detailed study, we treat theory

exploration within Chapter 4. It is seen in the context of archives as a specific way of automated knowledge buildup guided by suitable definition schemes and proposition schemes.

The ***translation*** of an archive to plain *Theorema* involves a partial loss of structure, breaking the hierarchy of an archive into a flat knowledge base. The result of translating an archive will be essentially a logical formula. Namespaces and labels are eliminated so that the resulting formula blurs the distinction between "logic" and "organization" (e.g. it can not distinguished whether a $\Leftrightarrow$ stems from a $\rightleftharpoons$ or not). The actual flattening process comes in two variants: either a full elimination of the hierarchy, yielding a list of *Theorema* formulae, or a shifting of the hierarchy to a logic–external level (by translating the nested $\rightleftharpoons$ symbols into nested **`Theory`** environments of *Theorema*). The main purpose of this embedding into plain predicate logic is to provide a rigorous semantics of the archive language; besides that, it also connects archives with the other components of the *Theorema* framework.

An important problem in MKM is ***mathematical knowledge retrieval***, i.e. finding information on a given notion in a knowledge base. Chapter 5 is devoted to a detailed study of this issue, both for plain *Theorema* knowledge bases and for archives. It turns out to be a highly nontrivial task, due to the nature of mathematical information: The essence lies in its logical structure rather than its "literal" appearance: Equivalent formulae will not be discovered by a purely textual search engine, so a more general notion of retrieval is needed.

From the integrating perspective of *Theorema*, we can see retrieval in connection with both theorem proving (including program verification) and algorithm synthesis. Inside the proof of a theorem, extra definitions, theorems and lemmata (perhaps occurring in the knowledge base in an equivalent form) are usually needed besides the given assumptions. Retrieval mechanisms can thus be seen as collections of ***basic inference rules*** used to obtain the formula queried—or a semantically identical one, obtained by "simple" derivations on the knowledge base. Choosing a suitable collection of simple inference rules is a crucial and difficult step—it strikes the right balance between full–fledged theorem proving as one extreme and textual search as the other.

Two main ***types of retrieval*** are distinguished: syntactic and semantic. The former employs textual search techniques, matching and unification between the target formula and the knowledge base. The latter obtains the target formula from the knowledge base by applying basic inference steps. *Theorema* commands implementing various flavors of retrieval are also presented. In the context of archives, retrieval is more efficient than its plain counterpart on knowledge bases when the additional structure encoded in the namespaces is employed.

The thesis concludes by a critical reflection of what has been achieved and by an outlook on various issues of future research/development. The Appendix contains a chapter on lattice theory taken from a standard algebra textbook, formalized as a mathematical knowledge archive.

Main Contributions of the Thesis

- ***Statusquo of MKM:*** The literature survey presented in Chapter 2 is, as far as we know, the only one that gives a detailed and up–to–date description of the most important systems involved in MKM, its current trends and future challenges.
- ***Labels:*** Realized essentially as propositional constants, they provide a logic–internal construct for structuring mathematical texts hierarchically, in a natural analogy to the logic–external

chapter or section headings appearing in traditional mathematical textbooks. As far as we know, this is the first attempt to bring "names" referring to blocks of formulae into the object logic.

- ***Namespaces:*** For disambiguating synonymous symbols, namespaces are a convenient logic–internal construct that supplements the necessary context information. Based on the same natural intuition as their well–known counterparts in programming languages, they allow to bundle symbols into a single operation object and are therefore compatible with the domains of categories and functors.

- ***Home Namespaces:*** When working with definitions or axioms, it is often natural to associate them simultaneously with a label and a namespace designated by a common identifier. Such a fusion then allows to refer both to the group of formulae constituing the definition / axiom and to the bundle of the very symbols defined or characterized.

- ***Language of Archives:*** Combining labels and namespaces with various notational devices like multi–line quantifiers, archives provide a natural and compact interface. Their language is specified by suitable grammars at different levels of abstraction.

- ***Syntactic and Semantic Retrieval:*** Viewing retrieval as an integral part of theorem proving, we gave a classification of various types of retrieval. As opposed to the syntactic (i.e. textual) retrieval, its semantic counterpart employs a carefully selected calculus of basic inferences for finding an "occurrence" of the target formula. For syntactic retrieval, we provide the new notion of explicit synthesis in addition to the established techniques of searching occurrences and matching.

- ***Case Studies:*** For illustrating the organizational concepts and exemplifying the operations on archives, several case studies (of varying extent) have been carried out in diverse fields of mathematics and are interspersed in the thesis (e.g. from tuple theory, algebra, order theory). Furthermore, the Appendix formalizes a chapter on lattice theory from a standard textbook in the language of archives.

- ***Implementation:*** The language of archives (its parsing and formatting rules) and all operations (like merging, splitting, inserting, projection, retrieval, and exploration) are implemented as *Mathematica* programs within the framework of *Theorema.*

Statement of Originality. All of the above contributions are due to the author, except for the following inputs coming from Bruno Buchberger and communicated mainly in the *Theorema* Syntax Seminars organized by him: In fact, his seminal idea of using logic–internal devices for structuring mathematics was the starting point for this thesis, and the present notion of archive labels is essentially a combination of two of his proposals in the seminar (named "Translator 1" and "Translator 2" at that time). Moreover, he introduced the view of retrieval as "symbolic computation proving", which is the basis of our calculus of basic inferences. Conducive impulses have been provided by my supervisor Tudor Jebelean in detailed private discussions.

2 Mathematical Knowledge Management

In the present chapter, we will try to sketch the picture of the discipline MKM as it presents itself at the current moment. We note also that the list of MKM systems given in Section 2.3 is the basis for the corresponding part of the report [PiroiEtAl08].

2.1 Origin and Perspectives

The challenge of ***building up***, ***structuring*** and ***retrieving*** mathematical knowledge is not new, but it becomes more and more acute as the volume of mathematical information on different subjects increases. For instance, in order to avoid extensive literature search, mathematicians often prefer reinventing lemmas, 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.

Thus tools for managing mathematical knowledge (e.g. providing well-structured formalized mathematical information in an electronic format that can be browsed and searched automatically) are urgently needed. The underlying software technology is already here—but the ideas and the techniques are yet to be developed.

Developed from the outset as an interdisciplinary field, MKM finds itself at the intersection of mathematics, logic, computer science, and scientific publishing. Its goal may be summarized as the discovery of new techniques—based on formal mathematics and supported by software tools—for building up repositories of mathematical knowledge and structuring them from a practical perspective: e.g.mathematicians should be helped in their ***day–by–day research***, where knowledge retrieval is needed for finding slightly different formulations of an existing lemma, doing simple proofs etc.

As a research field, MKM was launched in September 2001 at the First International Workshop on MKM in Hagenberg, Austria (see the Special Issue [BuchbergerEtAl03] of the Annals of Mathematics and Artificial Intelligence). In its call for papers, Bruno Buchberger defined MKM as computer–supported management of mathematical knowledge, parsing "mathematical knowledge management" as "(mathematical knowledge) management" rather than "mathematical (knowledge management)". A dedicated conference [AutexierEtAl08] now reached already its eighth year. In this context, (MK)M should be understood as the general investigation of how one can manage ***mathematical knowledge*** with its very specific features: highly hierarchical structure, infinitely many variants of different logical foundations, etc.

In order to develop the new academic field, several ***European projects*** have been initiated: [Calculemus], [MOWGLI], [MKMNet], and others. Their objective is to identify various subdomains of MKM and to clarify foundational questions. In the actual evolution of these projects, one can also see the difficulty of formalizing everyday mathematics as well as the subtly different views of the scientists involved. Here, we want to present the emerging field of MKM mainly from the logical perspective, surveying various ideas for structuring, building up and retrieving mathematical knowledge.

2.2 Current Trends in Knowledge Buildup and Retrieval

Having formalized mathematical information in a computer–readable format is not enough for being able to use it in proving, solving or computing. One needs on top of the knowledge an organizational structure with browsing and searching capabilities. In a first approximation, we call a system with these capabilities an ***MKM System***. Even though there are various viewpoints as to what MKM encompasses [AspertiZacchiroli04, CaprottiEtAl04, Farmer04, MinerTopping03, MKMNet4.1, PiroiEtAl07], one can identify some ***characteristics*** that should be part of an MKM system. Along the lines of [Farmer04], we regard as defining components:

- the ***language*** (first-order or higher-order logic, simply-typed or some form of polymorphic lambda calculus), having a certain ***representation*** (that deals with the notational issues),
- the way in which the mathematical theories are ***organized*** both logic-internally (i.e. by their logical structure) and logic-externally (i.e.by their segmentation in chapters), and
- facilities of ***access***, i.e.tools for carrying out ***searches***, for performing deductions or computations in a (fully or partly) automated way, and for displaying the contents and structure of the knowledge under consideration.

In the following we present a more detailed description of these characteristics, emphasizing the logical aspects of build-up, structure and retrieval of mathematical knowledge.

2.2.1 Language and Representation

Formalized mathematical knowledge is expressed in a ***logical language***. This can vary from classical first order logic, as in the case of [Mizar], up to some version of higher order dependently–typed polymorphic lambda calculus, as in [CoQ]. It may include sorts like [Isabelle/Fol] or do without them like [TPS], it can be based on an underlying set theory like [Mizar], [Metamath] or dispense with it like [Ωmega].

Most people seem to prefer a formal language close to that of working mathematicians, as expressed in the concept of a “mathematical vernacular” [Bruijn87, Wiedijk09].

For storing mathematical knowledge one needs a ***representation***. The existing representation languages differ from system to system. Some of them are application–independent, like [MathML] and [OpenMath]. Other systems, like Mizar and *Theorema*, have their own representation language that can be easily translated to MathML, OpenMath or OMDoc [BancerekKohlhase07, KutsiaNakagawa01].

MathML was created for presenting ***mathematical formulae on the web***. Its authors described it as “an XML application for describing mathematical notation and capturing both its structure and content. The goal of MathML is to enable mathematics to be served, received, and processed on the World Wide Web, just as HTML has enabled this functionality for text.” Its main parts are presentation MathML, a markup used for the rendering of mathematical content on the web (can be seen as an

extension of HTML), and content MathML, the actual mathematical content that can be processable by machines.

It covers "high–school mathematics" (by having around 80 specialized elements for the most common functions and individual constants) and can currently be rendered on most available web browsers with the help of plugins.

OpenMath was originally constructed for facilitating communication of mathematical knowledge between different computer algebra systems. It extends content MathML by using so–called ***content dictionaries*** that define the "semantics" (or "meaning") of the objects expressed by the OpenMath symbols.

While content MathML defines the ***meaning*** of symbols only in the sense of an informal comment directed to human readers (as it is explained in the MathML Recommendation), OpenMath aims to provide machine-readable content dictionaries, defining the meaning of mathematical concepts expressed by OpenMath symbols [Kohlhase06]. Up to now, what is done in establishing the "meaning" of a concept like + is

- to select certain properties considered crucial (e.g. commutativity) and to formalize them in a custom-tailored language,
- to offer a common reference point for MKM-aware applications such that they can agree on one and the same concept of addition (no matter how it is represented in either application).

The context of a mathematical formula is strongly related with the way in which the mathematical information is build up and organized. As we see in the following subsection, languages like [OMDoc] are a building block for the organization of mathematical knowledge.

2.2.2 Organization and Buildup

In mathematics, one usually does not study a mathematical formula in isolation, but in the context of a ***mathematical theory***. For storage and organization of mathematics, the context where a formula occurs is therefore important. This is the premise of OMDoc [Kohlhase06], which is based on the following three-level hierarchy:

- mathematical formulae (coded in content MathML or OpenMath),
- various statements (like definitions, assertions and examples),
- theories (related by morphisms for expressing e.g. theory inclusion / interpretation).

One way of organizing mathematical knowledge by considering the mathematical knowledge as a book, and splitting it in sections and paragraphs, being included in and/or having links to other "books". The organization can also be ***logic–internal***, i.e. depending on the mathematical content.

An example of organized ***library of mathematics*** is the MML library of Mizar, which is, at this moment, the biggest repository of mathematical knowledge (see next section, the subsection on Mizar). Other libraries of mathematics include HELM [AspertiEtAl00] and MBase [KohlhaseFranke01]. One may also view the content dictionaries of OpenMath itself as a primitive

structuring tool. The ongoing efforts for reworking the theory of special functions into an online database in the DLMF project are another instance of structuring a considerable portion of mathematics. Also the authoring tools presented below have an organizational aspect.

Building mathematical knowledge is a first step necessary for storing and organizing mathematical knowledge. This process entails the generation of mathematical information by

- authoring new documents,
- supporting (or even automating) the systematic exploration of mathematical theories and
- retrodigitization (reconstructing mathematical contents from electronic files written in formats like LaTeX, or obtaining a scanned/text version out of a printed one).

There exist different tools for supporting the ***authoring*** of new documents. In *Theorema*, using the Label Management [PiroiBuchberger05] tool, the user can segment the mathematical content in chapters, sections. She can also choose the type of information input (e.g. whether is a **`Definition`** or a **`Theorem`**). Another example of supporting authoring of mathematical documents is ActiveMath, where one can input OMDoc documents interlaced with textual information.

Systematic theory exploration [Buchberger00] is a natural environment for mathematics. The buildup of mathematical knowledge is an involved process that can be carried out in various different ways [Buchberger99]. One can contrast top-down versus bottom-up approaches [Buchberger04]. In the latter case, one studies systematically all interactions between the concepts involved: Each time a new predicate or function is introduced, one investigates all interesting interactions to the old concepts. Thus one builds up a mathematical theory in layers, starting from elementary notions and axioms.

In the top-down approach, one starts out with a rudimentary knowledge base containing only the 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 will in turn produce new auxiliary lemmata, thus producing a whole ***cascade*** [BuchbergerCraciun03] of new mathematical formulae. 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 should be combined.

Theory exploration can also be contrasted to ***isolated theorem proving*** [Buchberger99], where one usually proves a single important theorem, directly from the axioms of the domain in which this theorem holds.

In mathematical practice, though, theorems typically come in a context of various propositions and lemmata that can and should be used (provided they are already proved). Thus exploring a mathematical theory can help to produce shorter and clearer proofs. MATHsAID [McCaslandBundy06] and HR [ColtonEtAl02] provide tools for automated theory exploration, SysteMathEx [CraciunHodorog07, Craciun08] carried case studies in systematic theory exploration. During the years, people also developed several domain-specific custom-tailored automated conjecture generators; see the survey papers [Larson05, Larson06].

Retrodigitization of mathematical knowledge is a vast field on its own. It starts with the numerous

retrodigitized mathematical journals available on the web, e.g. [EMIS], [JSTOR], [WDML–Göttingen], usually in pdf-format, and continues with tools like Hermes [Anghelache04] that translate TeX documents into machine-processable texts (e.g. OMDoc texts).

2.2.3 Access

An important role in ***accessing*** mathematical knowledge is played by various version of searching.

Bibliographic searches are based on BibTeX fields like author, title, keywords, etc. They perform online searches for mathematical information independent of the structure and content of the mathematical knowledge, taking into account only the metadata information stored in the BibTeX fields associated with documents containing mathematical information. Some examples are Zentralblatt Math or MathSciNet.

Mathematical web services also perform online searches for mathematical information. As explained in [AspertiZacchiroli04], generic "web services are software systems designed to support machine interaction over the network (usually the internet), typically exposing a programming interface based on exchange of XML documents." Mathematical web services are web services specialized for communicating mathematical information. For the latter, MSDL documents (Mathematical Service Description Language or MSDL is a standardized XML format) describe mathematical problems by their input/output type and pre/post conditions [BuswellEtAl03]. As such, mathematical web services can be seen as a special case of content-based searches (in this case the content is restricted to the pre/postconditions).

Mathematical web services can be classified according to

- specific problem classes treated (for each problem type the precondition and postconditions are given),
- standardized taxonomies like [GAMS]
- algorithms implemented by the individual services.

Examples of mathematical web services are [MONET], MathBroker [Baraka06] and MathServe [ZimmerAutexier06].

We name here as ***formula retrieval*** a search depending on the structure and content of mathematical knowledge. This type of knowledge retrieval provides techniques for searching (the content of) mathematical knowledge, covering various levels of difficulty: textual search, pattern matching, simple inference steps, full-fledged theorem proving.

In our view the activities of buildup, organization and formula retrieval are tightly interconnected: While appropriate structuring of knowledge should facilitate the retrieval process, appropriate buildup of knowledge will lead to a more natural structure of the knowledge base.

In the optimal case, a complete exploration saturates a given knowledge base by adding all properties that were "difficult" to prove (typically using domain-specific reasoners). Then ***retrieval*** of a mathematical formula will only involve "easy" inference steps [Buchberger01], yet a bit more than textual search and pattern matching: also formulae that are "obviously equivalent" to it should be found. More precisely, we want to find any formula that follows "easily" from the knowledge base by

an efficient variant of proving that may be called “symbolic computation reasoning”, “rewrite-style inferences”, etc. [Buchberger03]. We will come back to this point in Chapter 5.

Processing mathematical knowledge is tightly connected to ***formal proofs***. For their creation, one can see a wide spectrum of tools, ranging from proof checkers (given an assertion and a proof, check whether the proof establishes the assertion) through proof assistants (given an assertion, build up a proof interactively) to theorem provers (given an assertion, generate a proof automatically).

Of course, there is often a ***blurred transition*** between these cases (and the terminology is not always consistent). For example, some proof checkers may offer special modes (“auto tactics”) for automatically generating simple proof fragments, while an automated theorem prover is often too weak to prove an interesting theorem without breaking it up into numerous auxiliary lemmata.

The existing tools dealing with ***retrieval*** of mathematical knowledge depend to a larger extent on some representation of the mathematical knowledge involved, and not so much on its logical meaning.

Most of the tools for searching—available in [ActiveMath], [DLMF], [MathWebSearch] etc.—are strongly influenced by the Semantic Web Community, considering the retrieval of mathematical formulae as a special case of information retrieval.

Searches operating on mathematical formulae are reduced to database queries working on metadata (certain crucial properties of the formulae). In this view, search is ultimately textual retrieval, reaching at most some form of pattern matching. The advantage, though, is a relatively fast retrieval process.

2.3 A Survey of Leading Groups and Projects

In the following section, we present (in alphabetical order) *systems* that exemplify the current status in mathematical knowledge management.

2.3.1 ActiveMath

ActiveMath is an ***interactive learning environment*** for mathematics [LibbrechtMelis06], based on an extension of OMDoc. It is web–based and user–adaptive. Its building blocks are content items holding both mathematical expressions and text (that can contain hyperlinks); they are annotated with mathematical and pedagogical attributes and relations (i.e. metadata). For managing, referencing and searching mathematical information, it mainly uses the OMDoc level of definitions, examples and theorems.

Accessing mathematical information starts by opening a *course* containing several content items provided by an author. Unknown concepts or symbols can be linked by the authors to their corresponding content item (e.g. their definition) and are thus readily available to the user. Alternatively, the user can search by the name of a content item. The courses visible to a user are automatically adapted to his learning goals, his knowledge and the previous scenarios. Checking the user exercises is done by evaluating the mathematical input (by a computer algebra system connected to it) and providing correspondingly feedback.

LeActiveMath is a ***language–enhanced*** version of ActiveMath that improves among many things the search techniques not only for English, but also for German and Spanish. Another improvement is the use of the interactive concept mapping tool iCMap [MelisEtAl05].

iCMap is an ***interactive visual tool***, helping students discover relations between mathematical concepts. Students can play with notions borrowed from the existing ActiveMath courses, creating links between them, and validating them (after the validation, the program colors the correct edges in green, and the incorrect ones in red). The user can construct concept maps and use the validation for all his edges (global verification) or only for one (local verification). The verification is made by a small set of inferences upon the knowledge base and upon the (independent) authored information existing in the exercise. The user input is basically matched with the existing ontologies. E.g., if the user draws an edge between a and b, iCMap checks whether there is a relation between the object associated with a and the object associated with b. A relation can:

- be added by the author of the document, as a template edge,
- relations existing in the underlying OMDoc source (the OMDOC element theory has the purpose of gathering notions, definitions and theorems into theories),
- deductive relations, like **`is_equivalent`**, **`belongs_to`**. Two definitions relating to the same symbol are linked by the **`is_equivalent`** edge (are considered equivalent). Also transitivity rules are build in for several types of edge (like **`is_a`** or **`belongs_to`**).

In case of failure, explanations are given to the user like “This edge has a wrong type”, ”This edge is correct, but subsumes several steps. Please elaborate”.

Search queries in ActiveMath can be combinations of:

- ***Text queries*** that allow exact phrase searches or fuzzy searches. The latter are based on phonetic and edit-distance fuzziness.
- ***Attribute queries*** that check whether the mathematical item has certain characteristics. For example, one can search (in the current course) for all items of type “definition”, or for all items having the difficulty “very easy”.
- ***Mathematical expression queries*** that are entered by the [WIRIS] input editor.

Mathematical expressions are converted into text (linearizing the tree expression by giving depth indication for the components) and then indexed by [Lucene], a full-featured text-searching library. The actual *search* in ActiveMath treats both the formal and the informal part of an OMDoc document in the same way. By its ranking heuristic, it sorts the results that the user sees as a list of titles (hyperlinks to the real matches).

2.3.2 Coq

CoQ is a formal proof management system [BertotCasteran04]. Its logical framework is the Calculus of Inductive Constructions [GimenezCasteran06].
It provides the following ***features***:

- Definition of functions and predicates, statement of mathematical theorems, specification of software.
- Interactive construction of formal proofs that can be checked mechanically by a small certification kernel.
- Automated generation of certified programs from proofs of their specifications.
- Execution of functional programs on appropriate pieces of input.

Certified mathematical knowledge is stored in knowledge files but can only be used by explicitly referring to the relevant knowledge items through their names (the only search mechanisms available are grep and similar text-based tools). A recent example of a large knowledge library named C-Corn, building up portions of constructive algebra and analysis, has been developed at Nijmejen [CruzFilipe04].

The object-oriented language FoCAL, previously known as FoC, follows a somewhat different approach [Prevosto05]: Algorithms in computer algebra are verified by coding proofs in a specialized language that is then translated to COQ for ***verification***.

An external search tool for the CoQ library is Whelp, an experimental web searching and browsing tool, explained in the subsection on HELM.

2.3.3 DLMF

DLMF or the **Digital Library of Mathematical Functions** [MillerYoussef03, Youssef04] intends to be the (online and printed) successor of Abramowitz and Stegun's "Handbook of Mathematical Functions" [AbramowitzStegun65], a standard reference for engineers throughout several decades. The project, headed by the National Institute of Standards and Technology (NIST), creates documents coded in LaTeX, using special macros for producing a handbook on paper as well as a dynamic website.

This website will provide mathematical formulae, 3D interactive graphics, methods of computation, references, links to software, and an ***equation search capability***. The latter is understood as a layer on top of textual search. Mathematical information is preprocessed in the following way [Youssef04, Youssef05]:

- ***Textualization***: The mathematical information is transformed into structured text. For example, $x^a + y^b - t$ is encoded as **`x super a endsuper plus y sub b endsub minus t`**.

- ***Flattening and scoping***: The scope of quantifiers (linguistic constructs that bind variables) is made explicit and their (typically two-dimensional) syntax is linearized as in the previous step. For example, $\int_0^1 y\,dy$ becomes **`integral sub 0 endsub super 1 endsuper integrand y endintegrand dy.`**
- ***Normalization***: The mathematical equation is brought into a normal form with respect to certain algebraic and notational rules. For example, $a\,b^{-1}\,c\,d^{-1}$ has the normal form **`frac{ac}{bd}`**, and $x_2{}^3$ and $(x^3)_2$ both have the normal form **`x sub 2 endsub super 3 endsuper`**.

The preprocessed mathematical information is then indexed by the underlying text search system. A ***query*** (input in a simplified LaTeX-like query language) goes through the same process as above and triggers a search in the index [Youssef07, YoussefShatnawi06].

Besides this equation search, the user can also search for keywords like “Bessel Function” or browse the whole library of mathematical functions available. Once the desired function is found, one can see the actual formula and its properties (where it is referenced, informations about its components) and in several cases 2D or 3D visualizations of the function with MathView [WongSaunders05].

For example, Euler's Beta function

$$B(a, b) = \int_0^1 t^{a-1}(1-t)^{b-1}\,dt = \frac{\Gamma(a)\,\Gamma(b)}{\Gamma(a+b)}$$

includes among its properties:

- the function B(a,b) being defined
- the meaning of the symbols occurring in the definition: **`Γ(z)`** as the Gamma function, **`a`**, **`b`** being real or complex variables, with links to their definition;
- and the links to formulae using the function (in this case Euler Beta Integral).

An experimental version of the whole library is available at http://dlmf.nist.gov/.

2.3.4 HELM

HELM stands for the ***Hypertextual Electronic Library of Mathematics***. The system "aims to study and develop a technological infrastructure for the creation and maintenance of a virtual, distributed, hypertextual library of formal mathematical knowledge" [AspertiEtAl00]. HELM imports libraries from COQ [BertotCasteran04] and (experimentally) from Nuprl [Jackson94], transforms the representation into an intermediate MathML content representation [GoguadzeEtAl02], and makes it available on the internet for browsing and searching. One can access directly the mathematical objects (i.e. definitions, theorems etc.) or browse through the theories (i.e. structured collections of explanatory text and mathematical objects, assembled by an author for presentational purposes). The mathematical information can be viewed as raw semantic encoding, as a MathML formula or presented as HTML (generated on the fly from the MathML encoding).

The user can add its own document (written in LaTeX or in CoQ) to the content of the library. These documents will be then translated to MathML, with the help of HERMES [Anghelache04] and a script converting CoQ mathematical formulae to MathML.

The ***search engine*** of HELM is called Whelp [AspertiEtAl00a, AspertiEtAl06] and is located at http://helm.cs.unibo.it; it has its own syntax, which in most kind of queries is disambiguated into suitable terms of Coq's Calculus of Inductive Constructions.

Whelp is based on the ***metadata*** model described in [AspertiSelmi04, Coen04, GuidiCoen03]. Metadata is used for indexing mathematical notions and marking input/output types as well as the components in the body (hypotheses, conclusion, consistency proof of the notion). For the metadata, only the constant symbols of the original mathematical formula are taken into account. For example, from $\forall_{n:nat}\ \forall_{m:nat}\ \forall_{p:nat}\ n*(m+p) = n*m + n*p$, the automatically extracted metadata contains only the constants **nat**, $=$, $*$, $+$ and their position (hypothesis/conclusion) in the formula. Search is then not performed on the stored knowledge, but only on the metadata. Thus one may view the metadata as taking the role of indexes in a database, allowing quick searches.

The ***search facilities*** are as follows:

- ***Locate*** is the simplest search. The user introduces a string regular expression, and the tool outputs the list of all formulae that have this string as "name".
- ***Hint*** retrieves all theorems that can be applied to derive the current goal. This means, given a theorem $t_1 \longrightarrow t_2 \longrightarrow \ldots \longrightarrow t_n \longrightarrow t$ and a goal **g**, it checks whether there exists a substitution θ such that $t\,\theta = g$. This is simulated in HELM by metadata filtering.
- ***Match*** amounts to reverting the operation of indexing, looking for terms matching the metadata set, which are automatically computed from the user input. The query $\forall_{x,y,z:nat}\ x*(y+z) = x*y + x*z$ gives answers that have a type α–convertible with the input. One of them, having the type $\forall_{n:nat}\ \forall_{m:nat}\ \forall_{p:nat}\ n*(m+p) = n*m + n*p$, is located in the file **cic:/Coq/Arith/Mult/mult_plus_distr_l.con**. Also "fake matches", like a

symmetric analog of distributivity, having type $\forall_{n,m,p:nat} (n + m) * p = n * p + m * p$ and being located in **`cic:/Coq/Arith/Mult/mult_plus_distr_r.con`** are found.

- ***Elim*** gives the induction axioms for a domain. It is used for inductive proofs of properties of functions and relations over algebraic types. For example, giving nat as input, we obtain $\forall\ P.P\,0 \rightarrow (\forall\ n.(P\,n) \rightarrow (P\ n+1)) \rightarrow (\forall\ n.(P\,n))$ the usual induction principle on natural numbers.

Also Semantic Web techniques custom-tailored for ***mathematical content*** [AspertiEtAl06, AspertiEtAl00a], are used in HELM. The difference between HELM and OpenMath–based platforms is in their different use of XML. Whereas HELM uses XML only for the management of libraries by proof assistants, OpenMath employs it for communication between applications.

For manipulating ***mathematics on the web***, the mathematical web service called [MONET] and an interactive prover called Matita [AspertiEtAl07a] were implemented in the same framework. Matita [AspertiEtAl07, AspertiEtAl07a] is an tactic–based, interactive prover meant as an interface between the user and the mathematical knowledge library of HELM. It also provides authoring facilities and a user interface for browsing, adding mathematical knowledge, indexing and searching mathematical objects in the HELM library. For proving a certain goal, the system automatically retrieves the appropriate assumptions from the existing knowledge base. The retrieval is based on the signature and context of the goal. The basic automation tactic in Matita is the **`auto`** tactic, which boils down to an iterated use of **`apply`** (modus ponens). Other tactics used by the prover are **`intro`** (∀–introduction), **`elim`** (induction), **`cut`** (forward reasoning) and **`simplify`**.

A recent extension of the prover is the addition of the paramodulation tactic: taking an equational goal and using a given-clause algorithm [NieuwenhuisRubio01], Matita returns all known equational facts from the library (and the local context). For more details see [AspertiTassi07].

The associated project [MOWGLI] has developed system– (and reasoner–) independent tools for presenting mathematical information on the Internet.

2.3.5 HR

HR invents concepts and theorems in mathematical domains like finite algebras/groups/rings, graph theory and number theory [ColtonEtAl02]. The underlying idea is the formation of new conjectures, which are then passed to a resolution theorem prover (usually Otter [Otter]): If the proof goes through, the conjecture is adopted as a theorem; otherwise, a model finder (often Mace [Mace]) is called for drawing counterexamples from it.

Each concept is an operation or a relation on the carrier set of a certain model of interest (e.g. a fixed finite group), represented by a combination of a “data table” (also called Cayley table in the case of groups) and a formula with an indication of its free variables (the “arguments” of the operation or relation). For example, the concept of multiplication in $\mathbb{Z}_6$ is described by a table containing rows like **`[4,2,2]`** and **`[3,3,5]`** together with the formula $[n, a, b] : a \mid n \wedge b \mid n \wedge a \times b = n$.

Concept formation works as follows:

- ***Initial concepts*** are given axiomatically (like the multiplication in a group) or by an explicit definition (like divisibility in number theory). The corresponding data tables are either supplied by the user as key examples or generated by MACE.
- ***Production rules*** [BundyEtAl98] are applied to the existing concepts. For example, universal quantification (with respect to the first argument, say) produces an **(n-1)**–ary predicate from an **n**–ary one.
- ***Measures of interestingness*** are used for ranking the concepts on a heuristic basis [ColtonEtAl00], for instance "parsimony" by measuring the relative size of its data table. Note that these measures are recomputed at various stages in the concept/theory formation cycle.
- ***Filtering*** the eventually accepted concepts works by computing a total score computed from various measures of interestingness; these are subsequently used for creating conjectures and new concepts.

Examples of concepts invented in group theory include: Abelian group, cyclic group, order of elements, central elements.

Theory formation [ColtonEtAl02, ColtonEtAl06] is built on top of concept formation. HR takes the newly obtained concept and forms conjectures about it:

- If the set of models is empty, a ***non–existence conjecture*** is made (i.e. the concept definition is inconsistent with the axioms). For instance, the concept of non-trivial idempotent elements in groups does not have a model, thus the conjecture $\nexists (a \neq id \wedge a * a = a)$ is generated.
- If the sets of models coincide for the newly created concept and an already existing one, an ***equivalence conjecture*** is made. In the case of idempotent elements, HR notices that for all group examples it has, the idempotent elements are the identity, thus generating the conjecture $a * a = a \leftrightarrow a = id$.
- If neither of the above two cases applies, HR adds the concept to the theory and looks for ***subsumption conjectures***. These are asserted about concepts whose set of models is contained in the set of models of the new concept. For each such case, HR conjectures that the definition of the new concept is implied by the definition of an old one.

The theories produced in this fashion will include concepts, theorems that relate concepts and proofs of these theorems.

Improvements of HR allow using Otter as a filter for newly obtained theorems [Colton02], inputting initial concepts in Maple format, and substituting Otter and MACE by humans. HR is currently connected to the MathWeb software bus [ZimmerEtAl02].

2.3.6 MathsAid

MATHsAID [McCaslandBundy06] is a system for ***automated discovery*** of "interesting" mathematical theorems. It automatically discovers and proves theorems (lemmas, corollaries, etc.) from a given set of axioms and definitions (supplied by the user). Its main components are the automatic hypothesis generator, the theorem generator and the theorem filter [McCaslandBundy06].

Given the set of axioms and definitions, the ***hypothesis generator*** (HG) builds up:

- a finite sequence $(H_i)_{i=1 \ldots n}$ where H_i is a set of hypotheses and
- a selection of axioms and terms of interest corresponding to each H_i.

This allows the system to concentrate on local aspects of the theory, and thus (in the following components) to build theorems in layers, rather than build them all at once. The HG also looks at the existing axioms and theorems for finding converses. If such a converse can exist, it is passed over to the theorem generator that will attempt to prove it.

The ***theorem generator*** takes as input a set of hypotheses H_k, asserts them and applies a forward—chaining process, using the build—in first—order theorem prover and deriving all conclusions. Conclusions satisfying certain criteria are then asserted, the process starts over, until no more new conclusions can be found. The resulting conclusions are passed over to the theorem filter.

The most important criteria a conclusion has to satisfy is that it should not be a conclusion of an axiom, the "trivial" conclusion of a theorem or "trivially derived" from the existing knowledge (axioms, theorems). A more detailed description of triviality is found in [McCaslandBundy06].

The ***theorem filter*** takes these conclusions, and runs them through a number of tests, like irredundancy (checking whether all hypothesis from H_k are needed to prove the conclusion) or simplicity (based on a heuristic detailed in [McCaslandEtAl2006]). All conclusions failing a test are eliminated. All remaining conclusions are coupled back with H_k, and recorded as theorems.

The system was recently extended towards automated discovery of inductive theorems [McCaslandEtAl07].

2.3.7 MathWebSearch

MathWebSearch is an online ***search engine*** for mathematical expressions developed by the KWARK team at Bremen [KohlhaseSucan06, KohlhaseSucan07]. It contains:

- knowledge bases obtained by harvesting the internet for content representations of mathematical expressions (currently the OpenMath and the content MathML formats are supported),
- an indexing mechanism using substitution trees [Graf96] that creates index terms for all mathematical expressions (i.e. for each math element in MathML and more or less for each apply element in OpenMath) and

- a query language for mathematical expressions, realized as a generic extension mechanism for XML–based representation formats.

The repositories of mathematical knowledge that are harvested include the ActiveMath repository mentioned above, the [CONNEXIONS] corpus and the MBase system mentioned below. The mathematical expressions are interpreted by the indexing mechanism as first–order terms.

The ***query language*** is obtained by adding new attributes and tags to the XML–based languages. For example, the attribute **`mq:generic`** is used for matching any subterm in the index, the attribute **`mq:anyorder`** for specifying an arbitrary order of the parameters of the current node.

The user can input the desired search query in OpenMath or MathML format. In the former case, one may also take advantage of the [WIRIS] OpenMath Editor (a user-friendly Java plug-in for editing mathematical expressions).

The standard ***term retrieval algorithm*** for substitution trees [Graf94] is used as a search algorithm. The system can find an expression (occurring as a subexpression in any statement) up to α-conversion by adding the **`mq:generic`** attribute to every bound variable in the search query. Searches in informal text are not possible.

A related OMDoc tool also developed by the KWARK team is a ***semantic wiki*** (a wiki with Semantic Web Capabilities) called SWiM [Lange08].

2.3.8 MBase

The system [MBase] is a structured and distributed repository of mathematical knowledge [KohlhaseFranke00]. It is implemented as a ***mathematical service*** of the MathWeb mathematical software bus, which is in turn built on top of OMDoc. MathWeb connects a wide range of mathematical services (including interfaces to Ωmega, Bliksem, Otter, SPASS, RDL, Mace, λ–Clam, Maple, CoCoA).

The online demo of MBase contains several preloaded theories. These are imported from the OMDoc libraries of Omega and TPS, and from OpenMath content dictionaries in OMDoc form.

The demo allows browsing through the theories and ***retrieval facilities*** like

- textual search on all mathematical elements available in the knowledge base (called QuickSearch),
- textual search restricted to names of theories, axioms, symbols, definitions and theorems,
- pattern search for terms and formulae. The pattern matching mechanism (applicative higher-order) is inherited from Oz, the underlying programming language. Pattern matching will be performed on the internal representation of terms and formulae (i.e. on variable-free tuple trees). E.g. **`foo(a b c)`** will match **`foo(X b c)`** but not **`foo(_ _)`** or **`foo(X b X)`**. Note that in Oz the variables start with a capital letter (**`X`**,**`Y`**,**`Z`**), and the constants with lower case letters (**`a`**, **`b`**, **`c`**, **`foo`**).

Some ***query examples*** are:

- Searching for regular expressions ending with “group”:

```
s (name : regexp (".group$"))
```

- Searching for the commutativity law *F(x, y) = F(y, x)* is done by

```
eq (a (F X Y) a (F Y X))
```

which is a shorthand for the expanded form

```
a (s (name : regexp (' ^= $) | (^[Ee][Qq])' )) [a (F [X Y]) a (F [Y X])]
```

As announced in [KohlhaseFranke01], there are plans to improve the system by adding inference procedures like higher-order matching and (logic-internal) structuring mechanisms like theory morphisms.

2.3.9 Mizar

MIZAR strives to transform mathematical ideas into journal articles that are readable both by humans and machines, warranting consistency of the definitions and correctness of proofs [RudnickiTrybulec01]. The mathematical objects in Mizar are theorems, definitions and schemes (i.e. theorems with second-order variables).

The main components of the system are the Mizar Verifier and the Mizar Mathematical Library (MML). The Mizar Verifier takes a ***Mizar article*** (consisting of definitions, theorems/schemes and proofs, written according to Mizar syntax) as input and checks the file for logical errors [Wiedijk08]. It is based on classical first-order logic with the added capability of forming second-order schemes.

MML is the largest library of ***formalized mathematics*** available in the world at this time. It aims at the reconstruction of the core of mathematics. Built on the axioms of Tarski-Grotendieck set theory, its current version includes 942 articles, written by 182 authors, comprising 42694 theorems, 7957 definitions, 728 schemes, 6942 registrations, 5879 symbols, and 1903 keywords. MML articles contain mathematical theorems like Birkhoff's Variety Theorem, the Fundamental Theorem of Algebra, the Fundamental Theorem of Arithmetic, Gödel's Completeness Theorem, the Jordan Curve Theorem, the Reflection Theorem, Rolle's Theorem.

The ***Mizar Proof Advisor*** [Urban05a, Urban06] accepts as input a complete formula and outputs a list of theorems that could help in proving the input formula, sorted by their expected relevance. It is based on the machine learning toolkit SNoW [CarlsonEtAl99], which implements methods (like neural nets, Markov models, bayesian nets, etc) used for processing natural language.The proof advisor extracts features out of each Mizar formula (for now only the signature of the formula) and guesses the relation between these features and the theorems/definitions used in the proof of the formula. Based on these guesses, the system returns theorems and definitions that could be relevant for the user input. The best results were obtained using naive bayesian nets.

MMLQuery [BancerekRudnicki03] is a fast ***query language*** for MML. A set of indices (which is their version of metadata) is extracted from the library, and the user can retrieve a theorem by combining various filters on these indices. Some examples are:

- Getting the names and links to all articles containing **`NAT`** in their identifier: **`list of articles | grep-i NAT`**;
- Obtaining all theorems referring to the mode **`GROUP_2:mode 1`**: **`list of th from (GROUP_2:mode 1 article)`**;
- Items which are neither notations nor registrations: **`list of item |[not notat and not reg]`**;
- Finding all items referring to all constructors from the theorem labelled **`FUNCT_1:70`**: **`at least * (FUNCT_1:th 70 ref)`**.

Most of Mizar Matches (MoMM) is another search tool for mathematical databases, optimized for Mizar [Urban05]. It is based on Stephan Schulz's prover E. Its main goal is to eliminate redundant formulae. The tool translates parts of the MML into its own clausal-like format and eliminates the ones that can be subsumed by the others, using a kind of typed subsumption.

MizarMode [Urban06] was developed as an Emacs authoring environment for Mizar. It provides proof assistance functions by integrating the functionalities of MMLQuery, MoMM and the Mizar Proof Advisor.

Alcor [Cairns04] is a graphical user interface usable for Mizar. It provides a textual editor together with a search tool for keywords. The search can be performed by highlighting words or by entering text.The list of search results will contain the location and the type of the items found.To see the actual definition, the user clicks on the location, and the required item is displayed, as given in an Mizar abstract.

The ***Journal of Formalized Mathematics*** (available online at http://www.mizar.org/JFM) gathers the articles accepted in MML. It is a compromise between a formal–language and natural–language postprocessor. It has been published since 1989 and is now available on the web. MML articles can be exported to Prolog–based formats for theorem proving via the MPTP system [Urban05a]. Importing facilities for parts of Mizar are provided by ISAR and HOL.

2.3.10 Outlook

In the seven years that have passed since MKM was initiated at the 2001 conference, one of its main threads was concerned with the ***representation*** of mathematical knowledge for communicating it between systems or for presentation on the web. For searching in such knowledge repositories, textual search turned out to be insufficient, as already noted in [Buchberger01]: more and more systems evolve via pattern matching and unification towards simple instances of proving.

Another main thread is the question of ***integrating*** theorem provers and computer algebra systems, as promoted and pursued in the Calculemus community. Also older repositories of mathematical knowledge like Mizar are being incorporated into MKM systems. Such systems should provide an integrated environment for storing, accessing and using mathematical knowledge. While this is partially realized for special-purpose fields and math education, an overall tool for the working mathematician is yet to be developed.

The vision of such systems is not far–fetched, however. There are efforts for standardizing the representation of mathematical expressions on the web, for facilitating distributed communication of mathematical information between systems with moderately different logical frameworks and different languages. The difficulties of ***searching*** mathematical information become clearer and more explicit.

There are still several ***challenges*** that MKM has to overcome. The communication between groups working in MKM is made difficult by differences of terminology (e.g. terms like “semantics” and “retrieval” have different meanings in different groups). The repositories of formalized mathematics (still) cover very little from the vast domains of mathematics.

2.4 The *Theorema* System

2.4.1 General Features

Theorema is distinguished among today's mathematical assistants by its design principle of supporting the ***entire process of mathematical work*** [BuchbergerEtAl06], going along the “creativity spiral” [Buchberger93]:

- creating new notions and ***conjectures***,
- turning the conjectures into ***theorems*** by computer–supported verification (using automated proofs as much as possible)
- extracting ***algorithms*** (or at least algorithmic parts) from proofs,
- running the algorithms on suitable input data for producing relevant new ***facts***, which may in turn lead to new conjectures.

Theorema is an environment for writing papers and doing proofs/computations at the same time. It takes advantage of the two–dimensional syntax of *Mathematica*, facilitating thus a natural formalization for real–life mathematics. Catering the working mathematician, its goal is to support their ***routine tasks*** rather than “extraordinary tasks” like proving conjecture defying human attack. Various simplifiers, solvers, and provers from predefined libraries can be applied (see the detailed description in Subsection 4).

Though not often addressed in the automated proving community, the invention of new notions and theorems is a crucial component of the creativity spiral, and *Theorema* strives to accommodate these needs by a methodology named ***theory exploration*** [Buchberger00]. The SysteMathEx subproject carries out case studies of systematic explorations of mathematical domains (e.g. the theory of natural numbers [CraciunHodorog07]). For supporting such a theory exploration, one can employ a particular kind of meta–reasoner called **`Cascade`** (Subsection 4) and the explorations commands of the language of archives (Section 4.4).

By and large, *Theorema* encourages the ***natural style*** of working mathematicians. On the inference level, this implies a preference of natural deduction provers over machine oriented calculi like resolu-

tion (this is also important for extracting useful information from failed proofs by the **Cascade**). Regarding notation, it inherits from *Mathematica* all the type–setting facilities like the afore–mentioned two dimensional syntax. The concrete syntax can be changed easily by the user, without affecting the internal abstract syntax. Since the proofs generated by *Theorema* use natural language intermediate texts (generated automatically from the underlying formal proof object), they are presented in a style coming close to that of mathematical textbooks. Stored in a Mathematica notebook, such a proof can be viewed at different levels of detail by expanding or collapsing the cells containing the hierarchical proof structure. Being an interactive system, *Theorema* generally promotes theory exploration rather than the restricted paradigm of isolated theorem proving.

By using ***predicate logic as a programming language***, *Theorema* allows to describe, formulate and use algorithms as an integral part of the language. Proving and computing are thus integrated in a single framework (following in particular the Calculemus slogan of bridging Computer Algebra and Automated Theorem Proving). The crucial observation is that the fragment of predicate logic consisting of inductive function definitions and quantified formulae with bounded ranges forms a universal programming language. While many existing systems employ different languages for programming and specifying its requirements, such an integrated approach brings together algorithmic mathematics (like "symbolic computation") and inconstructive mathematics (like the structural approach of Bourbaki). Regarding proof presentation, this allows to realize the so–called Poincaré principle [Barendregt97, Gonthier08] of combining low–level inference steps in a few high–level computation steps embedded in proofs.

Generally speaking, *Theorema* allows arbitrary *Mathematica* notebooks mixing informal text, graphics, and formal input. In standard sessions (see Subsection 4), the user may continue to apply any *Mathematica* command, but certain inputs will be recognized as parts of the *Theorema* language. The latter consists of the following three ***language layers*** [BuchbergerEtAl00] described in the subsequent subsections:

- The ***expression language*** consists of *Theorema* terms and formulae (collectively known as expressions), described in Subsection 2.
- The ***formal text language*** allows to combine formulae with logic–external labels in environments like **Definition** and **Theorem** similar to the usual practice in mathematical papers; see Subsection 3.
- The ***user language*** contains commands for calling reasoners and for manipulating the global knowledge base, to be presented in Subsection 4.

2.4.2 Syntax of *Theorema* Expressions

Since the underlying logic of *Theorema* is a variant of higher–order predicate logic, its expressions (formulae and terms) follow the usual abstract syntax of predicate logic. The only noteworthy extension is the integration of so–called ***sequence variables*** [BuchbergerEtAl97, Kutsia02], denoted by overbars. They are very practical e.g. for defining operators of flexible arity like addition of natural numbers:

$$\text{add}[] = 0$$

$$\forall_{\bar{n}} \text{add}[0, \bar{n}] = \text{add}[\bar{n}]$$

$$\forall_{n} \forall_{\bar{n}} \text{add}[n^{+}, \bar{n}] = \text{add}[n, \bar{n}]^{+}$$

Note that in the above example the individual variable **n** is considered distinct from the sequence variable **n̄**. Sequence variables come with a natural induction principle. For proving that some formula holds for all **n̄**, it suffices to prove the formula with **n̄** replaced by the empty sequence and to prove the following induction step: Whenever the formula holds for **n̄**, it must also hold when the sequence **n̄** is prolonged by an arbitrary element **n**.

Passing from the abstract to the concrete syntax, *Theorema* offers a wide range of attractive notational features that make formalized texts look close to the style used by the working mathematician. It inherits the ***two–dimensional syntax*** provided by *Mathematica*, offering the usual notation not only for the logical quantifiers ∀ and ∃ but also for various special quantifiers like ∑, ∏, ⋂, limits of sequences (where the index **n** of a sequence is bound by the construct **n→∞** in its underscript).

In *Theorema*, the usual ***parsing and formatting*** rules of *Mathematica* are adapted in various ways [BuchbergerEtAl00], for ensuring that the *Theorema* expression language is kept apart from the *Mathematica* syntax. Translating the box structure (concrete syntax) into the internal **FullForm** representation (abstract syntax), the parser is realized by the *Mathematica* function **MakeExpression**. In the other direction, one invokes the formatter by the function **MakeBoxes**. Consider the following example of a *Theorema* expression:

$$\forall_{n} (n^2 \geq 0)$$

This is how its box structure looks like:

```
RowBox[{UnderscriptBox["∀", "n"], RowBox[{"(", RowBox[{SuperscriptBox["n", "2"], "≥", "0"}], ")"}]}]
```

The *Theorema* parser will then create the following internal representation from it (visible by wrapping the expression into the special symbol • and displaying the outcome in **InputForm** or **FullForm**):

```
™ForAll[•range[•simpleRange[•var[n]]], True, ™GreaterEqual[™Power[•var[n], 2], 0]]
```

One can see that this representation differs from what the *Mathematica* parser produces, due to changes in certain symbols like ™GreaterEqual in place of GreaterEqual, etc. This is important to prevent *Mathematica* from interfering with the *Theorema* reasoners (provers / computers / solvers) when evaluating the expressions. The logical integrity of the *Theorema* system depends on the important principle that no knowledge other than that explicitly specified by the user can enter into proofs. Note that the archive I/O commands **LoadArchive** and **SaveArchive** are respectively extensions of the *Theorema* parser and formatter (see Section 4.1).

For specifying and manipulating model classes like the rings mentioned in the introduction, *Theorema* encourages the usage of so–called ***categories and functors***. 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). While their standard usage in *Theorema* is explained in Subsection 3.1.2 and their extended support in archives in Section 3.5, let us only make the following observation at this point: The keyword **`Functor`** provided by *Theorema* is syntactic sugar for the native description quantifier $\exists_{x}$ of higher–order logic ("such an **x** that …").

Beyond the usual formulae making up the bulk of mathematical theories, a special type of externally stored knowledge can be specified in various ***schemes***, including definition schemes, algorithm schemes, theorem schemes, and problem schemes. They are considered as expressing the accumulated experience of mathematicians for inventing definitions, algorithms, theorems, and problems [Buchberger04]. For instance, an algorithm scheme is a generic pattern for introducing an algorithm in terms of some ingredient notions (functions / predicates / constants). Unlike a concrete function definition, a scheme uses higher–order variables for denoting the ingredient notions. An example is provided by the **`Divide–Conquer`** scheme in Section 4.4.

2.4.3 Logic–External Structuring Mechanisms

As opposed to the logic–internal structuring mechanisms provided by the language of archives (Chapter 3), *Theorema* offers ***logic–external*** labels for individual formulae as well as environments like propositions, definitions and nested theory constructs. Alternatively, it supplies a suite of organizational tools for label management [PiroiEtAl07]. One may view the language of archives as a continuation of these functionalities, embedding the organizational tools into the object language.

In mathematical textbooks, mathematical formulae are classified as definitions, axioms, theorems, and the like; they are furthermore grouped into a hierarchy of mathematical theories. In *Theorema*, we simulate this by so–called ***environments*** [BuchbergerEtAl00]. Its general form is:

```
envname[envlabel, any[x, y, z, …], with[cond],
  stat1   statlabel1
  stat2   statlabel2
    ⋮        ⋮
]
```

The identifier **`envname`** indicates the type of formula desired. By inspecting the global parameter **`$TmaEnvironmentPatterns`**, one can see the possibilities provided by standard *Theorema*:

```
"Definition" | "Definitions" | "Theorem" | "Theorems" | "Lemma" | "Lemmata" | "Axiom" | "Axioms" | "Corollary" |
  "Corollaries" | "Proposition" | "Propositions" | "Theory" | "Theories" | "KnowledgeBase" | "KnowledgeBases" |
  "Algorithm" | "Algorithms" | "Assumption" | "Assumptions" | "Formula" | "Formulae" | "System" | "Systems"
```

The environment label **`envlabel`** is used for referring to the whole environment e.g. in specifying goal or assumptions in a prover call. The statements **`stat1`**, **`stat2`**, … are formulae of the Theorema expression language (see Subsection 2); they are given formula labels **`statlabel1`**, **`statlabel2`**, …for identifying the individual statements e.g. in the course of a proof. Hence each

statement is uniquely identified by a composite label consisting of the environment label followed by the formula label.

The **any[x,y,z,…]** specification designates **x**, **y**, **z**,… as the free variables in the statements, while the **with[cond]** specification imposes the condition **cond** on them. Note that these two specifications are logically equivalent to prepending a universal quantifier on **x**, **y**, **z**,… with relativization **cond** to the conjunction of the statements. The difference is of a purely stylistic nature, paralleling the distinction between textbook formulation like “for all rings **R** we have” followed by a formula versus the corresponding universally quantified formula.

Here is an example from group theory (note that we have used the capital letter O instead of zero because numerals are treated specially in *Theorema*):

```
Proposition["Uniqueness of Inverse", any[+, O, −], with[is–ab–group[+, O, −]],
  ∀x,y ((y + x = O) ⇒ (y = −x))    "left"
  ∀x,y ((x + y = O) ⇒ (y = −x))    "right"
]
```

In the same fashion, one would define new notions by **Definition**, state a theorem by **Theorem**, formulate an algorithm by **Algorithm**, etc. The distinction between these different environment names is only offered for stylistic reasons; in the reasoners (provers and simplifiers), they are all treated alike.

For building ***logic–external hierarchies*** of formulae, one employs the **Theory** construct. In connection with the above example, one might have a theory containing—besides this proposition—plain group theory and the definition of an abelian group:

```
Theory["Basic Abelian Group Properties",
      Theory["Groups"]
    Definition["AbelianGroup"]
  Proposition["Uniqueness of Inverse"]
]
```

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 one often has to repeat essentially the same name three times: Once in the section heading, then in the label of the environment, and finally in the formula label. In the language of archives, we overcome this problem by using logic–internal labels (see Section 3.3); a logic–external alternative is provided in *Theorema* by a toolbox called ***Label Management*** for organizing mathematical libraries [Piroi04, PiroiBuchberger04]. These libraries can be edited, browsed and used in proving and computing. A mathematical library is stored in a special type of *Mathematica* notebook, based on the **Theorema`Theory** stylesheet; through this stylesheet, cells and cell groups in the library are marked, allowing the notebook to be parsed into *Theorema* environments. The command used for loading the label management tools and for opening a toolbar containing several operations on mathematical libraries is **OpenLibraryUtilities[]**.

The user can create a new mathematical document based on a template. The notebook created typically has a title like **BasicNotions:Tuples** and a notebook label, given by the user or automatically created from the notebook title. The mathematical formulae are given in input cells and grouped hierarchically under section and subsection headings. These headings are subsequently used for automatically creating unique ***composite labels*** that reflect the hierarchy of the formulae as well as that of the sections. For example, imagine the notebook has a subsection **Definition** within a section **Concatenation**. Then it will automatically get the composite long label **Basic**␣**Notions:Tuples.Concatenation.Definition**, and the formulae right underneath it will be labelled by appending a counter to this label, so that the first formula would have the label **Basic**␣**Notions:Tuples.Concatenation.Definition.1**. The same subsection will internally also receive a segmented decimal label (representing the position of the subsection cell within the cell hierarchy, like **7.1.4**) as well as a short label generated from the long one by a string truncation algorithm (preserving uniqueness of labels, like **BN:Tuples.Conc.Def**).

Existing formalized knowledge from other libraries can be embedded in the new one by using the **Include** command. This inclusion can be static, i.e. the fragment desired will be copied in the new library, or dynamic, i.e. only a link to the old library will be set up so that when the formula in the source library is modified, also the new one will contain the new formula.

At any moment, the user can browse the existing collection of mathematical libraries, by using the toolbar option **Library Contents**. For each library, the user can also inquire its content type, e.g. asking what are the axioms in a given library or searching for labels that contain a given substring. The system returns the list of labels of the corresponding formulae in the respective theory. Simple retrieval operations are provided for searching in the library; see Section 5.4.

After its creation, the mathematical library can be parsed by using the toolbar button **Process Documents**. This generates the labels for formulae and headings, checks whether the inclusion of foreign libraries produces a cycle, and finally applies the *Theorema* parser on the formulae in the library. Based on the annotations of the library, this creates the corresponding *Theorema* environments, which are then written to a *Mathematica* package file. Loading this file enables *Theorema* reasoners to use the library for reasoners.

2.4.4 Proving, Computing and Solving

Arguably, the three ***fundamental activities*** practiced by mathematicians are proving, computing, and solving [BuchbergerEtAl00]. Accordingly, *Theorema* offers the three corresponding commands **Prove**, **Compute** and **Solve**. Their general form, exemplified by the **Prove** command is:

```
Prove[Proposition["Uniqueness of Inverse"],
  using → ⟨Definition["AbelianGroup"], Theory["Groups"]⟩, by → PredicateProver]
```

In the example above, the goal formula to be proved is the proposition on the uniqueness of inverse mentioned in the previous subsection, with the assumption list consisting of the definition of abelian groups and general group theory. The prover to be invoked is a generic prover for predicate logic. Similarly, in a **Compute** call, the first argument is the expression (term or formula) to be

computed/simplified. (Normally we do not distinguish between the terms “compute” and “simplify”, although the former is sometimes restricted to ground terms.) The option **using** has the same meaning as before, while **by** specifies the simplifier to be applied. The **Solve** command is currently not provided in *Theorema;* it is planned for future versions.

In addition to the assumption lists given explicitly in each call of a reasoner (meaning: prover, simplifier or solver), the user can define a ***global knowledge base*** by the command **Use**, for example by calling **Use[Definition["AbelianGroup"]]**. In a subsequent proof call, the definition of abelian groups is then automatically used. New global knowledge is added by the command **UseAlso** in the same fashion. For both commands, one may also supply a list of environments like in the **using** option of the reasoners. The global knowledge base can be emptied by **Use[]**, and one can see its current contents by saying **Info["Knowledge Base"]**.

One of the basic principles of *Theorema* is that one should not attempt to build up all of mathematics with a single, monolithic prover. (The same holds for simplifiers and solvers, where this is even more obvious: Who expects solving algebraic equations, nonlinear differential equations, integral equations and the like by a single method?) This is in harmony with the general *Theorema* philosophy of following the style of the working mathematician: Different branches of mathematics come with their ***individual proof techniques***, which are in general restricted and therefore well–adapted to special theories.

In *Theorema*, a whole suite of ***special provers*** are therefore available. For example, the PCS prover [Buchberger01b] focuses on theorems in real analysis, relying on constraint solving in real–closed fields. Other special provers are induction provers on lists and natural numbers, the set theory prover, etc. For dealing with combinatorial identities, the Gosper–Zeilberger prover can be used, whereas for tackling universally quantified boolean combinations of arithmetical equalities over the complex numbers, one can apply the Groebner–Bases prover. Nevertheless, *Theorema* also provides some universal provers (i.e. assuming no special knowledge, but as a rule not aiming at completeness), e.g. the **PredicateProver** [BuchbergerEtAl00] based on natural deduction and the **EquationalProver** [Kutsia03] implementing unfailing Knuth–Bendix completion.

Special provers often come as ***black–box provers***—they use a low–level computation as a proof step in a high–level proof (thus realizing the Poincaré principle mentioned in Subsection 1). For example, this is how the cylindrical algebraic decomposition (CAD) method is considered as a single step in a PCS proof. A black–box prover thus typically acts as a decision procedure; the goal formula is proved/disproved (unless the method fails), but no proof details are given. In other cases, like in the CAD method, the goal formula is transformed into an equivalent but simpler formula, again without further details. An extreme case of a black–box prover is also provided by the interface [KutsiaNakagawa01] to external systems that links *Theorema* to provers like Vampire [RiazanovVoronkov99] or model checkers like Mace [McCune03]; also in this case, only the result is transmitted (besides a textual trace of the execution).

At the beginning of each proof, the function **FlattenKB** is used for transforming the nested **Theory** constructs into a flat list of assumptions represented internally as an expression having the head **•asml**. Each ***prover package*** is implemented as a set of inference rules, typically expressed as rewrite rules transforming a proof situation (also called a sequent: a pair consisting of a goal formula and a list of assumption formulae) into one or more new proof situations. Such basic provers can be

combined in various ways to form more powerful ones, the so–called user provers.

Proofs are internally represented by ***proof objects***, containing the complete history of inference rule applications. As the prover continues, the proof object is expanded from the initial proof situation to a tree representing the full proof (in case of success). At certain points, a proof situation is split into two, and the prover continues, either by trying to prove both newly created proof situations (e.g. when proving a conjunction), or one of them (e.g. when proving a disjunction). The final proof object contains also branches with failed attempts, which may be useful for the human reader (in particular for the natural deduction provers usually employed in *Theorema*) as well as for automated conjecture generators.

A case in point is the **Cascade** [Buchberger02], a method for analyzing failing proofs and extracting general conjectures from it. It takes as arguments a prover and a conjecture generator. Its call results in the automated generation of new proof attempts; the initial assumption list (given to the prover) is recursively enlarged with lemmata—automatically generated and proved in the course of failure analysis—until the proof succeeds.

As mentioned before, the command **Compute** can be used for computing and simplifying expressions in *Theorema.* In the same way as for **Prove**, the knowledge base used has to be specified explicitly; this provides complete flexibility in attaching meaning to expressions. For instance, consider the definition of rationals given below:

Definition["Basic Rationals",

Q = Functor[F, any[c, d],

$$\$ = \langle 0 : F,\ + : F\times F \to F,\ - : F \to F,\ - : F\times F \to F,\ 1 : F,\ * : F\times F \to F,\ / : F\times F \to F\rangle$$

$$\in_F[c] \Longleftrightarrow \text{IsRational}[c]$$

$$0 =_F 0$$

$$1 =_F 1$$

$$c +_F d = c + d$$

$$-_F c = -c$$

$$c -_F d = c - d$$

$$c *_F d = c * d$$

$$c /_F d = c / d$$

]]

Computing different operations on rationals is then performed in the following fashion:

Compute$\left[\frac{2}{3} +_Q \frac{67}{9}, \text{using} \to \textbf{Definition}[\text{"Basic Rationals"}]\right]$

$$\frac{2}{3} + \frac{67}{9}$$

Note that the + is internally represented as a ™Plus, so *Mathematica* does not evaluate it further. But often one desires various degrees of ***interoperability*** with *Mathematica.* This is realized by so–called built–ins, which assign interpretations to certain symbols (e.g. the symbol * can be inter-

preted as the *Mathematica* * symbol—represented internally as Times). For the example above, assigning for the arithmetic operations their corresponding *Mathematica* functions is realized by:

```
Built–in["MmaOperations",
    ™Plus → Plus
  ™Minus → Minus
  ™Times → Times
 ™Divide → Divide
]
```

Now, computing the sum of the two rationals given above will now yield:

```
Compute[2/3 +ℚ 67/9, using → Definition["Basic Rationals"], built–in → Built–in["MmaOperations"]]
```

$$\frac{73}{9}$$

For situations when the user wants to have an accumulating knowledge base for a series of computations (this is the usual behavior of computer algebra systems like *Mathematica* or *Maple*), a so–called computational session can be set up. In such a session, it is assumed that every new definition, axiom, etc. is added to the global knowledge base, and a (universally valid) standard simplifier is applied to the expression entered into an input cell. Moreover, the formulae do not need the environments described in Subsection 3. The command used for invoking a computation session is **`ComputationalSession`**. It takes as first argument the name of the session, and imports knowledge from the standard session by the option **`using`**. For example, working with the rationals defined above, a computational session looks as follows:

```
ComputationalSession["RatOper", using → ⟨Definition["Basic Rationals"], Built–in["MmaOperations"]⟩]
```

```
2/3 +ℚ 67/9
```

$$\frac{73}{9}$$

```
any[x, y] :
        (mean[x, y] := (x +ℚ y) /ℚ 2)
```

```
mean[1, 2]
```

$$\frac{3}{2}$$

```
EndComputationalSession[]
```

Computational sessions are practical in particular in the experimental phase of the mathematical exploration cycle (see Subsection 1).

3 Mathematical Knowledge Archives in *Theorema*

In this chapter we come to the detailed description of the language of archives. The ***structure of the chapter*** is as follows. After motivating the usage of labels, we introduce the notions of categories and functors, and we explain the concept of namespaces in Section 3.1. Archives and basic commands for manipulating them are discussed in Section 3.2. Next we present various useful notational conventions for already existing *Theorema* constructs: conjunction, universal quantifiers, existential quantifiers, substitution quantifiers, and description quantifiers. In Section 3.3 and 3.4, we introduce labels and namespaces together with various concepts relating to them. Labels are attached to hierarchical blocks of formulae, which may contain global and local symbols as well as symbols residing in a specified namespace. Finally in Section 3.5, we work out the relations between namespaces and the domains of categories and functors.

3.1 Crucial Issues in Formalizing Large Repositories

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 [Piroi04, PiroiBuchberger04] or the metadata of [LibbrechtMelis06] 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 [HosnAndersen05]. 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 [MillerYoussef03] 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.

A last remark on terminology. We use the term “symbol” for the syntactic entities referring to functions, predicates or constants (i.e. “function constants”, “predicate constants” or “object constants”).

3.1.1 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 Section 3.3 and namespaces in Subsection 3.4.2.

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 3.2.1), we try to give an intuitive understanding beforehand. In *Theorema*, the definition of a real vector space would look as follows:

Definition["RealVectorSpace", any[V],

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

Underscripted operation symbols like $\underset{V}{+}$ 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:

$$\underset{V}{\forall}$$

$$\text{is–vecspace}[V] \Leftrightarrow$$

$$\underset{x,y,z\in V}{\forall}\ \underset{\lambda,\mu\in\mathbb{R}}{\forall}$$

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

$$x \underset{V}{+} 0 \underset{V}{=} x$$

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

$$x \underset{V}{+} y = y \underset{V}{+} x$$

$$\lambda \underset{V}{\cdot} \left(x \underset{V}{+} y\right) = \lambda \underset{V}{\cdot} x \underset{V}{+} \lambda \underset{V}{\cdot} y$$

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

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

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

Actually we can make it even a bit more readable if we open **V** as a namespace (see Section 3.5):

$$\underset{V}{\forall}$$

$$\text{is–vecspace}[V] \Leftrightarrow$$

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

$$\underset{x,y,z\in V}{\forall}\ \underset{\lambda,\mu\in\mathbb{R}}{\forall}$$

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

$$x + 0 = x$$

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

$$x + y = y + x$$

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

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

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

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

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 3.3, 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 Appendix). 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.

3.1.2 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 3.1.1.

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, −, ·] ⇔ …

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 Pol : R ↦ R[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 **Pol:** $\mathcal{Rng}\rightarrow \mathcal{Rng}$. 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 (≈ 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 3.1.1) 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 here, 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, Buchberger01a], 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 Section 3.5). For example, the operation object corresponding to a ring **R** encompasses the following components:

Operation object	R
Carrier	R[ϵ]
Addition	R[+]
Zero	R[0]
Negative	R[−]
Multiplication	R[$*$]
One	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 language (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 indicated below.

This notation does not take us out of higher–order logic; the expression **`Functor[D,…]`** is just an abbreviation for the description quantifier $\ni_{D} \ldots$ 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 Section 3.5).

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

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

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

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

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

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

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

]]

3.1.3 Symbols with Multiple Meanings in Mathematics

In mathematics, some symbols are used in different contexts and for different purposes. Such ***ambiguous symbols*** are ubiquitous, 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 occurrences of **`0`** refer to the zero module, the third to the zero element in a quotient module, the next two occurrences to the natural number **`0`**, the last before the exercises to the zero element in another module. The second example exhibits **`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 3.1.2. 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 unambiguously 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 Section 3.5.

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 3.4.2), 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 Section 3.5.

3.2 The Coarse Structure of Archives

An archive is a single formula in an extension of the *Theorema* language (see Section 2.4). 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 3.3 and Section 3.4. 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 ***mathbook***, 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 **`FormalX`** 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 Section 3.3) occurring in the mathbook. This will restrict parsing to the first cell group containing the keyword.

As an example consider loading the "subsection" entitled **`BasicProperties`** of the previous mathbook:

```
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 mathbook).

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 annota-

tions: the annotations can be translated to logic but the resulting formula blurs the distinction between "logic" and "organization". The translation command

```
ArchiveToTma[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 here we discuss only the statical aspects of an archive. For the dynamical aspects, we refer to Chapter 4.

3.2.1 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 equivalence 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 distinguish 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.

3.2.2 The Universal Quantifier

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

$$\forall_{\circ}\,\forall_{D}$$

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

$$\text{is–associative}[D, \circ] \Leftrightarrow \forall_{x,y,z\in D}\,((x\circ y)\circ z = x\circ(y\circ z))$$

$$\text{is–commutative}[D, \circ] \Leftrightarrow \forall_{x,y\in D}\,(x\circ y = y\circ x)$$

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:

$$\forall_{x,\bar{x}}\,\forall_{A}$$

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

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

$$\nexists_{\bar{a}}\,(A = \langle \bar{a}\rangle) \Rightarrow (\text{is–list}[A] \Leftrightarrow \text{False})$$

Parametrizing the element type (changing the constant $\mathbb{Z}$ to the variable **Z**), we obtain the type of polymorphic 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 Section 3.5.) The idea of parametrization (“making constants into variables”), usually wrapped into corresponding functors, is applied frequently also in mathematics: 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.

3.2.3 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:

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

$$\text{is–incident}[p, l]$$

$$\text{is–incident}[q, l]$$

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

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

$$\text{is–incident}[p, l]$$

$$\text{is–incident}[p, m]$$

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

Note that in *Theorema* ranges can be described by unary predicates, like **is–point** and **is–line** in the above example; 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]:

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

$$\text{quick–sort}[\langle\rangle] = \langle\rangle$$

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

$$\forall_{\text{is–tuple[X]}}$$

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

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

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

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 precede 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 3.4.1, i.e. they are a means of avoiding name clashes.

3.2.4 The Substitution Quantifier

As for the existential and universal quantifiers, the language of archives provides a multi–line variant of the ***substitution quantifier*** ⇤←, typically verbalized “let” in prefix and “where” in postfix usage. It is easiest to first consider an example (claiming correctness of Cardano's formula):

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

$$\Lsh\!\leftarrow$$

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

$$\leftarrow$$

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

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

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

As can be seen from this example, substitution quantifiers can be nested into each other. They are used for avoiding multiple occurrences 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:

$$\Lambda$$

$$\leftarrow$$

$$x_1 = \tau_1$$

$$\ldots$$

$$x_n = \tau_n$$

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

3.2.5 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], $\min[D] = \underset{x \in D}{\ni} \underset{y \in D}{\forall} (y \neq x \Rightarrow y > x)$]

The intuitive meaning of the description $\underset{x}{\ni} \phi$ 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:

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

$$\min[D] = \underset{x \in D}{\ni} \underset{y \in D}{\forall} (y \neq x \Rightarrow x < y)$$

It translates to the plain *Theorema* Definition above.

3.3 Labelling Blocks of Formulae

As mentioned in Subsection 2.1.1, 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 formulae associated with blocks of formulae via the ⇌ symbol. In the example below, **`BinaryRelations`**, **`BasicProperties`** and **`CompoundProperties`** are labels:

$$\text{BinaryRelations} \rightleftharpoons$$

$$\text{BasicProperties} \rightleftharpoons \underset{R}{\forall} \underset{D}{\forall}$$

$$\text{is–reflexive}[D, R] \Leftrightarrow \underset{x \in D}{\forall} R[x, x]$$

$$\text{is–symmetric}[D, R] \Leftrightarrow \underset{x,y \in D}{\forall} (R[x, y] \Rightarrow R[y, x])$$

$$\ldots$$

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

$$\text{is–partial–order}[D, R] \Leftrightarrow$$

$$\text{is–reflexive}[D, R]$$

$$\text{is–antisymmetric}[D, R]$$

$$\text{is–transitive}[D, R]$$

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 2.2.1. We distinguish 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] ⇔

$\underset{\underset{M}{\in}[x,y]}{\forall} \underset{M}{\in}\left[x \underset{M}{\circ} y\right]$

Semigroups ⇌ $\underset{S}{\forall}$

is–semigroup[S] ⇔

is–magma[S]

$\underset{\underset{S}{\in}[x,y,z]}{\forall} \left(\left(x \underset{S}{\circ} y\right) \underset{S}{\circ} z = x \underset{S}{\circ} \left(y \underset{S}{\circ} z\right)\right)$

…

LatticeTheory ⇌

Posets ⇌

…

Chains ⇌

…

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

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

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

$$\text{Magmas} \Leftrightarrow \forall_{M} \left(\text{is–magma}[M] \Leftrightarrow \forall_{\epsilon_M[x,y]} \epsilon_M\left[x \circ_M y\right] \right)$$

$$\text{Semigroups} \Leftrightarrow \forall_{S} \left(\text{is–semigroup}[S] \Leftrightarrow \left(\text{is–magma}[S] \bigwedge \forall_{\epsilon_S[x,y,z]} \left(\left(x \circ_S y\right) \circ_S z = x \circ_S \left(y \circ_S z\right) \right) \right) \right)$$

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

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

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

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 label*** rather than a hierarchical one.

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:

$$\text{Posets} \rightleftharpoons \forall_{P}$$

▪Basics $\rightleftharpoons$

▪Definitions $\rightleftharpoons$

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

$$\forall_{\epsilon_P[x,y,z]}$$

$$x \preceq_P x$$

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

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

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

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

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

…

▪Theorems ⇌

is–poset[P] ⇒ is–poset[converse[P]]

…

▪Advanced ⇌

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

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

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

▪Theorems ⇌

is–lattice[P] ∧ is–finite[P] ⇒

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

…

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 ⇔

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

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

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

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

Posets▪Advanced▪Definitions ⇔

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

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

Posets▪Advanced▪Theorems ⇌

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

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

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 3.2.2, 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**.

3.4 Resolving Ambiguities of Symbols

3.4.1 Global and Local Symbols

As we saw in Subsection 2.1.3, 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 3.2.3, 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 3.2.3]

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 occurrences 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 ⇌

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

…

Regularity ⇌

$$\forall_x \left(\exists_y y \in x \Rightarrow \exists_y \left(y \in x \bigwedge \nexists_z (z \in x \wedge z \in y) \right) \right)$$

▪Definitions ⇌

Subset ⇌

$$\forall_x (x \subseteq y) \Leftrightarrow \forall_z (z \in x \Rightarrow z \in y)$$

EmptySet ⇌

$$\forall_x (x \notin \emptyset)$$

…

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.

3.4.2 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 cautious 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 3.1.2: Namespaces are function symbols wrapping symbols in order to distinguish their different meanings. For example, the package

BinRel : ⟨is–transitive⟩ ⇌

$$\forall_{\sim} \text{is–transitive}[\sim] \Leftrightarrow \forall_{x,y,z} (x \sim y \wedge y \sim z \Rightarrow x \sim z)$$

…

is translated to

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

so that $\underset{\text{BinRel}}{\text{is–transitive}}$ will be distinguished from other occurrences of the symbol **`is-transitive`**, e.g. the following set theoretic notion:

SetTh : ⟨is–transitive⟩ ⇌

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

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 $\sigma_1, \ldots, \sigma_n$ to the namespace associated with a label L, its so–called ***home namespace,*** in a block of formulae:

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

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 $L[\sigma_1], \ldots, L[\sigma_n]$; we then say that $\sigma_1, \ldots, \sigma_n$ are bound to the namespace **L**. Such packages will be called ***wrapped***, as opposed to the ***plain*** ones of Section 3.3.

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 improving 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 $\mathbb{R}$ 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 $\rightleftharpoons \underset{x}{\forall}$

BinRel : ⟨is–transitive⟩

is–transitive[<]

$\underset{a,b}{\forall}$ (a ⊏ b ⇔ |a − x| < |b − x|) ⇒ is–transitive[⊏]

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

…

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

…

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 Chapter 5.

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 category of semigroups:

$\underset{S}{\forall}$

Semigroup[S] ⇔

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

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

$\in[x * y]$

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

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

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

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

If one uses hierarchical labels (see Section 3.3), 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[$\sim$] $\Leftrightarrow \underset{x,y,z}{\forall} (x \sim y \wedge y \sim z \Rightarrow x \sim z)$

…

The translation to *Theorema* reads as follows:

Algebra $\Leftrightarrow$ (Algebra▪Relations $\wedge$ …)

Algebra▪Relations $\Leftrightarrow$ (Algebra▪Relations▪UnRel $\wedge$ Algebra▪Relations▪BinRel $\wedge$ …)

Algebra▪Relations▪UnRel $\Leftrightarrow$ …

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

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 $\left(\mathrm{L} \Leftrightarrow \underset{\mathrm{L}}{\mathrm{M}}\right) \bigwedge \left(\underset{\mathrm{L}}{\mathrm{M}} \Leftrightarrow \phi\right)$. As explained in Subsection 3.1.2, the notation $\underset{\mathrm{L}}{\mathrm{M}}$ 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 overload 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.

3.5 Categories and Functors via Namespaces

As already mentioned in Subsection 3.1.2, 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 ℕ, + and ∈ can be realized thus (note the difference between ∈ and ϵ):

NaturalSemigroup : ⟨ϵ, ∗⟩ ⇌ $\underset{\mathrm{x,y}}{\forall}$

ϵ[x] ⇒ x ∈ 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 introduction 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 3.1.3 (note the difference between O and **0**):

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

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

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

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

$$\overline{O}\, g = \underset{O}{0}$$

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 3.1.2, 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:

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

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

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

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

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

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

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

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:

$$\text{DP} \Leftrightarrow \underset{G,H}{\forall} \underset{x,y}{\forall} \left(\left(\underset{\underset{DP}{G \times H}}{\epsilon}[x] \Leftrightarrow \left(\text{is–tuple}[x] \bigwedge (\text{card}[x] = 2) \bigwedge_{G} \epsilon[x_1] \bigwedge_{H} \epsilon[x_2] \right) \right) \bigwedge \right.$$
$$\left. \left(\left(x \underset{\underset{DP}{G \times H}}{+} y \right) = \left\langle x_1 \underset{G}{+} y_1,\ x_2 \underset{H}{+} y_2 \right\rangle \right) \bigwedge \left(\underset{\underset{DP}{G \times H}}{0} = \left\langle \underset{G}{0},\ \underset{H}{0} \right\rangle \right) \bigwedge \left(\underset{\underset{DP}{G \times H}}{-} x = \left\langle \underset{G}{-} x_1,\ \underset{H}{-} x_2 \right\rangle \right) \right)$$

Observe that the formula above (except for the label DP) is also what the *Theorema* function **Flat**: **tenKB**, 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 he insists.

Namespace declarations also facilitate the specification of categories, as we have seen in the example of semigroups given in Subsection 3.4.2. 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:

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

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

$$\text{AbelianGroup}[R]$$

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

Distributive[R]

$\forall_{G}$ AbelianGroup[G] $\Leftrightarrow$

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

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

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

$\forall_{G}$ Group[G] $\Leftrightarrow$

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

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

$\forall_{\epsilon[x]} \forall_{\epsilon[y]} \forall_{\epsilon[z]}$

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

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

$\forall_{D}$ Distributive[D] $\Leftrightarrow$

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

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

$\forall_{M}$ Monoid[M] $\Leftrightarrow$

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

$\forall_{\epsilon[x,y,z]}$

$\epsilon[x \circ y]$

$\epsilon[1]$

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

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 3.2). 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):

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

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

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

Since symbols like $\underset{M}{\in}$ are internally represented as **M[∈]** and the like, the rule of β–reduction and case distinction on the lambda expression above finally leads to:

$$\underset{\underset{Q}{\in}[x,y,z]}{\forall}\left(\underset{Q}{\in}\left[x \underset{Q}{*} y\right] \bigwedge \underset{Q}{\in}[1] \bigwedge \left(x \underset{Q}{*} y\right) \underset{Q}{*} z = x \underset{Q}{*} \left(y \underset{Q}{*} z\right)\right)$$

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 Basic Operations on Mathematical Knowledge Archives

In the previous chapter we introduced the notions of label and namespace, which permit a structured representation of a mathematical knowledge base; we have called such a knowledge base an archive. Based on the *Theorema* language, archives offer constructs for splitting formulae in multiple cells, with quantifier ranging over whole cell groups and labels for attaching a name to the groups. This makes them very readable and particularly suited for large bodies of mathematical knowledge. Up to now we have studied archives from a statical perspective, analyzing their syntax, their expressive power and their relation to *Theorema*. But archives are not an aim in themselves—we would like to use them in ***various operations***, like knowledge buildup and retrieval, especially in the context of theorem proving and algorithm synthesis. Starting with basic I/O operations on archives (Section 4.1), we consider next several operations for restructuring archives, for "mixing" them in various ways and for manipulating their parts (Section 4.2). We turn then to the issue of translating them to plain predicate logic (Section 4.3), and to archive operations for theory exploration (Section 4.4). Archive operations for retrieval will be treated in Chapter 5.

4.1 Loading and Saving Archives in Theorema

4.1.1 General I/O Commands for Archives

In order to load an archive saved under the filename **`Algebra.nb`** in the home directory, one uses the command:

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

By this call, the box structure of the mathbook is parsed into a *Mathematica* expression [Wolfram03], subsequently stored in the variable **`archive`**. The underlying expression language is that of *Theorema* [BuchbergerEtAl00], extended by the language features presented in the preceding chapter. Its precise grammar is provided in Subsection 4.1.2.

As a second argument, the user can also specify a keyword referring to a label of the mathbook, extracting the package headed by the label. Consider for example loading a package entitled **`Basic`**`:`**`Properties`**:

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

In the following subsections we give a detailed description of how mathbooks are processed: parsing yields what we call a ***concise archive*** (since its richer language permits natural descriptions avoiding redundancies), which is then expanded to a ***verbose archive*** (so called because it will typically be much longer due to unfolding some constructs in the concise archive).

For viewing and browsing archives (in particular those generated by the operations described in Section 4.2), an inverse operation to **`LoadArchive`** is needed. The command

```
SaveArchive[arch, fname]
```

generates from **`arch`** a *Mathematica* notebook to be stored in the file **`fname`**. This will create a mathbook (typically not identical but equivalent to the original mathbook), which is ***regular*** in the sense explained in Subsection 4.1.2.

There are two other I/O commands of lesser importance, mainly intended for writing and reading intermediate portions of an archive, as a *Mathematica* expression, namely **`WriteArchive`** and **`ReadArchive`**. An archive expression **`arch`** is stored on the harddisk under the filename **`fname`** by the command **`WriteArchive[arch,fname]`**. The archive expression stored in the file **`fname`** is regained by **`ReadArchive[fname]`**. Since in this case one is working with archive expressions, one cannot view or browse them in the same nice manner as mathbooks. But the advantage is that reading/writing is much faster than parsing/formatting as involved in the commands **`LoadArchive`** and **`SaveArchive`**, so they are useful for storing intermediate results.

For example, assume the archive **`dirprod`** is the result of loading the mathbook defining the direct product introduced in Section 3.5:

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

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

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

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

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

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

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

The user can then write it to a scratchfile by saying **`WriteArchive[dirprod,"scratch.m"]`**. If he later reads from this file, he will regain the archive expression **`dirprod`**:

$$\text{™Binding}[\mathrm{DP}, \times] \rightleftharpoons \underset{G,H}{\forall} \text{™Conjunction}\Big[\text{™Binding}[G \times H, \epsilon, +, 0, -],$$
$$\underset{x,y}{\forall} \text{™Conjunction}\Big[(\epsilon[x] \Leftrightarrow \text{IsTuple}[x]) \wedge \text{card}[x] = 2 \wedge G[\epsilon][x_1] \wedge H[\epsilon][x_2],\ x + y = \left\langle x_1 \underset{G}{+} y_1,\ x_2 \underset{H}{+} y_2 \right\rangle,$$
$$0 = \langle G[0], H[0] \rangle,\ -x = \left\langle \underset{G}{-} x_1,\ \underset{H}{-} x_2 \right\rangle\Big]\Big]$$

Note that the difference between **LoadArchive** and **SaveArchive** on the one hand and **ReadArchive** and **WriteArchive** on the other hand is comparable to the difference between the *Mathematica* commands **NotebookRead** and **NotebookWrite** on the one hand and its commands **Get** and **Put** on the other hand.

4.1.2 Parsing a Mathbook

Up to now we spoke about mathbooks in an informal manner, presenting them by examples. We start now to expose their grammar. As briefly mentioned before, a ***mathbook*** is a list of *Mathematica* **RowBox** structures, which we denote in curly braces by $\{r_1, r_2, \ldots\}$ as for *Mathematica* lists. Thus the comma between r_1 and r_2 represents the "newline" separating the corresponding cells, with r_1 being the leader of the cell group containing $r_1, r_2, \ldots$. (The leader is the first cell of a cell group—it remains visible when the cell group is collapsed.)

We apply a variant of EBNF grammar with the following extra conventions: $\|\ldots\|^*$ means repeating zero or more times, $\|\ldots\|^+$ one or more times, and $\|\ldots\|^-$ stands for an option (i.e. zero or one times).

We use the following syntactic categories: **MBK** for mathbooks, **LFM** for labelled formulae, **UFM** for unlabelled formulae, **ETM** for extended terms, **NSP** for namespace declarations, **SUB** for substitutions, **LBL** for labels, **UNI** for universal and **QNT** for generic quantifier prefixes. The terminal symbols **Id**, **Tfm** and **Ttm** stand respectively for the legitimate identifiers in *Mathematica*, the formulae and the terms in plain *Theorema* (i.e. formulae and terms contained in a single cell and without the symbols : and ⇌).

MBK:: = { Title ||, LFM||$^+$ }

LFM:: = {LBL || : ⟨||Id||$^+$⟩||$^-$ ⇌ QNT {, SUB}$^+$ ||, UFM||$^+$}
| LBL || : ⟨||Id||$^+$⟩||$^-$ ⇌ ||LFM,||* LFM
| UNI ||, LFM||$^+$
| UFM

UFM:: = {QNT ||, UFM||$^+$}
| { Tfm (⇔| ⇒) ||, SUB||$^-$ ||, UFM||$^+$}
| NSP
| Ttm =, ETM
| {Tfm, SUB}
| Tfm

ETM:: = Ttm
| {|| ∋ / Id ||$^+$ QNT ||, UFM||$^+$}

NSP:: = (LBL | Ttm) : ⟨||Id||$^+$⟩

SUB:: = { ⇤← ||, Tfm||$^{+}$}

LBL:: = ||▪||$^{-}$ Id
| ||Id ▪||$^{+}$ Id

UNI:: = $\left\| \underset{\|\|\mathrm{Id}\|\|^{+}}{\forall} \right\|^{*}_{\mathrm{Tfm}}$

QNT:: = $\|\forall|\ \exists\ \|^{*}_{\|\|\mathrm{Id}\|\|^{+}}$ Tfm

Let us also rephrase the rules of the ***grammar*** in a more intuitive form: A mathbook is a list containing the title of the notebook and one or more labelled formulae. A labelled formula is essentially a package containing one or more ⇌. The head of the package is a label, optionally followed by the list of symbols to be bound in the namespace associated to the given label. The body of the package can either be a list of other labelled formulae or of unlabelled formulae with substitutions, optionally preceded by a quantifier prefix. A labelled formula can also be a universal prefix whose scope encompasses other labelled formulae in the subsequent cells. A degenerate case of a labelled formula (with no ⇌) is an unlabelled formula, which is nothing else than a multi–line *Theorema* formula.

An extended term is either a *Theorema* term or a multi–line *Theorema* description (containing on the first line the description quantifier, optionally followed by a quantifier prefix, and unlabelled formulae on the subsequent lines). A namespace contains a label (or a *Theorema* term) followed by the symbols residing in it. A substitution is a list of *Theorema* formulae headed by a cell containing the symbol ⇤←. A label is either a legitimate identifier or a sequence of identifiers connected by ▪. Finally, a universal/generic quantifier prefix contains universal/generic quantifiers, binding variables satisfying conditions given by *Theorema* formulae.

Note that in practice (and also in several examples), we allowed an extra facility for the user. As already mentioned in Section 3.3, universal quantifiers having ⇌ in their scope are also permitted; it is understood that they are to be distributed to all multi–line *Theorema* formulae in the blocks underneath. Thus the second rule for LFM could be seen as:

LBL || : ⟨||Id||$^{+}$⟩||$^{-}$ ⇌ UNI ||LFM,||* LFM

Note that this is just a notational facility, so we kept it out of the grammar.

In executing a **`LoadArchive`** command, the first process invoked is the parser. It works along the grammar presented above: For each of its rules, it applies a *Mathematica* command for creating the corresponding archive expression. For example, the mathbook

TupleTheory▪BasicProperties : ⟨ ≈ , dfo, ≍⟩ ⇌

PermutedVersion ⇌

Definition ⇌ …

Propositions ⇌

$$\forall_{\text{is–tuple[X]}} X \approx X$$

$$\forall_{\text{is–tuple[X,Y]}} (X \approx Y \Rightarrow Y \approx X)$$

$$\forall_{\text{is–tuple[X,Y,Z]}} (X \approx Y \wedge Y \approx Z \Rightarrow X \approx Z)$$

$$\forall_{\text{is–trivial–tuple[X]}} \forall_{\text{is–tuple[Y]}} (X \approx Y \Rightarrow (X = Y))$$

$$\forall_{\text{is–tuple[A,B,Y,Z]}} ((A \approx Y \wedge B \approx Z) \Rightarrow (A \asymp B) \approx (Y \asymp Z))$$

…

is parsed into the following internal representation:

™Binding[TupleTheory▪BasicProperties, ≈ , dfo, ≍] ⇌

(PermutedVersion ⇌ ™Conjunction[Definition ⇌ …, Propositions ⇌

™Conjunction[$\forall_{\text{IsTuple}[X]} (X \approx X)$, $\forall_{\text{IsTuple}[X,Y]} (X \approx Y \Rightarrow Y \approx X)$, $\forall_{\text{IsTuple}[X,Y,Z]} (X \approx Y \wedge Y \approx Z \Rightarrow X \approx Z)$,

$\forall_{\text{is–trivial–tuple}[X]} \forall_{\text{IsTuple}[Y]} (X \approx Y \Rightarrow (X = Y))$, $\forall_{\text{IsTuple}[A,B,Y,Z]} (A \approx Y \wedge B \approx Z \Rightarrow (A \asymp B) \approx (Y \asymp Z))$, …]])

Here the namespace is represented by **™Binding**, the multi–line conjunction by **™Conjunction** and the predicate **is–tuple** by its *Theorema* counterpart **IsTuple**.

The output of the parser is an ***archive***, meaning a formula in the following extended *Theorema* language, which is again specified by a suitable grammar. This time we use the following syntactic categories: **ARX** for (sub)archives, **FRM** for unlabelled formulae, **AVP** for argument/value pairs, **LBL** for labels, and **UNI** for universal quantifier prefixes. The terminal symbols are as for the previous grammar, plus the constructors **™Conjunction**, **™Binding**, **™FiniteFunction**, and **™ArgVal**.

ARX:: = LBL ⇌ UNI ARX | LBL ⇌ UNI ™Conjunction[||ARX||$^{+}$] | FRM

FRM:: = Tfm | ™Binding[Ttm, ||Id||$^{+}$] | ™FiniteFunction[||AVP||$^{+}$]

AVP:: = ™ArgVal[Id, Ttm]

LBL:: = ||▪||$^{-}$ ||Id▪||* Id | ™Binding[||▪||$^{-}$ ||Id▪||* Id , ||Id||$^{+}$]

UNI:: = $\left\| (\forall)_{\|\text{Id}\|^{+}} \right\|^{*}_{\text{Tfm}}$

Let us again formulate the ***grammar*** in a more intuitive fashion: An archive is a formula in an extended *Theorema* language (by adding the ⇌ and the above mentioned constructors). Its outermost symbol is usually the ⇌, that has as a first argument a label, and as second argument either a conjunction of subarchives or a universal quantifier prefix having a subarchive in its scope. The degenerate case of an archive is a formula: This is either a plain *Theorema* formula, a namespace declaration (denoted by a *Theorema* term followed by the list of symbols to be bound in it), or a theory interpretation in the sense of Section 3.5. Theory interpretations are realized by finite functions consisting of argument/value pairs. A label is either an "extended" identifier (i.e. an atomic identifier or a composite one having ▪ between its segments and/or in front of it) or a home namespace in the sense of Subsection 3.4.2. Of course, a universal quantifier prefix consists of normal *Theorema* universal quantifiers. Note that here—in contrast to the grammar of mathbooks—we do not deal with the cell structure.

We mentioned in the previous section that **`SaveArchive`** is an inverse operation to **`LoadArchive`**. More precisely, it produces a *regular* mathbook in the following sense:

1. It does not contain any non–formal contents (e.g. comments).

2. Cells are broken after every occurrence of ⇌. For example the archive

```
Relations ⇌ ™Conjunction[UnaryRelations ⇌ …,
    BinaryRelations ⇌ ™Conjunction[BasicProperties ⇌ …, Operations ⇌ …, …]]
```

will yield the following mathbook:

Relations ⇌

UnaryRelations ⇌

…

BinaryRelations ⇌

BasicProperties ⇌

…

Operations ⇌

…

…

3. Cells containing groups of quantifiers will be broken before and after the quantifier group, so the archive

$$\text{Modules} \rightleftharpoons \text{™Conjunction}\Big[\ldots,\ \text{LinearTransformations} \rightleftharpoons \text{™Conjunction}\Big[\underset{\text{is–ring}[\boldsymbol{R}]}{\forall}\ \underset{\text{is–module}[\boldsymbol{R}][\boldsymbol{A}]}{\forall}\ \underset{\text{is–module}[\boldsymbol{R}][\boldsymbol{B}]}{\forall}\ \text{is–abelian–group}[\text{mod–hom}[\boldsymbol{R}][\boldsymbol{A}, \boldsymbol{B}]], \ldots\Big]\Big]$$

leads to the following mathbook:

Modules ⇌

…

LinearTransformations ⇌

$$\underset{\text{is–ring}[R]}{\forall}\ \underset{\text{is–module}[R][A]}{\forall}\ \underset{\text{is–module}[R][B]}{\forall}$$

is–abelian–group[mod–hom[R][A, B]]

…

4. Cells will be split after ⇔ and ⇒. Thus the archive

$$\text{is–chain}[C] \Rightarrow \underset{\in C}{\forall}[m]\ \text{™Conjunction}[\text{is–minimal}[C, m] \Leftrightarrow \text{is–least}[C, m], \text{is–maximal}[C, m] \Leftrightarrow \text{is–greatest}[C, m]]$$

will be formatted as the mathbook:

is–chain[C] ⇒

$$\underset{\in C}{\forall}[m]$$

is–minimal[C, m] ⇔

is–least[C, m]

is–maximal[C, m] ⇔

is–greatest[C, m]

5. Similarly, formulae containing = are split after the equal sign and descriptions after the э. Consider the following archive fragment:

$$\min[D, <] = \underset{x\in D}{э}\ \underset{\substack{y\in D\\ y\neq x}}{\forall}\ (x < y)$$

It will yield the mathbook:

$$\min[D] = \exists_{x \in D} \; \forall_{\substack{y \in D \\ y \neq x}} \; x < y$$

It is clear that applying the command **LoadArchive** on the mathbook resulting from **SaveArchive[arch]** we regain the original archive **arch**. But if one uses the command **SaveArchive** on the archive **LoadArchive[mbk]**, the resulting regular mathbook is typically not identical (but of course equivalent) to the input mathbook **mbk**. One can thus view the composite operation **SaveArchive∘LoadArchive** as a canonical simplifier on mathbooks; its canonical representatives are exactly the regular mathbooks.

(Note that the explanations given above for **LoadArchive** are somewhat simplified in order to convey the basic idea. As we shall see in the next subsection, archives are subsequently expanded in various ways, and the default behavior of the **LoadArchive** operation is to incorporate this expansion. If one explicitly wants to execute only the parser, bypassing the expansion process, one has to specify the option **Expansion⟶False**.)

4.1.3 Expanding an Archive

As we have seen in the previous subsection, the first step in loading the archive is parsing the mathbook. The result is a ***concise*** archive: an archive employing several shortcuts as announced in Chapter 3. These shortcuts are only there as notational facilities for the user; they do not extend the expressive power of the language of archives. Expanding these shortcuts by using the command **ExpandArchive** yields a so–called ***verbose*** archive, which uses only the primitive features of the language of archives.

In Section 3.3, we mentioned that quantifiers appearing in package heads higher up in the hierarchy (i.e. containing $\rightleftharpoons$ symbols in their scope) are to be distributed to all formulae in the blocks underneath. For example, having the package

$$\text{GroupTheory} \rightleftharpoons \forall_{G}$$

$$\text{Definitions} \rightleftharpoons \forall_{N} \left(\text{is–normal–subgroup}[N, G] \Leftrightarrow \left(\text{is–subgroup}[N, G] \bigwedge \forall_{\substack{\in[x] \\ G}} \forall_{\substack{\in[y] \\ N}} \in_{N} \left[x *_{G} y *_{G} i_{G}[x] \right] \right) \right)$$

$$\text{Theorems} \rightleftharpoons \forall_{f} \left(\text{is–endomorphism}[f, G] \Rightarrow \text{is–normal–subgroup}[\ker[f], G] \right)$$

amounts to

GroupTheory ⇌

$$\text{Definitions} \rightleftharpoons \underset{G}{\forall}\,\underset{N}{\forall}\left(\text{is–normal–subgroup}[N, G] \Leftrightarrow \left(\text{is–subgroup}[N, G] \bigwedge \underset{\in[x]}{\underset{G}{\forall}}\,\underset{\in[y]}{\underset{N}{\forall}}\, \in_N\left[x *_G y *_G i_G[x]\right]\right)\right)$$

$$\text{Theorems} \rightleftharpoons \underset{G}{\forall}\,\underset{f}{\forall}\,(\text{is–endomorphism}[f, G] \Rightarrow \text{is–normal–subgroup}[\ker[f], G])$$

This ***distribution of quantifiers*** in an archive **`arch`** can be done by the command **`Split`**`:` **`Quantifiers[arch]`** or as a step in the full expansion of the archive, executed by the command **`ExpandArchive`** described below.

The ***representation of labels*** is another issue dealt with at this stage of expanding an archive. As mentioned in Subsection 3.4.2, hierarchical labels can be interpreted as shortcuts for labels residing in a namespace. For example, the package

BinaryRelations ⇌

- BasicProperties ⇌ …
- CompoundProperties ⇌ …
- Operations ⇌ …

$$\text{strict}[R] = \underset{S}{\exists}\,\underset{x,y}{\forall}\,(S[x, y] \Leftrightarrow (R[x, y] \wedge x \neq y))$$

$$\text{converse}[R] = \underset{S}{\exists}\,\underset{x,y}{\forall}\,(S[x, y] \Leftrightarrow R[y, x])$$

$$\text{strict–converse}[R] = \underset{S}{\exists}\,\underset{x,y}{\forall}\,(S[x, y] \Leftrightarrow (R[y, x] \wedge x \neq y))$$

$$\text{neg}[R] = \underset{S}{\exists}\,\underset{x,y}{\forall}\,(S[x, y] \Leftrightarrow \neg R[x, y])$$

$$\text{neg–strict}[R] = \underset{S}{\exists}\,\underset{x,y}{\forall}\,(S[x, y] \Leftrightarrow (\neg R[x, y] \wedge x \neq y))$$

$$\text{neg–converse}[R] = \underset{S}{\exists}\,\underset{x,y}{\forall}\,(S[x, y] \Leftrightarrow \neg R[y, x])$$

$$\text{neg–strict–converse}[R] = \underset{S}{\exists}\,\underset{x,y}{\forall}\,(S[x, y] \Leftrightarrow (\neg R[y, x] \wedge x \neq y))$$

is thus a shortcut for

BinaryRelations : ⟨BasicProperties, CompoundProperties, Operations⟩ ⇌

BasicProperties ⇌ …

CompoundProperties ⇌ …

Operations ⇌ …

Furthermore, an absolute label of the form

```
TupleTheory▪ExtendedProperties▪SortingAlgorithms▪MergeSort▪Definition
```

will be then interpreted as:

```
TupleTheory[ExtendedProperties][SortingAlgorithms][MergeSort][Definition]
```

Another possibility of ***normalizing labels*** is investigated in [RosenkranzEtAl09]. There a hierarchical label is viewed as having the type **`String×String×…→Bool`**. Labels are then represented by considering **▪** as a symbol with flexible arity. For example **`TupleTheory▪BasicProperties▪Concatenation`** will be represented internally as **`▪["TupleTheory","BasicProperties","Concatenation"]`**. We provide this normalization as an optional postprocessing step described in Section 4.3.

In a nutshell, the command **`ExpandArchive[arch]`** creates a verbose archive, performing the steps mentioned above: distribution of quantifiers and normalization of labels. If the user wishes to perform the individual steps separately, the commands **`SplitQuantifiers`** (mentioned above) and **`NormalizeLabels`** (that assumes the quantifiers of the input archive are already distributed) are at his disposal. The latter achieves both elimination of relative labels and representation of all labels. Eliminating these shortcuts leads to a more verbose form, which is used internally for efficiently executing the various operations presented in the next sections. In the subsequent sections, the term “archive” should therefore be understood in the sense of “verbose archives”.

4.2 Merging, Splitting and Inserting Archives

Two operations are available for “mixing” archives **`arch1`** and **`arch2`**, differing in how they treat global symbols: While the command **`MergeArchive[arch1,arch2]`** identifies them (meaning the symbols remain global so that they refer to the same concepts), its analog **`JoinArchive[arch1,arch2]`** wraps them into separating namespaces whose names are the top–level labels of the archives (thus creating the corresponding home namespaces). Both operations generate the conjunction of the original archives.

As an example, consider ***merging*** an archive on group theory with an archive on lattice theory as a preparation for asserting the theorem that the subgroups of a given group form a Moore system (and thus a lattice with the appropriate operations as defined in the functor below). Note that in the example below the structures are formalized with carrier sets instead of carrier predicates; by convention

we denote the former by # and the latter by ϵ. The first input archive is named **GrpThr**, and it uses global set–theoretic notions like set membership and binary intersection:

GroupTheory $\rightleftharpoons \underset{G}{\forall}$

Subgroups $\rightleftharpoons$

Definitions $\rightleftharpoons$

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

$\text{SubGrp}[G] = \left\{ S \;\middle|\; \underset{S \subseteq \#}{\underset{x,y \in S}{\forall}} x * y^{-1} \in S \right\}$

…

Theorems $\rightleftharpoons$

$G : \langle \# \rangle$

$\underset{\mathcal{G} \subseteq \text{SubGrp}[G]}{\forall} \bigcap \mathcal{G} \in \text{SubGrp}[G]$

$\# \in \text{SubGrp}[G]$

…

The second input archive is named **LatThr** and also contains global set–theoretic notions like set inclusion, powerset, unary and binary intersection:

LatticeTheory $\rightleftharpoons \underset{\mathcal{L}}{\forall} \underset{B}{\forall}$

Definitions $\rightleftharpoons$

…

$\text{is–Moore–system}[\mathcal{L}, B] \Leftrightarrow$

$\mathcal{L} \subseteq \mathcal{P}[B]$

$\underset{\mathcal{M} \neq \phi}{\forall} \left(\mathcal{M} \subseteq \mathcal{L} \Rightarrow \bigcap \mathcal{M} \in \mathcal{L} \right)$

$B \in \mathcal{L}$

…

Moore–lattice[$\mathcal{L}$] : $\langle \#, \bigsqcap, \bigsqcup \rangle$

$\underset{M,N}{\forall}$

$\# = \mathcal{L}$

$M \bigsqcap N = M \bigcap N$

$M \bigsqcup N = \bigcap \left\{ L \mid_{L \in \mathcal{L}} M \subseteq L \wedge N \subseteq L \right\}$

Theorems $\rightleftharpoons$

is–Moore–system[$\mathcal{L}$, B] $\Rightarrow$ is–lattice[Moore–lattice[$\mathcal{L}$]]

…

To the archive generated by **`MergeArchive[GrpThr,LatThr]`** the user can now add the theorem mentioned before:

$\underset{\text{is–group[G]}}{\forall}$ is–Moore–system$\left[\text{Subgroups[G]}, \underset{G}{\#}\right]$

Note that in the merged archive set–theoretic operation symbols like binary intersection must be identified in order to ensure the correctness of the statements claimed.

A typical example where ***joining*** would be the method of choice is when one wants to bring together archives like **`Modules`** and **`Algebras`**, where e.g. two instances of the predicate symbol **`generates`** must be kept apart. In the former case, a set **`S`** is called a generating set of a module **`M`** over a ring **`R`** if **`S`** is a subset of the module and all elements of **`M`** can be written as linear combinations of elements in **`S`**; this is the content of the predicate **`generates`** in **`Modules`**:

Modules $\rightleftharpoons \underset{R}{\forall} \underset{M}{\forall}$

SetTheory : $\langle \in, \subseteq, \omega \rangle$

M : $\langle \#, +, 0, \cdot \rangle$

$\underset{S}{\forall}$

generates[S, M] $\Leftrightarrow$

$S \subseteq \#$

$$\underset{x\in\#}{\forall}\ \underset{n\in\omega}{\exists}\ \underset{\lambda:\, n\to R}{\exists}\ \underset{s:\, n\to S}{\exists}\left(x = \Omega_0^{+}\ \lambda[i]\cdot s[i]\right)$$

Note that here linear combinations are expressed by the generic *Theorema* quantifier Ω. A term $\Omega^{\texttt{op}}_{\substack{\texttt{start}\\ \texttt{rng}}}\ \texttt{expr}$ has the following meaning: If the range **rng** is empty, it is equal to **start**. Otherwise, an arbitrary element is taken out from the range and the binary operation **op** is applied to this element and recursively to the Ω term with reduced range. In the example above, Ω simulates the extended summation quantifier ∑, with the operation + of the module **M** as its underlying addition. Note that · denotes the scalar multiplication in **M**.

In the archive describing algebras, the predicate **generates** asserts that a set **S** of so–called generators is sufficient to express every element of a **K**–algebra **A** as a linear combination of products of such generators:

Algebras ⇌ $\underset{K}{\forall}\ \underset{A}{\forall}$

SetTheory : ⟨ ∈ , ⊆ , ω⟩

A : ⟨#, +, 0, ∗, 1, ·⟩

$\underset{S}{\forall}$

generates[S, A] ⇔

$S \subseteq \#$

$$\underset{x\in\#}{\forall}\ \underset{n\in\omega}{\exists}\ \underset{\lambda:\, n\to R}{\exists}\ \underset{s:\, n\to S}{\exists}\ \underset{m:\, \omega\to\omega}{\exists}\ \underset{k:\, \omega\times\omega\to n}{\exists}\left(x = \Omega_0^{+}\ \lambda[i]\cdot\left(\Omega_1^{*}\ s[k[i,\ j]]\right)\right)$$

Note that here the generic quantifier Ω is used twice, first simulating the summation quantifier ∑ and then simulating a product quantifier Π. Hence the real polynomials $\mathbb{R}[x]$ are generated by $\{x^n \mid n \in \mathbb{N}\}$ as an $\mathbb{R}$–module and by $\{x\}$ as an $\mathbb{R}$–algebra.

In the above example about generating sets of modules and algebras, certain set–theoretic symbols have been employed, namely membership ∈, inclusion ⊆ and the set of natural numbers ω. If one prefers to treat these symbols as globals (such that one need not open the corresponding namespace), it is possible to use the command defined next.

Intuitively, merging is similar to set-theoretic union, whereas joining acts like a disjoint union. Both operations are subsumed by the following command:

```
CombineArchive[arch1, arch2, {sym, nmsp1, nmsp2}, …]
```

Again the formulae of the archives are conjoined, with all global symbols remaining global except for those explicitly specified after **arch1** and **arch2**: The symbol **sym** is respectively wrapped into the namespaces **nmsp1** and **nmsp2** in the subarchives corresponding to **arch1** and **arch2**.

A situation where this occurs is the following example. Assume we want to ***combine*** an archive treating the theory of topological fields and an archive describing order–theory. In both of these archives we have a theorem asserting that $\mathbb{R}$ is complete (in the senses defined in the given theories), both as an ordered field and as a topological field.

The first archive is named **`TopFld`**; it uses the global notion of **`is-field`** and defines the metric notion of completeness:

MetricSpaces $\rightleftharpoons \underset{M}{\forall}$

Definitions $\rightleftharpoons$

…

M : ⟨#, is–Cauchy–sequence, is–convergent⟩

is–complete[M] $\Leftrightarrow \underset{\sigma : \mathbb{N} \to \#}{\forall}$

is–Cauchy–sequence[σ] $\Rightarrow$ is–convergent[σ]

…

Theorems $\rightleftharpoons$

…

is–metric–space[$\mathbb{R}$] $\wedge$ is–complete[$\mathbb{R}$]

…

TopologicalFields $\rightleftharpoons \underset{K}{\forall}$

Definitions $\rightleftharpoons$

K : ⟨#, +, −, 0, $*$, /⟩

is–topological–field[K] $\Leftrightarrow$

is–field[K] $\wedge$ is–topological–space[K]

is–continuous–bin–op[+, #] $\wedge$ is–continuous–bin–op[−, #]

is–continuous–bin–op[$*$, #] $\wedge$ is–continous–bin–op[/, # \ {0}]

…

Theorems ⇌

…

is–topological–field[ℝ]

…

The latter archive is named **`OrdFld`**; it also uses the notion of field and defines completeness as an order–theoretic notion:

OrderedFields ⇌ $\underset{K}{\forall}$

Definitions ⇌

K : ⟨#, ≥ , +, 0⟩

is–ordered–field[K] ⇔

is–field[K] ∧ is–chain[K]

$\underset{a,b,c\in\#}{\forall}$

$a \geq b \Rightarrow a + c \geq b + c$

$a \geq 0 \wedge b \geq 0 \Rightarrow a * b \geq 0$

…

Theorems ⇌

K : ⟨#, is–bounded–above, exists–supremum⟩

is–ordered–field[K] ⇒

is–complete[K] ⇔ $\underset{S\subseteq\#}{\forall}$

is–bounded–above[S] ⇒ exists–supremum[S]

…

is–ordered–field[ℝ] ∧ is–complete[ℝ]

…

When we combine these two archives, we should share symbols like **is–field** but distinguish symbols like the metric notion **is–complete** from its order–theoretic namesake. Calling

```
CombineArchive[TopFld, OrdFld, {is–complete, MetricSpaces, OrderedFields}]
```

will produce the conjunction of **TopFld** and **OrdFld**, wrapping the symbol **is–complete** in the appropriate namespaces, corresponding to its origin (i.e. **is–complete** for the metric notion and **is–complete** for the order–theoretic one).
MetricSpaces **OrderedFields**

Besides the different flavors of combining archives, the user can refer to and manipulate parts of archives, the so–called subarchives. ***Referring to subarchives*** is done by their labels. In the following example, the label **TupleProperties▪Concatenation▪Definition** refers to the formula defining concatenation:

TupleProperties : ⟨is–tuple, is–empty–tuple⟩ ⇌

…

Concatenation : ⟨ ≍ ⟩ ⇌

Definition ⇌

$$\forall_{\overline{x},\overline{y}} \left(\langle \overline{x} \rangle \asymp \langle \overline{y} \rangle = \langle \overline{x}, \overline{y} \rangle\right)$$

Properties ⇌

$$\forall_{\text{is–tuple}[X]} \left(X \asymp \langle\rangle = X\right)$$

$$\forall_{\text{is–tuple}[X]} \left(\langle\rangle \asymp X = X\right)$$

$$\forall_{\text{is–tuple}[X]} \left((X \asymp X = X) \Rightarrow \text{is–empty–tuple}[X]\right)$$

$$\forall_{\text{is–tuple}[X,Y,Z]} \left((X \asymp Y) \asymp Z = X \asymp (Y \asymp Z)\right)$$

…

Moreover, one can also access the fine structure within packages by using the customary position markers (as also provided in *Mathematica*). For example, the position

```
TupleProperties▪Concatenation▪Properties□3□3□2
```

refers to the subformula **is–empty–tuple[X]** in the formula labelled:

```
TupleProperties▪Concatenation▪Properties
```

This subformula is obtained by traversing the formula tree in the following way: the third conjunct of the formula is taken; out of this conjunct, the third part is selected, which is the matrix of the universal quantifier (since in *Theorema* a quantifier takes three arguments: range, condition and matrix), and from this matrix the implicatum is extracted.

The command **`ArchivePart[arch,pos]`** takes from an archive **`arch`** the subformula at position **`pos`**, its counterpart **`ArchiveContext[arch,lbl]`** yields the surrounding context, where **`lbl`** is a label position. For example, consider the above archive **`arch`** on tuple theory. Executing the command

```
ArchivePart[arch, TupleProperties▪Concatenation▪Definition]
```

will yield the subarchive:

Definition ⇌

$$\underset{x,y}{\forall}\ (\langle \overline{x} \rangle \asymp \langle \overline{y} \rangle = \langle \overline{x}, \overline{y} \rangle)$$

By applying its counterpart

```
ArchiveContext[arch, TupleProperties▪Concatenation▪Definition]
```

we obtain:

TupleProperties : ⟨is–tuple, is–empty–tuple⟩ ⇌

…

Concatenation : ⟨ ≍ ⟩ ⇌

Properties ⇌

$$\underset{\text{is–tuple}[X]}{\forall}\ (X \asymp \langle\rangle = X)$$

$$\underset{\text{is–tuple}[X]}{\forall}\ (\langle\rangle \asymp X = X)$$

$$\underset{\text{is–tuple}[X]}{\forall}\ ((X \asymp X = X) \Rightarrow \text{is–empty–tuple}[X])$$

$$\underset{\text{is–tuple}[X,Y,Z]}{\forall}\ ((X \asymp Y) \asymp Z = X \asymp (Y \asymp Z))$$

…

Similarly, for inserting **`arch2`** into **`arch1`** at the label position **`lab`**, one uses the command **`InsertArchive[arch1,arch2,lab]`**. For example, consider the following archive:

Tuples▪SortingAlgorithms ⇌

…

•Auxiliaries ⇌

Merge ⇌ …

BasicSplits ⇌

LeftAndRightSplits ⇌ …

…

▪Merge–Sort ⇌

merge–sort[⟨⟩] = ⟨⟩

$\forall_{x}$ (merge–sort[⟨x⟩] = ⟨x⟩)

$\forall_{\text{is–tuple[X]}}$ (¬ is–trivial–tuple[X] ⇒
(merge–sort[X] = merge[merge–sort[left–split[X]], merge–sort[right–split[X]]]))

We want to insert at the position **`Tuples▪SortingAlgorithms▪Auxiliaries▪BasicSplits`** the following archive:

OddAndEvenSplits ⇌

Definitions ⇌

OddSplit ⇌

odd–split[⟨⟩] = ⟨⟩

$\forall_{x}$ (odd–split[⟨x⟩] = ⟨x⟩)

$\forall_{x,y,\overline{z}}$ odd–split[x, y, $\overline{z}$] = x ⌣ odd–split[⟨$\overline{z}$⟩]

EvenSplit ⇌

even–split[⟨⟩] = ⟨⟩

$\forall_{x}$ (even–split[⟨x⟩] = ⟨⟩)

$\forall_{x,y,\overline{z}}$ (even–split[x, y, $\overline{z}$] = y ⌣ even–split[⟨$\overline{z}$⟩])

Properties ⇌

$\forall_{\text{is–tuple[X]}}$ X ≻ odd–split[X]

$\forall_{\text{is–tuple[X]}}$ X ≻ even–split[X]

$\forall_{\text{is–tuple[X]}}$ (odd–split[X] ≍ even–split[X]) ≈ X

The result of applying the command **`InsertArchive`** will be:

Tuples▪SortingAlgorithms ⇌

…

•Auxiliaries ⇌

Merge ⇌ …

BasicSplits ⇌

LeftAndRightSplits ⇌ …

OddAndEvenSplits ⇌ …

…

▪Merge–Sort ⇌ …

Note that for **`ArchivePart`** we allow position markers, but in **`ArchiveContext`** and **`InsertArchive`** we restricted the positions to absolute labels.

Up to now we have dealt with simple ***structural operations*** on archives, combining them in various ways and manipulating their parts. As we will see in Section 4.4 and in Chapter 5, the current implementation provides also more sophisticated operations like theory exploration and knowledge retrieval. For using existing *Theorema* MKM Tools (described in Section 2.4) and for fully clarifying the semantics of archives, we have built a translator to plain predicate logic, that will be presented next.

4.3 Translating to Plain Predicate Logic

As we explained already in Sections 3.3 and 3.4, the translation of an archive to plain *Theorema* involves a partial loss of structure (this and the property of being idempotent is the motivation for calling this operation a projection). 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", e.g. one cannot say in general whether a $\Leftrightarrow$ stems from a $\rightleftharpoons$. The ***translation process*** has two phases:

- The elimination of home and foreign namespaces in an archive **`arch`** is realized by the command **`EliminateNamespaces[arch]`**.
- The projection of the archive to plain *Theorema*, flattens the $\rightleftharpoons$ as explained in Section 2.3 and 2.4, is implemented by **`ArchiveToTmaTheory[arch]`** and **`ArchiveToAsml[arch]`**.

The command **`ArchiveToTma[arch]`** combines the elimination of namespaces and the flattening. The user can choose the flattening of her choice by setting the option **`Type⟶Theory`** or **`Type⟶Asml`**. As mentioned in Subsection 4.1.3, by setting the option **`LabelAsString⟶True`**, an extra postprocessing step is taken: Labels will be represented as strings, so a label L1▪L2▪…▪Ln will be represented as •["L1","L2",…"Ln"]. The underlying language will then have an extra logical symbol ▪ of flexible arity. In this approach, namespaces occur as function constants in the corresponding internal representation. This approach is investigated further in [RosenkranzEtAl09].

In its turn, the ***elimination of namespaces*** occurs in two subphases:

- Home namespaces are considered as a shortcut for foreign namespaces, as described in Subsection 3.4.2.
- Foreign namespaces can be eliminated by replacing each occurrence of a symbol $\circ$ bound to a namespace **`N`** by **`N[∘]`**.

The elimination of a ***home namespace*** is done by moving it to the right–hand of the $\rightleftharpoons$, thus making it a foreign namespace. If quantifiers occur immediately after the $\rightleftharpoons$, they are shifted one level below, as in the following example. Otherwise, the namespace is prepended to the list of (labelled/unlabelled) formulae following the $\rightleftharpoons$. Let us now turn to an example defining posets:

$$\text{Posets} : \langle \text{is–poset} \rangle \rightleftharpoons \underset{P}{\forall}$$

$$P : \langle \square,\ \preceq \rangle$$

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

$$\forall_{x,y,z\in\square}$$

$$x \le x$$

$$x \le y \wedge y \le x \Rightarrow (x = y)$$

$$x \le y \wedge y \le z \Rightarrow x \le z$$

Replacing the home namespace **`Posets:⟨is–poset⟩`** by its corresponding foreign namespace, we obtain:

Posets $\rightleftharpoons$

Posets : ⟨is–poset⟩

$$\forall_{P}$$

P : ⟨□, ≤⟩

is–poset[P] $\Leftrightarrow$

$$\forall_{x,y,z\in\square}\ s$$

$$x \le x$$

$$x \le y \wedge y \le x \Rightarrow (x = y)$$

$$x \le y \wedge y \le z \Rightarrow x \le z$$

Eliminating also the foreign namespaces, the package becomes:

Posets $\rightleftharpoons$

$$\forall_{P}$$

$$\text{is–poset}_{\text{Posets}}[P] \Leftrightarrow$$

$$\forall_{x,y,z\in\square_{P}}\ s$$

$$x \underset{P}{\le} x$$

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

$$x \underset{P}{\leq} y \bigwedge y \underset{P}{\leq} z \Rightarrow x \underset{P}{\leq} z$$

The resulting archive does not contain namespaces anymore and all labels are absolute. In order to obtain plain *Theorema* formulae, we still have to eliminate the symbols ⇌ and **™Conjunction**, thus making the actual projection.

We have implemented two functions for ***projecting archives***: Besides the translation to a flat list of formulae (called an "assumption list" in *Theorema*, and represented internally by the **•asml** construct, as explained in Subsection 2.4.4) considered up to now, there is also a possibility of retaining the hierarchical structure encoded in the labels by rephrasing them as nested **Theory** environments of Theorema. (As explained in Section 2.4, the **Theory** hierarchies are logic–external but otherwise essentially equivalent to nested ⇌ symbols.) The flat translator is called by **ArchiveToAsml[arch]**, the other one by **ArchiveToTmaTheory[arch]**.

Here is how the package from the beginning of Section 3.3 looks like as a **Theory** in Theorema:

```
Theory["Algebra",
  Theory["GroupTheory"]
  Theory["LatticeTheory"]]
```

```
Theory["GroupTheory",
   Theory["Magmas"]
  Theory["Semigroups"]]
```

$$\text{Theory}\left["\text{Magmas}", \underset{M}{\forall}\left(\text{is–magma[M]} \Leftrightarrow \underset{\in[x,y]_M}{\forall} \in_M [x \circ_M y]\right)\right]$$

$$\text{Theory}\left["\text{Semigroups}", \underset{S}{\forall}\left(\text{is–sgroup[S]} \Leftrightarrow \text{is–magma[S]} \bigwedge_{\in[x,y,z]_S} \underset{S}{\forall} \left((x \circ_S y) \circ_S z = x \circ_S (y \circ_S z)\right)\right)\right]$$

```
Theory["LatticeTheory",
  Theory["Posets"]
  Theory["Chains"]]
```

…

Let us emphasize again, however, that these translators are mainly provided for illuminating the semantics of logic-internal labels and namespaces, and for connecting with other components of the *Theorema* framework.

4.4 Theory Exploration in Archives

Theory exploration is a specific way of ***automated knowledge buildup***, as opposed to the usual manual creation of archives by formalizing existing textbooks, articles, etc. In practice, both methods should be combined for optimal results.

As already explained in Section 2.2, theory exploration can be realized by two main approaches:

- A ***bottom–up exploration*** investigates for newly introduced predicates or functions all interesting interactions to the already existing concepts. Thus a mathematical theory is built up in layers, starting from elementary notions and axioms.
- In a ***top–down exploration***, the existing rudimentary knowledge base (containing only axioms and perhaps some important theorems) is enlarged by using the algorithm synthesis method described in [BuchbergerCraciun03] and implemented in [Craciun08]: Trying to prove a difficult conjecture, auxiliary lemmata are obtained, which are subsequently proved and then added to the knowledge base.

In the current implementation, ***theory exploration*** on archives is provided as an experimental feature based on [Buchberger00] and [BuchbergerEtAl06], and it follows the bottom–up approach. We distinguish two types of exploration: New propositions are invented by **`Invent`**`␣`**`Propositions[arch,lib,sym]`**, new concepts are created by **`Invent`**`␣`**`Concepts[arch,lib,sym,new]`**. Here **`arch`** is the archive that should be expanded by the new propositions or definitions, **`lib`** is a library containing suitable invention schemes, **`sym`** a list of concepts from **`arch`** to be involved in the new propositions or definitions, and **`new`** a symbol to be defined. Note that **`lib`** is also realized as an archive, albeit with a rather special interpretation.

In the first step, the command **`InventPropositions[arch,lib,sym]`** extracts the types and arities of the symbols specified in **`sym`** from their usage in **`arch`**. Next it extracts the first proposition scheme from **`lib`** whose arguments are compatible with the types and arities in **`sym`** and instantiate it accordingly. This process is repetead until all proposition schemes in **`lib`** are exhausted. The resulting new propositions are returned by the command, an can then be fed into a suitably selected *Theorema* prover. The command **`InventConcepts[arch,lib,sym,new]`** proceeds in a similarly but the definition schemes in **`lib`** are assumed to have the newly defined notion as their last parameter. For the extraction step, the new notion denoted by **`new`** is appended to the symbols listed in **`sym`**.

Let us illustrate the algorithm for ***inventing propositions*** by a simple example. Consider the theory of groups formalized in **`grps`**:

$$\text{GrpThr} : \left\langle \in, \bigcap, \bigcup, \subseteq, \text{is–grp}, \text{is–subgrp}, \times, \text{is–normal–subgrp} \right\rangle \rightleftharpoons \underset{G}{\forall}$$

Definitions $\rightleftharpoons$

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

$\text{is–grp}[G] \Leftrightarrow \underset{x,y,z \in \#}{\forall}$

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

$x * 1 = x$

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

$\underset{S}{\forall}$

$\text{is–subgrp}[S, G] \Leftrightarrow$

$S \subseteq \#$

$\underset{x,y \in S}{\forall} x * y^{-1} \in S$

$\underset{N}{\forall}$

$\text{is–normal–subgrp}[N, G] \Leftrightarrow$

$N \subseteq \#$

$\underset{x \in N}{\forall} \underset{y \in \#}{\forall} y * x * y^{-1} \in N$

…

We have omitted the definition of direct product of × groups as given in Subsection 4.1.1.

Assume the library of schemes **lib** contains, among others, the proposition asserting that a unary or binary relation **R** is preserved by a binary operation ∘ and the scheme of recursive function definition à la divide–and–conquer:

Schemes ⇌

PropositionSchemes ⇌ $\underset{\circ}{\forall} \underset{R}{\forall}$

BinOpRespUnRelParam[R, ∘] ⇔

$\underset{a,b}{\forall} \underset{x}{\forall} (R[a, x] \wedge R[b, x] \Rightarrow R[a \circ b, x])$

$$\text{BinOpRespBinRel}[\circ, R] \Leftrightarrow \underset{a,b,c,d}{\forall}$$

$$R[a, b] \wedge R[c, d] \Rightarrow R[a \circ c, b \circ d]$$

…

$$\text{DefinitionSchemes} \rightleftharpoons$$

$$\underset{\text{is–spc}}{\forall} \underset{\text{left,right}}{\forall} \underset{\text{mrg}}{\forall} \underset{F}{\forall}$$

$$\text{Divide–Conquer}[\text{is–spc}, \text{left}, \text{right}, \text{mrg}, F] \Leftrightarrow$$

$$\underset{x}{\forall} F[x] = \begin{cases} x & \Leftarrow \text{is–spc}[x] \\ \text{mrg}[F[\text{left}[x]], F[\text{right}[x]]] & \Leftarrow \text{True} \end{cases}$$

…

Taking **sym** to be {**is–normal–subgrp**, ⋂}, the algorithm can pick the scheme **BinOpResp**∴**UnRelParam**, instantiate it accordingly (respecting the arity and the type of symbol) and obtain the conjecture:

$$\underset{G}{\forall} \underset{M,N}{\forall}$$

$$\text{is–normal–subgroup}[M, G] \wedge \text{is–normal–subgroup}[N, G] \Rightarrow \text{is–normal–subgroup}[N \bigcap M, G]$$

Passing this conjecture to a prover, it will fail, but by using a conjecture generator, the extra condition **is–grp[G]** will be suggested. With a sufficiently strong prover, the corresponding formula will be proved, so

$$\underset{\text{is–grp}[G]}{\forall} \underset{M,N}{\forall}$$

$$\text{is–normal–subgroup}[M, G] \wedge \text{is–normal–subgroup}[N, G] \Rightarrow \text{is–normal–subgroup}[N \bigcap M, G]$$

can be added to the knowledge base. Otherwise, it is discarded and the algorithm tries another scheme; the successful conjectures are appended to **arch** as theorems.

For the same archive **grps** and scheme library **lib**, by taking **sym** to be {**is–subgrp**, ×}, the algorithm can choose the scheme **BinOpRespBinRel**, instantiate it and obtain the conjecture:

$$\underset{G,H}{\forall} \underset{M,N}{\forall}$$

$$\text{is–subgroup}[M, G] \wedge \text{is–subgroup}[N, H] \Rightarrow \text{is–subgroup}[M \times N, G \times H]$$

Following the same steps, the extra conditions **is–grp[G]** and **is–grp[H]** are produced. Upon proving the modified conjecture with a strong prover, it will be added to the knowledge base.

The ***invention of concepts*** works in a similar fashion, except that the definition schemes require as an additional argument the new symbol to be defined (and of course no proof is necessary since we restrict ourselves to explicit definitions). A typical example of a definition scheme is **Divide–Conquer**, occurring in the scheme library given above.

The scenario "Logical Algorithm Retrieval = Symbolic Computation Proving" of [Buchberger03] can be described by a background archive **arch** on tuple theory.

TupleThr : ⟨is–trivial–tuple, left–split, right–split, merge, …⟩ ⇌

SpecialTuples ⇌

TrivialTuple ⇌

is–trivial–tuple[⟨⟩]

$\forall_{x}$ is–trivial–tuple[⟨x⟩]

$\forall_{x,y,\bar{x}}$ ¬ is–trivial–tuple[⟨x, y, $\bar{x}$⟩]

LeftAndRightSplits ⇌

left–split[⟨⟩] = ⟨⟩

right–split[⟨⟩] = ⟨⟩

$\forall_{x}$ (left–split[⟨x⟩] = ⟨x⟩)

$\forall_{x}$ (right–split[⟨x⟩] = ⟨⟩)

$\forall_{x,\bar{y},z}$ left–split[x, $\bar{y}$, z] = x ⌣ left–split[⟨$\bar{y}$⟩]

$\forall_{x,\bar{y},z}$ right–split[x, $\bar{y}$, z] = right–split[⟨$\bar{y}$⟩] ⌢ z

Merge ⇌

merge[⟨⟩, ⟨⟩] = ⟨⟩

$\forall_{y,\bar{y}}$ (merge[⟨⟩, ⟨y, $\bar{y}$⟩] = ⟨y, $\bar{y}$⟩) $\bigwedge$ $\forall_{x,\bar{x}}$ (merge[⟨x, $\bar{x}$⟩, ⟨⟩] = ⟨x, $\bar{x}$⟩)

$$\forall_{x,\bar{x}}\ \forall_{y,\bar{y}}$$

$$\text{merge}[\langle x, \bar{x}\rangle, \langle y, \bar{y}\rangle] = \begin{cases} x \frown \text{merge}[\langle \bar{x}\rangle, \langle y, \bar{y}\rangle] & \Leftarrow \quad x > y \\ y \frown \text{merge}[\langle x, \bar{x}\rangle, \langle \bar{y}\rangle] & \Leftarrow \quad \text{True} \end{cases}$$

Taking {**is–trivial–tuple**, **left–split**, **right–split**, **merge**} for **sym** and **merge–sort** for **new**, we obtain a definition of the merge–sort algorithm:

$$\forall_{X}$$

$$\text{merge–sort}[X] = \begin{cases} X & \Leftarrow \quad \text{is–trivial–tuple}[X] \\ \text{merge}[\text{merge–sort}[\text{left–split}[X]], \text{merge–sort}[\text{right–split}[X]]] & \Leftarrow \quad \text{True} \end{cases}$$

In principle, one could combine this approach with the **KnowledgeSynthesis** command of Section 5.4 for synthesizing the merge-sort algorithm as in the scenario “Logical Algorithm Retrieval = Algorithm Invention” of [Buchberger03]. But a full integration of the synthesis method, described in [BuchbergerCraciun03] and implemented in [Craciun08], is beyond the scope of the current implementation.

Using labels, theory exploration can also be carried out in a single archive **big–archive**. In this case, **arch** and **lib** are labels referring to suitable portions of **big–archive**. The propositions or definitions generated by the exploration will then be appended to **big–archive** under specific labels **NewPropositions** and **NewDefinitions**, respectively.

5 Retrieving Mathematical Knowledge in *Theorema*

As we explained in Section 2.2, there are various views on ***retrieval***, influenced by techniques from data mining and semantic web on the one hand (the "computer science wing") and pattern matching and theorem proving on the other hand (the "logic wing"). The notion of knowledge retrieval is still controversial, and numerous definitions can be found in the literature. In this chapter we study retrieval from a *Theorema* perspective (clearly leaning towards the "logic wing"), providing several operations both on flat knowledge bases and on archives.

In Section 5.1 we introduce the definition of formula retrieval and a classification of different retrieval types, in terms of increasing difficulty. Scenarios of retrieval occurring in theorem proving and algorithm synthesis are presented. In Section 5.2 we analyze syntactic retrieval, based on textual occurrence of symbols or pattern–matching. In Section 5.3 we treat semantic retrieval, a type of retrieval that involves simple inference steps. In Section 5.4 we present the retrieval commands implemented for plain *Theorema* knowledge bases as wells as for archives (we refer to both of them simply as knowledge bases). As explained in Section 4, archives have the advantage of allowing a more targeted search guided by the structure of home namespaces.

5.1 Retrieval in Theorem Proving and Algorithm Synthesis

In our framework, we regard retrieval as an integral part of theorem proving [Buchberger03], typically employing a suitably reduced set of inference rules for obtaining the formula needed. We refer to such reduced deduction steps as ***basic inferences***; other terms are "high–school proving", "symbolic computation proving", "physicists' proving", etc. We think this is a very natural viewpoint because textual search is usually not a sufficient solution—it will typically not find equivalent formulae or "simple" consequences. (For reasons of efficiency and as building blocks of complex retrieval strategies, it may occasionally be useful to apply simpler retrieval mechanisms closer to textual search, so these are also provided.)

As an ***example*** for the crucial idea "Retrieval = Basic Inferences", consider the following proof goal occurring in the course of algorithm verification:

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

One can retrieve the information required if the knowledge base contains the formula:

$$\forall_{x,y} ((x * y = 0) \Rightarrow (x = 0) \vee (y = 0))$$

But note that this retrieval involves the propositional tautology:

$$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$$

This tautology has to be handled by suitable basic inference steps.

Let us now fix some more terminology. As mentioned in Section 2.2, we define ***formula retrieval*** to be a search depending on the structure and content of mathematical knowledge. Given a target formula φ and a knowledge base $\mathcal{K}$, one has to check whether the formula φ "occurs" (i.e. it is easily obtainable from the formulae) in the knowledge base $\mathcal{K}$. More generally, given a formula φ describing properties of (unknown) functions and a knowledge base $\mathcal{K}$, find the necessary functions such that the formula φ "occurs" in $\mathcal{K}$ as a suitable instance. The notion of "occurrence" is vague; it could vary between literal text search and full–fledged proving. Our solution is in between these two extremes, allowing various kinds of basic inferences (in the sense explained above) for obtaining "easy" consequences from the knowledge base $\mathcal{K}$. This will usually include rewriting, α–conversion, propositional logic inferences, and generally all deduction steps that can be executed fast. The precise selection of basic inference steps can be adjusted by the user; it typically depends on the theory in which one is working [Buchberger03]. We provide one particular set of basic inference rules (Section 3) that can be considered as a universal retrieval engine and that may serve as a basis for specialized retrieval engines bound to various mathematical theories.

Sometimes it is also useful to issue a ***query*** to a knowledge base $\mathcal{K}$, which means the following: Given a list of (object / function / predicate) symbols, one wants to see all formulae of $\mathcal{K}$ that contain all/some of these symbols (see Section 2). Such queries can serve as syntactic filters on large knowledge bases, prior to the actual retrieval operations to be applied to them.

We distinguish two main types of formula retrieval:

- ***Syntactic retrieval*** (or explicit retrieval) is a textual search on characters and patterns; it realizes the trivial idea of literal occurrences in a knowledge base. Of course this becomes interesting only in conjunction with queries, i.e. looking for formulae containing certain symbols (called ***explicit occurrences***). A useful extension of this type of query employs applicative higher–order patterns for searching formulae by their syntactic skeleton (called ***explicit matching***).

- ***Semantic retrieval*** (or implicit retrieval) can be seen as finding a formula that follows by basic inference steps from the knowledge base. By introducing unknowns (see below) in the target formula, this allows to search for certain symbol names (e.g. functions) characterized by their properties (axioms, definitions, theorems etc.), provided the knowledge base is complete enough in the sense explained later in this section.

We deal with syntactic retrieval in Section 2 and with semantic retrieval in Section 3.

The above typology refers to the amount of proving power involved in the search algorithm, increasing from the no–proving case of syntactic retrieval via semantic retrieval to the other extreme of full–fledged theorem proving (which would not make sense for a retrieval mechanism). There is also an orthogonal typology, based on [Buchberger03], which classifies the amount of solving power in the search algorithm:

- ***Occurrence*** refers to the case when the target formula contains no unknown symbols, so no solving power is needed. As indicated above, we distinguish further between explicit and implicit occurrences, with full–fledged theorem proving as the limiting case.

- ***Matching*** is needed when the target formula contains unknown (object / function / predicate) symbols that can be instantiated with corresponding symbols from the knowledge base. Again we distinguish between explicit and implicit matching (i.e. literal matches versus matches involving basic inference steps), with general predicate–logic solving as the limiting case.
- ***Synthesis*** is involved if no symbols in the knowledge base can be found for the unknowns, and the latter can be instantiated with terms built from the signature of the knowledge base. In analogy to the previous case, there is a gradual transition from explicit via implicit to complex synthesis; see [BuchbergerCraciun03, Craciun08] for an example of the latter.

In the frame of this thesis, we have implemented the implicit and explicit cases of occurrence / matching / synthesis (see Section 4).

Another aspect to be considered is the amount of mathematical information available in the knowledge base. An efficient retrieval will depend also on a systematically structured build-up of theories. In this context, the notion of knowledge base ***completeness*** is crucial for identifying the basic inference steps in a specific theory. For example, in the theory of groups a complete knowledge base means having the ten axioms resulting from Knuth–Bendix completion, and basic inference steps are given by term rewriting.

A special case of formula retrieval is ***algorithm retrieval***, where the formula one is looking for is the specification of an algorithm [Buchberger03]. This notion is useful e.g. in algorithm synthesis by lazy thinking [Buchberger03a], where one starts a proof attempt of the specification with various unknowns denoting the ingredient functions; the terms to be substituted for the unknowns will then be algorithms built from the basic operations contained in the knowledge base.

Retrieval occurs naturally inside proving. As an example consider the ***verification condition generator*** (VCG) of *Theorema*. It generates from a program and its specification a list of verification conditions whose proof will ensure that the program is correct with respect to the specification [KovacsEtAl05]. The verification conditions yield proof obligations, where typically some parts are trivial, others more difficult, but most of them are somewhere in between—and thus typical retrieval tasks in the sense of applying basic inference steps. The verification conditions are usually very simple formulae from a mathematical perspective, so they should ideally be checked by an automated prover. But usually this is possible only with extra knowledge (added manually). What is therefore called for is a prover with retrieval support. Let us take a typical example of a problem generated in the process of verification:

$$\forall_{a,b,x,y} \; (\text{IsInteger}[a] \land \text{IsInteger}[b]) \Rightarrow$$

$$\text{IsInteger}[a] \land \text{IsInteger}[b]$$

$$\forall_{z}$$

$$((z := a) \land (a \geq b)) \lor (z := b) \land (b > a) \Rightarrow$$

$y + z \geq y + a$

$y + z \geq y + b$

If we attempt to prove this conjecture, a normal prover will fail, since the information provided is not enough. Nevertheless it simplifies it into four simpler proof obligations:

- The first one has the assumptions $z_0 := a_0,\ a_0 \geq b_0$ and the goal $y_0 + z_0 \geq y_0 + a_0$
- The second has the assumptions $z_0 := a_0,\ a_0 \geq b_0$ and the goal $y_0 + z_0 \geq y_0 + b_0$
- The third has the assumptions $z_0 := b_0,\ b_0 \geq a_0$ and the goal $y_0 + z_0 \geq y_0 + b_0$
- The fourth has the assumptions $z_0 := b_0,\ b_0 \geq a_0$ and the goal $y_0 + z_0 \geq y_0 + a_0$

For the human reader all four proof situations are trivial; for an automated prover, more assumptions about natural numbers are needed: In the first and third case, the formula $\underset{x}{\forall}\ x \geq x$ is needed and in the other two cases, the formula $\underset{x,a,b}{\forall}\ (a \geq b \Rightarrow a + x \geq b + x)$. A retrieval engine will obtain these extra assumptions essentially by applying matching and backchaining, respectively, on the goal together with the (huge) external knowledge base. As an extra step, useful for reducing the search space, one can filter the syntactically appropriate formulae (here the ones containing + or ≥) from the external knowledge base. More details about this idea are presented in Section 2.

Here is another simple example (this time of a more computational nature) also occurring during verification condition generation:

$$\underset{X}{\forall}\ (X \geq 0 \wedge \text{IsInteger}[X] \wedge (X := 0)) \Longrightarrow \left(0 = X * \frac{X+1}{2}\right)$$

A prover without retrieval support will simplify the proof obligation to the assumptions $x_0 \geq 0$, $x_0 := 0$, IsInteger$[x_0]$ and the goal $0 = 0 * \frac{0+1}{2}$, and it will get stuck at this point. The retrieval engine will then extract—by using backchaining—from the underlying knowledge base the formula $\underset{X}{\forall}\ (0 = 0 * X)$, from which the goal can be proved easily.

Another example of using retrieval is in the process of algorithm synthesis. For example, the process of synthesizing the merge–sort algorithm [BuchbergerCraciun03] starts by trying to prove

is–left–right–merge–structure$[l^?, r^?, m^?]$

for the unknowns $l^?, r^?, m^?$ under the assumptions

is–left–right–structure$[l^?, r^?] \bigwedge$ is–merge–structure$[m^?]$

Here we have no assumptions, so a prover cannot do anything. By using a retrieval engine, however, we obtain the definitions of **`is-left-right-structure`**, **`is-merge-structure`**, and **`is-left-right-merge-structure`** from the underlying knowledge base:

$$\forall_{l,r}\left(\text{is–left–right–structure}[l, r] \Leftrightarrow \forall_{\substack{\text{is–tuple}[X]\\ \neg\text{is–trivial–tuple}[X]}}\left\{\begin{array}{l} l[X] \prec X \\ \text{is–tuple}[l[X]] \\ r[X] \prec X \\ \text{is–tuple}[r[X]] \\ (l[X] \asymp r[X]) \approx X \end{array}\right.\right)$$

$$\forall_{m}\left(\text{is–merge–structure}[m] \Leftrightarrow \left\{\begin{array}{l} \forall_{\text{is–tuple}[Y,Z]}\ \text{is–tuple}[m[Y, Z]] \\ \forall_{\text{is–tuple}[Y,Z]}\left(\left\{\begin{array}{l}\text{is–sorted}[Y]\\ \text{is–sorted}[Z]\end{array}\right. \Rightarrow \left\{\begin{array}{l}(Y \asymp Z) \approx m[Y, Z]\\ \text{is–sorted}[m[Y, Z]]\end{array}\right.\right) \end{array}\right.\right)$$

$$\forall_{ls,rs,merged}\left(\begin{array}{l}\text{is–left–right–merge–structure}[ls, rs, merged] \Leftrightarrow \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right.$$

$$\left\{\begin{array}{l} \forall_{\substack{\text{is–tuple}[X]\\ \neg\text{is–trivial–tuple}[X]}}\left\{\begin{array}{l} ls[X] \prec X \\ \text{is–tuple}[ls[X]] \\ rs[X] \prec X \\ \text{is–tuple}[rs[X]] \end{array}\right. \\ \forall_{\text{is–tuple}[Y,Z]}\ \text{is–tuple}[merged[Y, Z]] \\ \forall_{\substack{\text{is–tuple}[X,Y,Z]\\ \neg\text{is–trivial–tuple}[X]}}\left(\left\{\begin{array}{l} ls[X] \approx Y \\ rs[X] \approx Z \\ \text{is–sorted}[Y] \\ \text{is–sorted}[Z] \end{array}\right. \Rightarrow \left\{\begin{array}{l} merged[Y, Z] \approx X \\ \text{is–sorted}[merged[Y, Z]] \end{array}\right.\right) \end{array}\right.\right)$$

Thus the proof can proceed until it gets stuck again, with the goal $m[Y_0, Z_0] \approx X_0$ and a list of assumptions containing the statements that $l[X_0]$, $r[X_0]$, Y_0 and Z_0 are tuples and the following formulae:

$$(Y_0 \asymp Z_0) \approx m[Y_0, Z_0]$$

$$l[X_0] \approx Y_0 \wedge r[X_0] \approx Z_0$$

Based on syntactic criteria, a retrieval engine will now extract the following formulae from the external knowledge base:

$$\underset{\text{is–tuple[A,B,Y,Z]}}{\forall} ((A \approx Y \wedge B \approx Z) \Rightarrow ((A \asymp B) \approx (Y \asymp Z)))$$

$$\underset{\text{is–tuple[X,Y,Z]}}{\forall} (X \approx Y \wedge Y \approx Z \Rightarrow X \approx Z)$$

$$\underset{\text{is–tuple[A,B,Y,Z]}}{\forall} (X \approx Y \Rightarrow Y \approx X)$$

Now the prover can continue, finally obtaining the desired goal formula. Further examples will be provided in the next sections.

Finally let us note that a simple solution for retrieval in archives, enabling us to use the results on plain *Theorema* knowledge bases, would be to collapse the tree structure and feed the resulting assumption list into the flat version of retrieval or even into a *Theorema* prover. But using archives—in their hierarchical form—one can reduce the search space considerably.

5.2 Syntactic Retrieval

The simplest case of syntactic retrieval is the ***explicit occurrence*** of a formula in the knowledge base. Given a formula containing certain (object, function, predicate) constant symbols, a list of theory constants can be extracted and the given knowledge base can be filtered against these constants, returning to the user all the appropriate formulae from the knowledge base. A variant of this type of retrieval allows as input a list of symbols and the given knowledge base.

This constant symbols are ***theory constants*** in the following sense: Since we allow higher-order formulae, we consider higher–order functions and predicates as constants as long as they are not quantified. In contrast, the connectives $\neg, \Rightarrow, \wedge, \ldots$ are sometimes referred to as logical constants.

Preprocessing the knowledge base can be useful for speeding up the search process. For every formula in the archive, one obtains a list of its positive and negative literals, the predicate / function / object constants occurring in it, and also a measure of ***priority*** for each symbol (the less a symbol occurs, the higher its priority). For example, consider the following archive in analysis:

$$\text{RealAnalysis} : \langle +, -, *, /, 0, 1\rangle \rightleftharpoons \ldots$$

$$\text{ExponentialFunction} : \langle \exp\rangle \rightleftharpoons \underset{x,y}{\forall}$$

$$\exp[x + y] = \exp[x] * \exp[y]$$

$$\exp[0] = 1$$

$$(\exp[x] = 1) \Rightarrow (x = 0)$$

$$\neg (\exp[x] = 0)$$

Here the preprocessor would annotate the archive with the following information:

Symbol	Arity	Priority
+	2	40
∗	2	25
0	0	50
1	0	30
exp	1	15

The priority of symbols will be calculated based on how often they occur in the knowledge base: Since **exp** occurs much more seldom than e.g. +, it will get a higher priority (the highest priority is 1). Also, for each of the formulae, the positive and negative literals will be stored:

{exp[x + y] = exp[x] ∗ exp[y]}, {}

{exp[0] = 1}, {}

{exp[x] = 1}, {x = 0}

{}, {exp[x] = 0}

The most natural type of filtering, called **AndFiltering**, extracts formulae containing all the constants in the input formula. When it succeeds, one ends up with “similar” formulae (in a syntactic sense). If instead one wants to have all formulae containing just any of the constants occurring in the target formula, the **OrFiltering** can be applied. A third variant, called **PriorityFiltering**, takes into account the priority measure mentioned above and orders the results accordingly.

Consider the example of looking for an explicit occurrence of the predicate constant **is–Church–Rosser** within the knowledge base formalizing the theory of Gröbner Bases. Here we obtain, among others, the famous Newman's Lemma:

$$\forall_{\text{is–Noetherian}[\longrightarrow]} \text{is–Church–Rosser}[\longrightarrow] \Leftrightarrow \forall_{f,f1,f2} ((f \longrightarrow f1) \wedge (f \longrightarrow f2) \Rightarrow f1 \downarrow^{*} f2)$$

A more interesting example, where more constant symbols are involved, is the following formula from tuple theory:

$$\forall_{\text{is–tuple}[X]} \text{is–sorted–version}[X, \text{merging–by–sorting}[X]]$$

Looking for an explicit occurrence here amounts to searching for the theory constants **merging–by–sorting**, **is–sorted–version**, **is–tuple**. Assuming a priority filtering was chosen, this would yield formulae like these:

$$\forall_{\text{is–tuple[Y]}} \text{is–tuple[merging–by–sorting[Y]]}$$

$$\forall_{\text{is–tuple[Y]}} (\text{Y} \approx \text{merging–by–sorting[Y]})$$

$$\forall_{\text{is–tuple[Y]}} \text{is–sorted[merging–by–sorting[Y]]}$$

Note that **`merging-by-sorting`** has a higher priority than the other two. If nothing about this symbol is available in the knowledge base, a priority filtering would at least find the definition of **`is-sorted-version`**:

$$\forall_{\text{is–tuple[X],Y}} \text{is–sorted–version[X, Y]} \Longleftrightarrow \begin{cases} \text{is–tuple[Y]} \\ \text{X} \approx \text{Y} \\ \text{is–sorted[Y]} \end{cases}$$

Since one of *Mathematica*'s strengths is its built–in mechanism for pattern matching, it is natural to generalize its capabilities to ***explicit matching*** in the sense explained in Section 1. What is implemented is effectively a wrapper around *Mathematica*'s pattern–matching functions. As an example, suppose we have the group axioms:

$$\text{GroupTheory} : \langle *, 1, \text{OverHat} \rangle \rightleftharpoons$$

$$\text{Axioms} \rightleftharpoons \forall_{\text{x,y,z}}$$

$$(\text{x} * \text{y}) * \text{z} = \text{x} * (\text{y} * \text{z})$$

$$(\text{x} * 1 = \text{x}) \wedge (1 * \text{x} = \text{x})$$

$$(\text{x} * \hat{\text{x}} = 1) \bigwedge (\hat{\text{x}} * \text{x} = 1)$$

For checking whether in this archive we have a theorem of the form $(\Phi^? = 1) \bigwedge (\Psi^? = 1)$, it is sufficient to define a function that finds the positions in which this formula appears inside the archive (or in a flat knowledge base). In this case, it will return:

$$\forall_{x} ((x * (\hat{x}) = 1) \bigwedge ((\hat{x}) * x = 1))$$

On the other hand, if the user wants to search for terms of the form $\Phi^? * \Psi^? = \Gamma^? * \Delta^?$, the matching obtains only this formula:

$$\forall_{x,y,z} ((x * y) * z = x * (y * z))$$

Looking in the same archive for $\Phi^?$=1 will yield

$$\underset{x}{\forall}\ ((x * (\hat{x}) = 1) \bigwedge ((\hat{x}) * x = 1))$$

$$\underset{x}{\forall}\ ((x * (\hat{x}) = 1) \bigwedge ((\hat{x}) * x = 1))$$

because both the first and the second operand are matched.

Another simple example appears in analysis in the course of convergence proofs. Assuming one wants to prove is–convergent[g_0] for an arbitrary sequence g_0, one might want to find explicit matches for the pattern is–convergent[$g^?$] ⇔ EquivCond$^?$. Thus we would obtain the Cauchy criterion:

$$\underset{\text{is–sequence[f]}}{\forall}$$

$$\text{is–convergent[f]} \Leftrightarrow \underset{\varepsilon>0}{\forall}\ \underset{N}{\exists}\ \underset{m\geq N}{\forall}\ \underset{n\geq N}{\forall}\ |f[m] - f[n]| < \varepsilon$$

The proof can now proceed in the usual way, taking ε_0 arbitrary but fixed, etc.

But we enter into trouble as soon as we want something just slightly more complicated. For example, we may want to search for a formula that involves some unknown function special$^?$ in the following way:

$$\underset{\text{is–tuple[X]}}{\forall}\ \text{is–trivial–tuple[X]} \Rightarrow \text{is–sorted–version[X, special}^?\text{[X]]}$$

Now assume we have somewhere in the knowledge base:

$$\underset{\text{is–tuple[X]}}{\forall}\ (\text{is–trivial–tuple[X]} \Rightarrow \text{is–sorted–version[X, X]})$$

We expect to get the answer special$^?$[X]⟵X. Simple as this may be, it is not in the scope of explicit matching. In fact, this example is already an instance of ***explicit synthesis***. Here the solution is to replace the unknown function special$^?$ with a new existentially quantified variable $v_{special}$, with the existential quantifier inserted exactly after the universal quantifiers on X:

$$\underset{\text{is–tuple[X]}}{\forall}\ \underset{v_{special}}{\exists}\ (\text{is–trivial–tuple[X]} \Rightarrow \text{is–sorted–version[X, } v_{special}])$$

Now the formula will be further transformed by introducing a Skolem function for the universally quantified variable X. In general, each such Skolem function will have as arguments the existentially quantified variables occurring before the universal variable it represents; in this case we had none:

$$\underset{v_{special}}{\exists}\ (\text{is–tuple[}X_0\text{]} \wedge \text{is–trivial–tuple[}X_0\text{]} \Rightarrow \text{is–sorted–version[}X_0\text{, } v_{special}])$$

In the next step the existential quantifiers are eliminated by introducing metavariables as described in [BuchbergerEtAl06,§4]:

$$\text{is–tuple}[X_0] \wedge \text{is–trivial–tuple}[X_0] \Rightarrow \text{is–sorted–version}[X_0, v^*_{special}]$$

The knowledge base is in the meanwhile syntactically filtered for the symbols **`is-sorted-version`**, **`is-trivial-tuple`**, **`is-tuple`** and Skolemized in the usual way. For example, the formula stating that trivial tuples are sorted becomes:

$$\text{is–tuple}[X^*] \wedge \text{is–trivial–tuple}[X^*] \Rightarrow \text{is–sorted–version}[X^*, X^*]$$

By first-order unification, we obtain bindings for the universal variables in the knowledge base assumptions and the metavariables in the target formula, namely $X^* \leftarrow X_0$ and $v^*_{special} \leftarrow X^*$. Thus $v^*_{special}$ is chosen as X_0, and special$^?$ as the identity function.

We note that one can also view the step from solving to synthesis as allowing ***λ–terms*** in the bindings for the unknown functions, but we prefer to see matters from a slightly different viewpoint: For the moment we assume that the target formula is a quantified formula (without loss of generality, we may assume it in prenex normal form) containing an occurrence of one or more unknown functions, for example

$$\forall_{x_1} \ldots \forall_{x_n} \phi\,(F^?[x_1, \ldots, x_n])$$

The unknown function is replaced by a new existentially quantified variable v_F, with the existential quantifier inserted exactly after all quantifiers corresponding to $x_1, \ldots, x_k$; we obtain $\forall_{x_1} \ldots \forall_{x_n} \exists_{v_F} \phi(v_F)$ for the example above. This formula is then transformed by introducing Skolem functions for the universally quantified variables (each Skolem function having as arguments the existential quantified variables occurring before the respective universal variable). Finally, the existential quantifiers are eliminated by introducing metavariables, on which first-order unification against the external knowledge bas is applied (after Skolemizing it in the usual way): In the successful case, the unifier will contain bindings both for the universal variables in the assumptions of the knowledge base and for the metavariables in the target formula.

5.3 Semantic Retrieval

Retrieval usually goes beyond literal occurrence checks and mere pattern matching. Even the ***expansion of definitions*** is a simple, but often needed instance of semantic retrieval that searches for an implicit occurrence. For example, proving

$$\forall_{\text{is–tuple}[X]} \text{is–sorted–version}[X, \text{merging–by–sorting}[X]]$$

under the assumptions

$$\forall_{\text{is–tuple[Y]}} \left\{ \begin{array}{c} \text{is–tuple[merging–by–sorting[Y]]} \\ \text{Y} \approx \text{merging–by–sorting[Y]} \\ \text{is–sorted[merging–by–sorting[Y]]} \end{array} \right.$$

with the external knowledge base

$$\forall_{\text{is–tuple[X],Y}} \text{is–sorted–version[X, Y]} \Longleftrightarrow \left\{ \begin{array}{c} \text{is–tuple[Y]} \\ \text{X} \approx \text{Y} \\ \text{is–sorted[Y]} \end{array} \right.$$

is just a simple retrieval step (note that the assumptions are to be understood as being adjoined to the knowledge base).

Another simple case of finding implicit occurrences is needed when the user is looking for formulae identical with the formulae in the archive modulo bound variables; this step is usually referred to as ***α–conversion***. Let us explain how this works in a simple example. If one has to prove commutativity of addition in the form $\forall_{x,y} (x + y = y + x)$, but the external knowledge base contains the α–variant $\forall_{a,b} (a + b = b + a)$, the first step of the retrieval engine is to replace the universally quantified variables **x** and **y** by the corresponding Skolem constants x_0 and y_0 such that the resulting instance can immediately be matched against the knowledge base by the binding $a \leftarrow x_0$ and $b \leftarrow y_0$.

A slight variation of this mechanism is needed when the universal quantifier is ***relativized*** by a condition. Here is an example appearing in the verification of the merge–sort algorithm: One has to prove is–tuple[m[sm[l[x_0]], sm[r[x_0]]]] and is–sorted[m[sm[l[x_0]], sm[r[x_0]]]] for arbitrary x_0 under the assumptions is–tuple[sm[l[X_0]]] and is–tuple[sm[r[X_0]]], with the knowledge base containing:

$$\forall_{\text{is–tuple[X,Y]}} (\text{is–sorted[m[X, Y]]} \wedge \text{is–tuple[m[X, Y]]})$$

The proof needed is just a simple retrieval step (note that the assumptions are to be understood as being adjoined to the knowledge base).

In this case, the syntactic filtering techniques applied in the explicit synthesis of Section 2 do not help. The point is that interesting information does not necessarily satisfy syntactic criteria. Some ***basic inference rules***, like propositional inference rules, instantiation, arbitrary but fixed, replacement are used for obtaining the formula from the knowledge base. Since difficult proving steps like induction or finding term instances for existential goals would substantially decrease the speed of the search engine dramatically, they are not included. Note also that the VCG examples described in Section 1 are about searching implicit occurrences.

Let us summarize this discussion by sketching the ***inference calculus*** of these basic deduction steps. Most rules are directed at transforming the goal formula, so let us describe the essential ones of this kind in the form of a natural deduction calculus:

$$\frac{\Phi_{x \leftarrow x_0}}{\forall_x \Phi} \quad \frac{\Phi \quad \Psi}{\Phi \wedge \Psi} \quad \frac{\Phi}{\Phi \vee \Psi} \quad \frac{\Psi}{\Phi \vee \Psi}$$

While the knowledge base is untouched by these rules, the deduction rule involves both the assumptions and the goal formula if the latter has the form $\Phi \Rightarrow \Psi$, the new goal becomes Ψ while Φ is adjoined to the knowledge base as a local assumption (similar to the example at the beginning of this section). The deduction rule is accompanied by backchaining rules operating on the collection of positive and negative literals extracted by the preprocessor (see the beginning of Section 2). In order to avoid working with existential quantifiers, we assume that the external knowledge base is already Skolemized (otherwise this is done in a separate initialization step), transforming formulae of the form $\underset{x1}{\forall} \ldots \underset{xn}{\forall} \underset{y}{\exists} \Phi$ into $\underset{x1}{\forall} \ldots \underset{xn}{\forall} \Phi_{y \leftarrow f_y[x1,\ldots,xn]}$. By the same token we assume that the knowledge base has been transformed into clause form. Note also that the environment (typically a prover) calling the retrieval engine is expected to distribute negations in the goal. The terminal inference rules are of course

$$\frac{\Phi \qquad \underset{x1}{\forall} \ldots \underset{xn}{\forall} \Phi}{\Phi_{x1 \leftarrow \tau 1, \ldots xn \leftarrow \tau n}}$$

with the upper formulae occurring in the knowledge base and the lower ones denoting the goal. Furthermore, we allow input resolution [ChangLee73, p. 132] with the local assumptions serving as input clauses (thus including the two terminal rules above). Suitable instances of the replacement rule round up this simple example of a retrieval engine. Of course there are many variations and optimizations that can be applied to the above nucleus of basic inferences; our objective at this point was to give the flavor of a typical inference system for retrieval.

As an example, consider a knowledge base on order theory that contains—among other formulae—the law of trichotomy in the following form:

$$\underset{x,y}{\forall} (x < y \vee (x = y) \vee y < x)$$

Now the user wants to find the following slight modification:

$$\underset{a,b}{\forall} (a < b \vee b < a \Rightarrow (a = b))$$

In this case, the target formula will be instantiated with arbitrary but fixed constants a_0, b_0; then the deduction rule is applied, creating the local assumptions $a_0 < b_0$ and $b_0 < a_0$. By applying ***input resolution*** twice between these and the external knowledge base in clausal form, one obtains immediately $a_0 = b_0$.

Implicit matching can be reduced to searching implicit occurrences. The extra steps involve finding appropriate candidates for the unknown symbols: They are extracted from the goal formula, together with their arity.The list of unknown symbols is then matched against the knowledge base signature. The possible symbols are chosen, and appropriate instances of the target formula are created. For each of these formulae, implicit occurrences are searched.

Consider again as an example the synthesis of merge–sort explained in more detail in Section 1. Here the target formula

$$\text{is–left–right–merge–structure}[l^?, r^?, m^?]$$

can thus be instantiated with function symbols in the knowledge base. From the many possible instantiations, only the following ones are implicit occurrences:

$$\text{is–left–right–merge–structure}[\text{left–split}, \text{right–split}, \text{merged}]$$

$$\text{is–left–right–merge–structure}[\text{odd–split}, \text{even–split}, \text{merged}]$$

The other formulae are discarded.

For reducing ***implicit synthesis*** to searching implicit occurrences, one proceeds similarly as in the case of explicit synthesis (see the end of Section 2), but the resulting target formula containing the metavariables is now subjected to the basic inference calculus explained above, except that one will in general need unification instead of just matching in the terminal inference rules.

Consider the following example occurring in a synthesis of the Groebner basis algorithm [Buchberger04a]. Omitting restriction predicates like **`is–power–product`** and **`is–polynomial`** for the sake of simplicity, the crucial point in the synthesis requires proving

$$\forall_{p}\ \forall_{g,h}\ ((\text{lp}[g] \mid p) \bigwedge (\text{lp}[h] \mid p \Rightarrow F^?[g, h] \mid p))$$

with the background archive typically containing a formula like (note that we have assumed **`s`** and **`t`** reversed here):

$$\forall_{p}\ \forall_{s,t}\ ((t \mid p) \wedge (s \mid p) \Rightarrow (\text{lcm}[s, t] \mid p))$$

Here **`lp`** and **`lcm`** refer respectively to the leading power product of a polynomial and to the least common multiple of two power products. The target formula is reduced to $(\text{lp}[g_0] \mid p_0) \wedge (\text{lp}[h_0] \mid p_0) \Rightarrow (v_F^* \mid p_0)$, with v_F^* a new metavariable. Applying three times input resolution on the assumption clause $\neg (t^* \mid p^*) \vee \neg (s^* \mid p^*) \vee (\text{lcm}[s^*, t^*] \mid p^*)$ with metavariables (in this context also known as free variables) p^*, s^*, t^*, one succeeds with the following unifier:

$$p^* \leftarrow p_0,\ s^* \leftarrow \text{lp}[g_0],\ t^* \leftarrow \text{lp}[h_0],\ v_F^* \leftarrow \text{lcm}[\text{lc}[g_0], \text{lc}[h_0]]$$

Thus $F^?[g, h]$ has been found to be lcm[lc[g], lc[h]].

5.4 Tools for Retrieval in *Theorema*

The current implementation of retrieval tools in *Theorema* should only be understood as a first step in the direction of realizing retrieval as a suitably restricted case of theorem proving; it allows *Theorema* provers to execute retrieval operations on different levels of sophistication:

- The operation **KnowledgeLookup** filters out formulae containing given symbols (implementing the explicit–occurrence search).
- Using **KnowledgeMatch** allows applicative higher–order patterns in matching.
- For extending the patterns to work on unification—and thus synthesizing functions—one can apply the command **KnowledgeSynthesis**.
- Checking whether a formula follows from the knowledge base by basic inferences is achieved by **KnowledgeImplicitLookup**.
- Finding functions with desired properties is covered by **KnowledgeImplicitMatching**.
- If the function is to be synthesized, one may use **KnowledgeImplicitSynthesis**.

All these six commands can be used both on plain *Theorema* knowledge bases and on archives; the structure of the input allows to determine automatically whether one works on flat or hierarchical knowledge bases. In the sequel, we will refer to these two versions as the plain versus archive commands.

Let us now add a few remarks on the usage of these operations. The archive command

```
KnowledgeLookup[arch, sym1, sym2, …]
```

filters **arch** for the list of (object / function / predicate) constants **sym1**, **sym2**,…, returning all positions (in the sense introduced in Section 4.2) of formulae that contain the symbols.

With the help of the option **Check**, the user can reduce the search space. The default setting is **Check⟶AllArchive**, meaning the check is done in the whole archive. A more restricted (and faster) search can be accomplished by setting **Check⟶HomeNamespaces**. Then the constant symbols will be checked only in home namespaces (see Subsection 3.4.2). This feature is useful if the user has built the archive in such a way that all crucial properties—in particular, the definitions—of a symbol are asserted under the label designating its home namespace. An example of this usage is given by the concatenation symbol $\asymp$ introduced in Section 4.2.

Another option, named **Filtering**, can be set either to **OrFiltering**, **AndFiltering** or **PriorityFiltering**. The first searches formulae containing at least one of the constant symbols specified, the second requires them to contain all of them, and the last performs a filtering based on symbol priorities (see Section 2). For example, the command

```
KnowledgeLookup[arch, {+, ∗}, FilterType → AndFiltering]
```

returns a list of positions for the formulae containing both + and $*$, for example associativity in rings. By subsequently using **ArchivePart** for each of these positions, one can obtain the corresponding subarchives or subformulae.

The plain command **KnowledgeLookup**, operating on flat *Theorema* knowledge bases, is included among the retrieval functions for the Label Management framework described in [PiroiEt-Al08].

If one wants to search in an archive **arch** for formulae having a certain "skeleton" **form**, one can use **KnowledgeMatch[arch,form]**. The skeleton may be an arbitrary formula of the archive language, except that they may also contain metavariables designated by a superscripted question mark (subsequently transformed to patterns of *Mathematica*, to be interpreted by its applicative higher-order matching algorithms). Suppose, for example, that we want to search for an invariant $S^?$ with respect to a fixed equivalence relation $\approx$. Then we would employ the following skeleton:

$$\forall_{a,b} (a \approx b \Rightarrow (S^?[a] = S^?[b]))$$

With its default setting **MatchSubformulae⟶False**, matching is restricted to complete *Theorema* formulae (i.e. the leaves of the $\rightleftharpoons$ hierarchy). For descending into the internal structure of the *Theorema* formulae, one can set **MatchSubformulae⟶True**. For example, the above skeleton characterizing invariants could occur in the characteristic property of canonical simplifiers:

$$\forall_{x} \sigma[x] \approx x \bigwedge \forall_{x,y} (x \approx y \Rightarrow (\sigma[x] = \sigma[y]))$$

The plain command **KnowledgeMatch** for flat *Theorema* formulae is again included in the retrieval functions for the LabelManagement tools [PiroiEtAl08]. If term construction should be attempted for searching the "skeleton" **form** in **arch**, one uses the command **KnowledgeSynthesis[arch,form]** explained in Section 2.

As stated in Section 1, we see retrieval in its broader sense as a sequence of basic inferences. This idea is embodied in the commands **KnowledgeImplicitLookup**, **KnowledgeImplicitMatching** and **KnowledgeImplicitSynthesis**, which can therefore be seen as issuing a semantic search for formulae/functions specified by certain properties. Accordingly, the command **KnowledgeImplicitLookup** checks whether a formula "follows easily" from a knowledge base (see Section 3), and it is needed in the context of theorem proving (specifically in the program verification). The other two types of search are typically needed in the context of algorithm synthesis [Buchberger03]: Whereas **KnowledgeImplicitMatching** associates the functions only with specific function constants already present in the archive, **KnowledgeImplicitSynthesis** can construct terms denoting the functions.

The first type of semantic search is initiated by **KnowledgeImplicitLookup[arch,form]**, where **form** is now a *Theorema* formula. For example, consider again the user performing the synthesis of merge–sort. The available archive **arch** on tuple theory contains, among other things the transitivity and symmetry of $\approx$ and the following property:

$$\forall_{\text{is–tuple}[A,B,Y,Z]} (A \approx Y \wedge B \approx Z \Rightarrow (A \asymp B) \approx (Y \asymp Z))$$

Assume the user wants to check:

$$\forall_{\text{is–tuple}X,Y,Z}$$

$$\bigwedge \left\{ \begin{array}{c} (Y \asymp Z) \approx \text{merged}[Y, Z] \\ Y \approx \text{left–split}[X] \wedge Z \approx \text{right–split}[X] \\ (\text{left–split}[X] \asymp \text{right–split}[X]) \approx X \end{array} \right\} \Rightarrow \text{merged}[Y, Z] \approx X$$

In this case, **`KnowledgeImplicitLookup`** will indeed return true.

If the user searches for unknown functions, she will use **`KnowledgeImplicitMatching[arch,form]`**, where the unknown functions in **`form`** are designated as for **`KnowledgeMatch`**. In the example of synthesizing the merge-sort algorithm [BuchbergerCraciun03], at a certain stage one has to come up with two functions on tuples, $f^?$ and $g^?$, such that the concatenation of $f^?[X]$ and $g^?[X]$ is a permuted version of X. Written in the formalism employed there, this means one takes an arbitrary tuple X_0 and searches

$$(f^?[X_0] \asymp g^?[X_0]) \approx X_0$$

in an archive on tuple theory, which will typically contain formulae like:

$$\forall_{\text{is–tuple}[T]} \text{left–part}[T] \asymp \text{right–part}[T] = T$$

$$\forall_{\text{is–tuple}[T]} T \approx T$$

Extracting from the archive all unary function symbols (including here **`left-part`** and **`right-part`**) and substituting them for $f^?$ and $g^?$, the algorithm will then proceed to prove the resulting candidate instances by using a suitably restricted set of inferences. In order to restrict the search space further, various selection heuristics are studied (e.g. filtering the knowledge base by the preprocessing explained in Section 2).

The command **`KnowledgeImplicitSynthesis[arch,form]`** is used if the unknown functions in **`form`** cannot be instantiated with appropriate function symbols from **`arch`**. One then combines the synthesis with the implicit–occurrence search as explained in Section 3. For the above target formula, a knowledge base containing the formula

$$\forall_{\text{is–tuple}[X]} (X \asymp \langle\rangle) \approx X$$

will lead to the bindings $f^?[X_0] \leftarrow X$ and $g^?[X_0] \leftarrow \langle\rangle$ and $X \leftarrow X_0$, while the condition is–tuple$[X_0]$ has to be present in the list of local assumptions.

6 Conclusion

We view the present thesis as a first step in the general undertaking of developing a logic–internal paradigm for doing MKM in a computer–supported environment. We have introduced the notions of labels and namespaces, which facilitate a structured representation of a mathematical knowledge base; we have called such a knowledge base an archive. Based on the *Theorema* language, archives offer constructs for splitting formulae in multiple cells, with quantifiers ranging over whole cell groups and labels for attaching names to the groups. This makes archives very readable and particularly suited 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:

- domains as used for categories and functors, and
- the intuitive usage of “contexts” for resolving ambiguous symbols.

A decisive feature of archives is that all its language constructs, in particular the symbols $\rightleftharpoons$ for attaching labels and : for declaring namespaces, are logic-internal—they extend *Theorema* in such a way that there is a natural translation back to plain *Theorema*.

We have also introduced various crucial operations on archives, notably mathematical knowledge retrieval and theory exploration. A lot of work is still necessary here, in particular for elaborating the view of retrieval as a calculus of basic inference rules. As noted in our treatment of this topic, this calculus could and should be extended by various more efficient deduction rules that are custom–tailored for certain retrieval scenarios. The need for such additional rules becomes more pronounced when retrieval is performed within specific theories like fragments of elementary arithmetic or metric spaces. Regarding the operations of theory exploration, we have only scratched the surface; specifically its combination with the cascade mechanism for fixing failed proofs deserves further study.

Apart from its immediate merits, the concepts and tools developed in this thesis should also be understood as providing new momentum to the ongoing discussion of how one can capture the logical organization of mathematical repositories in an effective and natural way. In fact, the logic–internal approach put forth in this thesis should be seen as complementing the numerous logic–external treatments existing across MKM communities. We hope this will also shed light on other structural issues of MKM.

Appendix: Formalization Example in Lattice Theory

We present here a formalization of Chapter XIV (consisting of §§1—9) from [MacLaneBirkhoff67]. We have formalized all proclaimed assertions (e.g. Definition, Theorem, Lemma, Corollary), assuming an analogous formalization for all the remaining material occurring in the running text.

$$\S1\text{–Posets–DualityPrinciple} : \langle \text{is–poset, dual, is–chain, cover–relation} \rangle \rightleftharpoons \underset{P}{\forall}$$

$$P : \langle \#,\ \leq,\ \geq,\ <,\ > \rangle$$

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

$$\underset{x,y,z\in\#}{\forall}$$

$$x \leq x$$

$$x \leq y \wedge y \leq x \Rightarrow (x = y)$$

$$x \leq y \wedge y \leq z \Rightarrow x \leq z$$

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

$$x < y \Leftrightarrow (x \leq y \wedge x \neq y)$$

$$x \geq y \Leftrightarrow y \leq x$$

$$x > y \Leftrightarrow (y \leq x \wedge x \neq y)$$

$$P : \langle \text{is–chain} \rangle$$

$$\text{is–chain} \Leftrightarrow \underset{x,y\in\#}{\forall} (x \leq y \vee y \leq x)$$

$$\underset{X}{\forall}\ \underset{a,b}{\forall}\ \underset{O,I}{\forall}$$

$$P : \langle \text{is–least, is–greatest, is–minimal, is–maximal, is–lower–bound, is–upper–bound} \rangle$$

$$\text{is–least}[a, X] \Leftrightarrow$$

$$a \in X$$

$$\underset{x\in X}{\forall}\ a \leq x$$

is–greatest[b, X] ⇔

$b \in X$

$\forall_{x \in X}\ b \geq x$

is–lower–bound[O] ⇔ is–least[O, #]

is–upper–bound[I] ⇔ is–greatest[I, #]

is–minimal[a, X] ⇔

$a \in X$

$\forall_{x \in X}\ (x \leq a \Rightarrow x = a)$

is–maximal[b, X] ⇔

$b \in X$

$\forall_{x \in X}\ (b \leq x \Rightarrow x = b)$

$\text{is–chain[P]} \Leftrightarrow \left(\text{is–poset[P]} \bigwedge \text{is–chain}_{P}\right)$

$\text{Proposition1} \rightleftharpoons \forall_{X \subseteq \#}$

P : ⟨is–minimal, is–maximal⟩

NaturalNumbers : ⟨is–finite⟩

$\text{is–finite[X]} \bigwedge X \neq \emptyset \Rightarrow \exists_{a,b}\ (\text{is–minimal[a, X]} \wedge \text{is–maximal[b, X]})$

▪$\text{Lemma} \rightleftharpoons \forall_{\text{is–chain[C]}}$

C : ⟨#, is–minimal, is–maximal, is–least, is–greatest⟩

NaturalNumbers : ⟨is–finite⟩

$\forall_{X \subseteq \#}\ \forall_{a}\ (\text{is–minimal[a, X]} \Leftrightarrow \text{is–least[a, X]})$

$\forall_{X \subseteq \#}\ \forall_{b}\ (\text{is–maximal[b, X]} \Leftrightarrow \text{is–greatest[b, X]})$

is–finite[#] $\bigwedge$ # $\neq \emptyset \Rightarrow \underset{a,b}{\exists}$ (is–least[a, #] $\wedge$ is–greatest[b, #])

Proposition2 $\rightleftharpoons \underset{C}{\forall}$

PosetBasics : $\langle \approx \rangle$

PosetExamples : ⟨fin–chain⟩

NaturalNumbers : ⟨is–finite⟩

is–chain[C] $\wedge$ is–finite[C] $\Rightarrow$

$C \approx$ fin–chain$\left[\underset{C}{\#}\right]$

Theorem3 $\rightleftharpoons$

is–poset[P] $\Rightarrow$ is–poset[$\lceil \# \mapsto \#,\ \leq \mapsto \geq,\ < \mapsto >,\ \geq \mapsto \leq,\ > \mapsto < \rceil$]

Duality $\rightleftharpoons$

dual[P] = $\lceil \# \mapsto \#,\ \leq \mapsto \geq,\ < \mapsto >,\ \geq \mapsto \leq,\ > \mapsto < \rceil$

$\underset{P,Q}{\forall}\ \underset{f}{\forall}$

is–dualmorphism[f, P, Q] $\Leftrightarrow$

$f : \underset{P}{\#} \rightarrow \underset{Q}{\#}$

$\underset{x,y \in \underset{P}{\#}}{\forall} \left(x \underset{P}{\leq} y \Rightarrow f[x] \underset{Q}{\geq} f[y] \right)$

CoverRelation $\rightleftharpoons$

cover–relation[P] : ⟨covers⟩

$\underset{a,b}{\forall}$

covers[a, b] $\Leftrightarrow$

$a > b$

$\nexists_{x}$ $(a > x \wedge x > b)$

§2–LatticeIdentities : ⟨lattice–from–poset, poset–from–semilattice,
poset–from–lattice, is–lattice–order, is–lattice, is–semilattice⟩ ⇌

LatticeFromPoset ⇌ $\underset{P}{\forall}$

P : ⟨#, ≤⟩

lattice–from–poset[P] : ⟨ex–meet, ex–join, $\bigsqcap$, $\bigsqcup$⟩

lattice–from–poset[P][#] = #

$\underset{x,y,z\in\#}{\forall}$

ex–meet[x, y] ⇔ $\left(\underset{z\in\#}{\exists}\ (z \le x \wedge z \le y) \bigwedge \underset{\zeta\in\#}{\forall}\ (\zeta \le x \wedge \zeta \le y \Rightarrow \zeta \le z)\right)$

ex–join[x, y] ⇔ $\left(\underset{z\in\#}{\exists}\ (x \le z \wedge y \le z) \bigwedge \underset{\zeta\in\#}{\forall}\ (x \le \zeta \wedge y \le \zeta \Rightarrow z \le \zeta)\right)$

$x \bigsqcap y = \underset{z\in\#}{\ni}\left((z \le x \wedge z \le y) \bigwedge \underset{\zeta\in\#}{\forall}\ (\zeta \le x \wedge \zeta \le y \Rightarrow \zeta \le z)\right)$

$x \bigsqcup y = \underset{z\in\#}{\ni}\left((x \le z \wedge y \le z) \bigwedge \underset{\zeta\in\#}{\forall}\ (x \le \zeta \wedge y \le \zeta \Rightarrow z \le \zeta)\right)$

LatticeOrder ⇌ $\underset{P}{\forall}$

P : ⟨#, ≤⟩

lattice–from–poset[P] : ⟨ex–meet, ex–join⟩

is–lattice–order[P] ⇔ $\underset{x,y\in\#}{\forall}$ (ex–join[x, y] ∧ ex–meet[x, y])

▪Lemma1 ⇌ $\underset{P}{\forall}$

§1–Posets–DualityPrinciple : ⟨is–poset⟩

P : ⟨#, ≤⟩

is–poset[P] ⇒

lattice–from–poset[P] : ⟨ex–meet, ex–join, $\bigsqcap$, $\bigsqcup$⟩

$$\forall_{x,y,z\in\#}$$

ex–meet[x, x] ⇒

x ⨅ x = x

ex–join[x, x] ⇒

x ⨆ x = x

ex–meet[x, y] ⇒

x ⨅ y = y ⨅ x

ex–join[x, y] ⇒

x ⨆ y = y ⨆ x

ex–meet[y, z] ∧ ex–meet[x, y ⨅ z] ∧ ex–meet[x, y] ∧ ex–meet[x ⨅ y, z] ⇒

x ⨅ (y ⨅ z) = (x ⨅ y) ⨅ z

ex–join[y, z] ∧ ex–join[x, y ⨅ z] ∧ ex–join[x, y] ∧ ex–join[x ⨅ y, z] ⇒

x ⨆ (y ⨆ z) = (x ⨆ y) ⨆ z

ex–join[x, y] ∧ ex–meet[x, x ⨆ y] ∧ ex–meet[x, y] ∧ ex–join[x, x ⨅ y] ⇒

x ⨅ (x ⨆ y) = x

x ⨆ (x ⨅ y) = x

x ≤ y ⇔ (x ⨅ y = x)

x ≤ y ⇔ (x ⨆ y = y)

▪Lemma2 ⇌ $\forall_{\text{is–lattice–order[L]}}$

L : ⟨#, ≤⟩

lattice–from–poset[L] : ⟨ ⨅ , ⨆⟩

$$\forall_{x,y,z\in\#}$$

$y \leq z \Rightarrow$

$x \sqcap y \leq x \sqcap z$

$x \sqcup y \leq x \sqcup z$

▪Lemma3 $\rightleftharpoons \underset{\text{is–lattice–order[L]}}{\forall}$

$L : \langle \#,\ \leq,\ \geq \rangle$

lattice–from–poset[L] : $\langle \sqcap,\ \sqcup \rangle$

$\underset{x,y,z \in \#}{\forall}$

$x \sqcap (y \sqcup z) \geq (x \sqcap y) \sqcup (x \sqcap z)$

$x \sqcup (y \sqcap z) \leq (x \sqcup y) \sqcap (x \sqcup z)$

▪Lemma4 $\rightleftharpoons \underset{\text{is–lattice–order[L]}}{\forall}$

$L : \langle \#,\ \leq \rangle$

lattice–from–poset[L] : $\langle \sqcap,\ \sqcup \rangle$

$\underset{x,y,z \in \#}{\forall}$

$x \leq z \Rightarrow$

$x \sqcup (y \sqcap z) \leq (x \sqcup y) \sqcap z$

Semilattice $\rightleftharpoons \underset{L}{\forall} \underset{\sqcap}{\forall}$

$L : \langle \#,\ \sqcap \rangle$

is–semilattice[L, $\sqcap$] $\Leftrightarrow \underset{x,y,z \in \#}{\forall}$

$x \sqcap x = x$

$x \sqcap y = y \sqcap x$

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

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

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

is–lattice[L] $\Leftrightarrow$

is–semilattice[L, $\sqcap$]

is–semilattice[L, $\sqcup$]

$\underset{x,y\in\#}{\forall} ((x \sqcap (x \sqcup y) = x) \wedge (x \sqcup (x \sqcap y) = x))$

▪Corollary $\rightleftharpoons \underset{P}{\forall}$

§1–Posets–DualityPrinciple : ⟨is–poset⟩

is–poset[P] $\Rightarrow$

lattice–from–poset[P] : ⟨#, ex–meet, $\sqcap$⟩

$\underset{x,y\in\#}{\forall}$ ex–meet[x, y] $\Rightarrow$ is–semilattice[P, $\sqcap$]

PosetFromSemilattice $\rightleftharpoons \underset{S}{\forall} \underset{\diamond}{\forall}$

poset–from–semilattice[S, $\diamond$] : $\langle \leq \rangle$

$S : \langle \#, \diamond \rangle$

poset–from–semilattice[S, $\diamond$][#] = #

$\underset{x,y\in\#}{\forall} (x \leq y \Leftrightarrow (x \diamond y = x))$

▪Lemma5 $\rightleftharpoons \underset{\text{is–semilattice[S,}\diamond\text{]}}{\forall}$

§1–Posets–DualityPrinciple : ⟨is–poset⟩

is–poset[poset–from–semilattice[S, $\diamond$]]

lattice–from–poset[poset–from–semilattice[S, $\diamond$]] : $\langle \sqcap \rangle$

$\underset{x,y\in\#}{\forall} (x \diamond y = x \sqcap y)$

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

L : ⟨ $\bigsqcap$ ⟩

poset–from–lattice[L] = poset–from–semilattice[S, $\bigsqcap$]

Theorem4 $\rightleftharpoons \underset{L}{\forall}$

is–lattice[L] ⇒ is–lattice–order[poset–from–lattice[L]]

is–lattice–order[L] ⇒ is–lattice[lattice–from–poset[L]]

§3–SublatticesAndProductsOfLattices :
⟨is–lattice–morphism, is–closure–property, lattice–from–closureproperty⟩ $\rightleftharpoons$

LatticeMorphism $\rightleftharpoons \underset{f}{\forall} \underset{L,M}{\forall}$

is–lattice–morphism[f, L, M] ⇔

$f : \underset{L}{\#} \to \underset{M}{\#}$

$\underset{x,y\in \underset{L}{\#}}{\forall}$

$f\left[x \underset{L}{\bigsqcap} y\right] = f[x] \underset{M}{\bigsqcap} f[y]$

$f\left[x \underset{L}{\bigsqcup} y\right] = f[x] \underset{M}{\bigsqcup} y[y]$

ClosureProperty $\rightleftharpoons \underset{\phi}{\forall} \underset{J}{\forall} \underset{\mathcal{J}}{\forall}$

is–closure–property[ϕ, J, $\mathcal{J}$] ⇔

ϕ[J]

$\underset{\mathcal{K}\subseteq\mathcal{J}}{\forall} \left(\underset{K\in\mathcal{K}}{\forall} \phi[K] \Rightarrow \phi\left[\bigcap \mathcal{K}\right] \right)$

Theorem5 $\rightleftharpoons \underset{L}{\forall}$

CompleteLattices : ⟨is–complete–lattice⟩

L : ⟨#, $\bigsqcap$, $\bigsqcup$, I, O, inf⟩

is–complete–lattice[L] ⇒

$$\forall_{S\subseteq\#}\left(I \in S \bigwedge \forall_{T\subseteq\#} (T \subseteq S \Rightarrow \inf[T] \in S) \Rightarrow \right.$$

$$\left. \text{is–complete–lattice}[\lceil \# \mapsto S,\ \bigsqcap \mapsto \bigsqcap,\ \bigsqcup \mapsto \bigsqcup, I \mapsto I, O \mapsto O\rceil]\right)$$

LatticeFromClosureProperty $\rightleftharpoons \forall_{\phi}\forall_{J}\forall_{\mathcal{J}}$

lattice–from–closureproperty[ϕ, J, $\mathcal{J}$] : ⟨#, $\bigsqcap$, $\bigsqcup$, O, I⟩

= $\mathcal{J}$

O = $\emptyset$

I = J

$$\forall_{K,L}$$

$$K \bigsqcap L = K \bigcap L$$

$$K \bigsqcup L = \bigcap \left\{ M \mid_{M\in\mathcal{J}} M \supseteq K \bigsqcup L \right\}$$

▪Corollary $\rightleftharpoons \forall_{\phi}\forall_{J}\forall_{\mathcal{J}}$

CompleteLattices : ⟨is–complete–lattice⟩

is–closure–property[ϕ, J, $\mathcal{J}$] ⇒ is–complete–lattice[lattice–from–closureproperty[ϕ, J, $\mathcal{J}$]]

PosetProduct $\rightleftharpoons \forall_{P,Q}$

$$P \underset{\text{Posets}}{\times} Q : \langle \#,\ \leq \rangle$$

$$\# = \#_{P} \times \#_{Q}$$

$$\forall_{x,y\in\#}$$

$$x \leq y \Leftrightarrow \left(x_1 \underset{P}{\leq} y_1 \bigwedge x_2 \underset{Q}{\leq} y_2\right)$$

Theorem6 $\rightleftharpoons \forall_{L}\forall_{M}$

Posets : ⟨×⟩

§2–LatticeIdentities : ⟨is–lattice⟩

is–lattice[L] ∧ is–lattice[M] ⇒ is–lattice[L×M]

§4–ModularLattices : ⟨is–modular–lattice, is–distributive–lattice⟩ ⇌

ModularLattice ⇌

§2–LatticeIdentities : ⟨is–lattice, poset–from–lattice⟩

is–modular–lattice[L] ⇔

is–lattice[L]

poset–from–lattice[L] : ⟨ ≤ ⟩

L : ⟨#, ⨆, ⨅⟩

$\forall_{x,y,z\in\#}$

x ≤ z ⇒

x ⨆ (y ⨅ z) = (x ⨆ y) ⨅ z

DistributiveLattice ⇌

is–distributive–lattice[L] ⇔

§2–LatticeIdentities : ⟨is–lattice⟩

is–lattice[L]

L : ⟨ ⨆, ⨅⟩

$\forall_{x,y,z\in\#}$

x ⨅ (y ⨆ z) = (x ⨅ y) ⨆ (x ⨅ z)

x ⨆ (y ⨅ z) = (x ⨆ y) ⨅ (x ⨆ z)

Theorem7 ⇌ $\forall_{G}$

GroupTheory : ⟨is–group, is–normal–subgroup⟩

§3–SublatticesAndProductsOfLattices : ⟨lattice–from–closureproperty⟩

is–group[G] ⇒
is–modular–lattice$\Big[$lattice–from–closureproperty$\Big[\underset{H}{\lambda}$ is–normal–subgroup[G, H], $\underset{G}{\#}$, $\mathcal{P}\Big[\underset{G}{\#}\Big]\Big]\Big]$

Theorem8 ⇌ $\underset{L}{\forall}$

§2–LatticeIdentities : ⟨is–lattice⟩

is–lattice[L] ∧ ¬ is–modular–lattice[L] ⇒

LatticeBasics : ⟨is–isomorphic, is–sublattice⟩

LatticeExamples : ⟨N5⟩

$\underset{N}{\exists}$ (is–sublattice[N, L] ∧ is–isomorphic[N, N5])

Theorem9 ⇌ $\underset{R}{\forall}\ \underset{A}{\forall}$

RingTheory : ⟨is–ring⟩

ModuleTheory : ⟨is–module, is–submodule⟩

§3–SublatticesAndProductsOfLattices : ⟨lattice–from–closureproperty⟩

is–ring[R] ∧ is–module[R, A] ⇒
is–modular–lattice$\Big[$lattice–from–closureproperty$\Big[\underset{B}{\lambda}$ is–submodule[A, B], $\underset{A}{\#}$, $\mathcal{P}\Big[\underset{A}{\#}\Big]\Big]\Big]$

§5–JordanHölderDedekindLattices ⇌

Theorem10 ⇌ $\underset{M}{\forall}$

§4–ModularLattices : ⟨is–modular–lattice⟩

M : ⟨#, $\bigsqcap$, $\bigsqcup$⟩

is–modular–lattice[M] ⇒

$\underset{a,b\in\#}{\forall}$

LatticeBasics : ⟨is–lattice–isomorphism, interval⟩

FunctionBasics : ⟨are–mutually–inverse⟩

is–lattice–isomorphism[ϕ, interval[b, a $\bigsqcup$ b], interval[a $\bigsqcap$ b, a]]

are–mutually–inverse[ϕ, ψ]

↤

$\phi = \underset{x}{\lambda}$ (x $\bigsqcap$ a)

$\psi = \underset{y}{\lambda}$ (y $\bigsqcup$ b)

Corollary10 $\rightleftharpoons \underset{L}{\forall}$

§1–Posets–DualityPrinciple : ⟨cover–relation⟩

§4–ModularLattices : ⟨is–modular–lattice⟩

L : ⟨#, $\bigsqcap$, $\bigsqcup$⟩

is–modular–lattice[L] ⇒

$\underset{a,b,c\in\#}{\forall}$

a ≠ b ⇒

cover–relation[L] : ⟨covers⟩

covers[a, c] ∧ covers[a, b] ⇒ covers[a $\bigsqcup$ b, a] ∧ covers[a $\bigsqcup$ b, a]

covers[c, a] ∧ covers[c, b] ⇒ covers[a, a $\bigsqcap$ b] ∧ covers[b, a $\bigsqcap$ b]

Transposes : ⟨transposes⟩ $\rightleftharpoons \underset{M}{\forall}$

transposes[M] : ⟨are–transposes, are–projective⟩

M : ⟨#, $\bigsqcap$, $\bigsqcup$⟩

$\underset{K,L}{\forall}$

are–transposes[K, L] ⇔

$\exists_{a,b\in\#}$

LatticeBasics : ⟨interval⟩

K = interval[a ⨅ b , a]

L = interval[b , a ⨆ b]

BasicRelations : ⟨transitive–closure⟩

are–projective = transitive–closure[are–transposes]

Lemma5–1 ⇌ $\forall_R \forall_A$

RingTheory : ⟨is–ring⟩

ModuleTheory : ⟨is–module, is–submodule, /, ≈⟩

§3–SublatticesAndProductsOfLattices : ⟨lattice–from–closureproperty⟩

is–ring[R] ∧ is–module[R, A] ⇒

$\mathcal{L}$: ⟨#⟩

$\forall_{S,T\in\#}\ \forall_{U,V\in\#}$

transposes[$\mathcal{L}$] : ⟨are–projective⟩

LatticeBasics : ⟨interval⟩

are–projective[interval[S, T], interval[U, V]] ⇒ T / S ≈ U / V

↤

$\mathcal{L}$ = lattice–from–closureproperty$\left[\lambda_B \text{ is–submodule}[A, B],\ \#_A,\ \mathcal{P}\left[\#_A\right]\right]$

Theorem11 ⇌ $\forall_M$

§4–ModularLattices : ⟨is–modular–lattice⟩

LatticeBasics : ⟨is–lattice–of–finite–length⟩

Transposes : ⟨transposes⟩

ChainsInLattices : ⟨is–connected–chain⟩

is–lattice–of–finite–length[M] ∧ (cover–condition–ξ[M] ∨ alt–cover–condition–ξ[M]) ⇒

$\underset{C,D}{\forall}$

is–connected–chain[M, C] ∧ is–connected–chain[M, D] ∧ ($C_1 = D_1$) ∧ ($C_n = D_m$) ⇒ (n = m)

↤

n = card[C]

m = card[D]

is–lattice–of–finite–length[M] ∧ is–modular–lattice[M] ⇒

$\underset{C,D}{\forall}$

M : ⟨O, I⟩

is–connected–chain[M, C] ∧ is–connected–chain[M, D] ∧
($C_1 = O$) ∧ ($D_1 = O$) ∧ ($C_n = I$) ∧ ($D_m = I$) ⇒

m = n

FunctionBasics : ⟨is–bijection⟩

NaturalNumbers : ⟨number–segment⟩

transposes[M] : ⟨are–projective⟩

$\underset{\pi}{\exists}\left(\text{is–bijection}[\pi, \text{number–segment}[n]] \bigwedge \underset{i=1,\ldots,n}{\forall} \text{are–projective}[C_i, D_{\pi[i]}]\right)$

↤

n = card[C]

m = card[D]

Corollary11 ⇌ $\underset{R}{\forall}\,\underset{A}{\forall}$

RingTheory : ⟨is–ring⟩

ModuleTheory : ⟨is–module, is–submodule, /, ≈⟩

§3–SublatticesAndProductsOfLattices : ⟨lattice–from–closureproperty⟩

§4–ModularLattices : ⟨is–modular–lattice⟩

LatticeBasics : ⟨is–lattice–of–finite–length⟩

$$\text{is–ring[R]} \wedge \text{is–module[R, A]} \wedge \text{is–lattice–of–finite–length}[\mathcal{M}] \wedge \text{is–modular–lattice}[\mathcal{M}] \Rightarrow$$

$$\forall_{C,D}$$

$\mathcal{M}$: ⟨O, I⟩

$$\text{is–connected–chain}[\mathcal{M}, C] \wedge \text{is–connected–chain}[\mathcal{M}, D] \wedge (C_1 = O) \wedge (D_1 = O) \wedge (C_n = I) \wedge (D_m = I) \Rightarrow$$

m = n

FunctionBasics : ⟨is–bijection⟩

NaturalNumbers : ⟨number–segment⟩

$$\exists_{\pi} \left(\text{is–bijection}[\pi, \text{number–segment[n]}] \bigwedge_{i=2,\ldots,n}^{\forall} C_i / C_{i-1} \approx D_{\pi[i]} / D_{\pi[i]-1} \right)$$

↤

$$\mathcal{M} = \text{lattice–from–closureproperty}\left[\lambda_B \text{is–submodule[A, B]}, \#_A, \mathcal{P}\left[\#_A\right]\right]$$

§6–DistributiveLattices : ⟨function–lattice⟩ ⇌

Theorem12 ⇌ $\forall_L$

L : ⟨#, ⨅, ⨆⟩

§2–LatticeIdentities : ⟨is–lattice⟩

is–lattice[L] ⇒

$$\forall_{x,y,z\in\#} (x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)) \Leftrightarrow \forall_{x,y,z\in\#} (x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z))$$

Lemma12 ⇌ $\forall_C$

§1–Posets–DualityPrinciple : ⟨is–chain⟩

§2–LatticeIdentities : ⟨is–lattice–order⟩

is–chain[C] ⇒ is–lattice–order[C]

FunctionLattice ⇌ $\underset{S}{\forall}\ \underset{D}{\forall}$

function–lattice[S, D] : ⟨#, ≤⟩

$\# = \left\{ f \mid_{f} f : S \to D \right\}$

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

§2–LatticeIdentities : ⟨poset–from–lattice⟩

$f \leq g \Leftrightarrow \text{where}\left[\sqsubseteq \ = \text{poset–from–lattice}[D][\leq], \underset{x \in S}{\forall} f[x] \sqsubseteq g[x] \right]$

Theorem13 ⇌ $\underset{S}{\forall}\ \underset{D}{\forall}$

§4–ModularLattices : ⟨ is–distributive–lattice⟩

is–distributive–lattice[D] ⇒ is–distributive–lattice[function–lattice[S, D]]

Theorem14 ⇌ $\underset{L}{\forall}$

§4–ModularLattices : ⟨ is–distributive–lattice⟩

L : ⟨#, $\bigsqcap$, $\bigsqcup$⟩

is–distributive–lattice[L] ⇒

$\underset{c,x,y \in \#}{\forall}$

$(c \bigsqcap x = c \bigsqcap y) \wedge (c \bigsqcup x = c \bigsqcup y) \Rightarrow (x = y)$

§7–RingsOfSets : ⟨is–ring–of–sets, is–field–of–sets, special–elements⟩ ⇌

SetFamilies ⇌ $\underset{\Phi}{\forall}\ \underset{I}{\forall}$

is–ring–of–sets[Φ, I] ⇔

$\Phi \subseteq \mathcal{P}[I]$

$\forall_{S,T\in\Phi}$

$S \bigcap T \in \Phi$

$S \bigcup T \in \Phi$

is–field–of–sets[Φ, I] ⇔

is–ring–of–sets[Φ, I]

$\forall_{S\in\Phi}\ I \setminus S \in \Phi$

▪Lemma1 ⇌ $\forall_{P}$

§1–Posets–DualityPrinciple : ⟨is–poset⟩

P : ⟨#, ≤⟩

$\text{is–poset[P]} \Rightarrow \text{is–ring–of–sets}\left[\left\{A \;\middle|\; \underset{A\subseteq\#}{\forall} \underset{a\in A}{\forall} \underset{x\in\#}{\forall} (x \le a \Rightarrow x \in A)\right\}, \#\right]$

•Lemma2 ⇌ $\forall_{P}$

§1–Posets–DualityPrinciple : ⟨is–poset⟩

§4–ModularLattices : ⟨is–distributive–lattice ⟩

LatticeBasics : ⟨length⟩

NaturalNumbers : ⟨is–finite⟩

P : ⟨#, ≤⟩

is–poset[P] ∧ is–finite[P] ⇒

is–ring–of–sets[$\mathcal{C}$, #]

is–distributive–lattice[$\mathcal{L}$]

length[$\mathcal{L}$] = card[#]

↤

$$\mathcal{C} = \left\{ A \;\middle|\; \underset{A \subseteq \#}{} \; \underset{a \in A}{\forall} \; \underset{x \in \#}{\forall} \; (x \leq a \Rightarrow x \in A) \right\}$$

$$\mathcal{L} = \left[\# \mapsto \mathcal{C},\ \sqcap \mapsto \bigcap,\ \sqcup \mapsto \bigcup \right]$$

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

special–elements[L] : ⟨is–join–irreducible⟩

L : ⟨#, O, $\sqcup$⟩

$\underset{a}{\forall}$

is–join–irreducible[a] ⇔

$a \in \#$

$a \neq O$

$\underset{b,c \in \#}{\forall} ((b \sqcup c = a) \Rightarrow (b = a) \vee (c = a))$

•Lemma3 $\rightleftharpoons \underset{L}{\forall}$

§2–LatticeIdentities : ⟨poset–from–lattice⟩

LatticeBasics : ⟨is–bounded–distributive–lattice, derived–operations⟩

is–bounded–distributive–lattice[L] ⇒

poset–from–lattice[L] : ⟨ ≤ ⟩

special–elements[L] : ⟨is–join–irreducible⟩

derived–operations[L] : ⟨join⟩

L : ⟨#⟩

$\underset{p}{\forall} \underset{\overline{x}}{\forall}$

is–join–irreducible[p] $\bigwedge \underset{i=1,\ldots k}{\forall} \langle \overline{x} \rangle_i \in \# \bigwedge p \leq \text{join}[\overline{x}] \Rightarrow \underset{i=1,\ldots k}{\exists} p \leq \langle \overline{x} \rangle_i$

↤

$k = |\langle \overline{x} \rangle|$

•Corollary3 $\rightleftharpoons \underset{L}{\forall}$

§4–ModularLattices : ⟨ is–distributive–lattice⟩

LatticeBasics : ⟨is–lattice–of–finite–length, derived–operations⟩

L : ⟨#⟩

special–elements[L] : ⟨is–join–irreducible⟩

derived–operations[L] : ⟨join–of–set⟩

is–distributive–lattice[L] ∧ is–lattice–of–finite–length[L] ⇒

$\underset{a\in\#}{\forall} \underset{J}{\exists}$

$\underset{x\in J}{\forall}$ is–join–irreducible[x]

a = join–of–set[J]

$\underset{K\subset J}{\forall}$ a ≠ join–of–set[K]

▪Lemma4 $\rightleftharpoons \underset{L}{\forall}$

§4–ModularLattices : ⟨ is–distributive–lattice⟩

LatticeBasics : ⟨length, is–lattice–of–finite–length⟩

L : ⟨#⟩

special–elements[L] : ⟨is–join–irreducible⟩

is–distributive–lattice[L] ∧ is–lattice–of–finite–length[L] ⇒

$\left(\text{card}\left[\left\{x \mid \underset{x\in\#}{} \text{is–join–irreducible}[x]\right\}\right] = \text{length}[L]\right)$

Theorem15 $\rightleftharpoons \underset{L}{\forall}$

§2–LatticeIdentities : ⟨poset–from–lattice⟩

§4–ModularLattices : ⟨ is–distributive–lattice⟩

LatticeBasics : ⟨length, is–lattice–of–finite–length, ≈⟩

L : ⟨#⟩

poset–from–lattice[L] : ⟨ ≤ ⟩

is–distributive–lattice[L] ∧ is–lattice–of–finite–length[L] ⇒

$$L \approx \left[\# \mapsto \left\{ A \;\middle|\; A \subseteq \# \right\}, \bigsqcap \mapsto \bigcap, \bigsqcup \mapsto \bigcup \right]$$

Corollary15 ⇌ $\underset{n \in \mathbb{N}}{\forall}$

§1–Posets–DualityPrinciple : ⟨is–poset⟩

§4–ModularLattices : ⟨is–distributive–lattice⟩

NaturalNumbers : ⟨number–segment⟩

$$\text{card}\left[\left\{L \;\middle|\; \underset{L}{\text{is–distributive–lattice}[L]} \bigwedge \underset{L}{\#} = \text{number–segment}[n]\right\}\right] =$$

$$\text{card}\left[\left\{P \;\middle|\; \underset{P}{\text{is–poset}[P]} \bigwedge \underset{P}{\#} = \text{number–segment}[n]\right\}\right]$$

§8–BooleanLattices–BooleanAlgebras : ⟨is–boolean–algebra, is–boolean–morphism⟩ ⇌

Theorem16 ⇌ $\underset{L}{\forall}$

LatticeBasics : ⟨is–boolean–lattice, has–complement⟩

is–boolean–lattice[L] ⇒

L : ⟨#, ⨅, ⨆, O, I⟩

$$\underset{x \in \#}{\forall} \; \underset{y \in \#}{\exists_1} \; \text{has–complement}[x, y, L]$$

$$\underset{x,y \in \#}{\forall} \; \underset{r,s \in \#}{\forall}$$

has–complement[x, r, L] ⇒

x ⨅ r = O

x ⨆ r = I

$$\forall_{z\in\#} \text{ (has–complement[r, z, L]} \Rightarrow (x = z))$$

has–complement[x, r, L] $\wedge$ has–complement[y, s, L] $\Rightarrow$
has–complement[x $\sqcap$ y, r $\sqcup$ s, L] $\wedge$ has–complement[x $\sqcup$ y, r $\sqcap$ s, L]

BooleanAlgebra $\rightleftharpoons \forall_{A}$

is–boolean–algebra[A] $\Leftrightarrow$

$A : \langle \#, \sqcap, \sqcup, O, I, ' \rangle$

§4–ModularLattices : $\langle$ is–distributive–lattice$\rangle$

is–distributive–lattice[A]

$\forall_{x,y\in\#}$

$x \sqcap x' = O$

$x \sqcup x' = I$

$(x')' = x$

$(x \sqcup y)' = x' \sqcap y'$

$(x \sqcap y)' = x' \sqcup y'$

BooleanMorphism $\rightleftharpoons \forall_{f} \forall_{A,B}$

is–boolean–morphism[f, A, B] $\Leftrightarrow$

is–lattice–morphism[f, A, B]

$A : \langle \# \rangle$

$\forall_{x\in\#} \left(f\left[{}^{\text{TM}}\text{Derivative1}_{A}[x] \right] = {}^{\text{TM}}\text{Derivative1}_{B}[f[x]] \right)$

Proposition17 $\rightleftharpoons \forall_{A,B}$

is–boolean–algebra[A] $\wedge$ is–boolean–algebra[B] $\Rightarrow$

$\forall_{f}$

§3–SublatticesAndProductsOfLattices : ⟨is–lattice–morphism⟩

$$f : A \to B \bigwedge \left(f\left[\underset{A}{O}\right] = \underset{B}{O}\right) \bigwedge \left(f\left[\underset{A}{I}\right] = \underset{B}{I}\right) \bigwedge \text{is–lattice–morphism}[f, A, B] \Rightarrow$$
$$\text{is–boolean–morphism}[f, A, B]$$

$$\text{Theorem18} \rightleftharpoons \underset{L}{\forall}$$

§4–ModularLattices : ⟨ is–distributive–lattice⟩

LatticeBasics : ⟨is–sublattice, is–bounded, is–complemented⟩

L : ⟨#⟩

$$\text{is–distributive–lattice}[L] \wedge \text{is–bounded}[L] \Rightarrow \text{is–sublattice}\left[\left\{x \;\middle|\; \underset{x \in \#}{} \text{is–complemented}[x, L]\right\}, L\right]$$

$$\text{Theorem19} \rightleftharpoons \underset{B}{\forall}$$

LatticeBasics : ⟨is–boolean–lattice, is–lattice–of–finite–length, length, ≈⟩

NaturalNumbers : ⟨number–segment⟩

$$\text{is–boolean–lattice}[B] \wedge \text{is–lattice–of–finite–length}[B] \Rightarrow$$
$$B \approx \left[\# \mapsto \mathcal{P}[\text{number–segment}[\text{length}[B]]],\ \sqcap \mapsto \bigcap,\ \sqcup \mapsto \bigcup\right]$$

$$\text{Corollary19} \rightleftharpoons \underset{B}{\forall}$$

LatticeBasics : ⟨is–boolean–lattice, is–lattice–of–finite–length, length, ≈ , Un⟩

NaturalNumbers : ⟨number–segment⟩

$$\text{is–boolean–lattice}[B] \wedge \text{is–lattice–of–finite–length}[B] \Rightarrow$$

§1–Posets–DualityPrinciple : ⟨is–chain⟩

Posets : ⟨×⟩

$$\underset{C}{\exists}$$

$$\text{is–chain}[C] \wedge (\text{card}[C] = 2)$$

$$B \approx \underset{i=1,\ldots,n}{\Omega^{\times}_{\text{Un}}} C$$

$\nleftarrow$

n = length[B]

§9–FreeBooleanAlgebras $\rightleftharpoons$

Theorem20 $\rightleftharpoons \underset{A}{\forall}\ \underset{\overline{x}}{\forall}$

§8–BooleanLattices–BooleanAlgebras : ⟨is–boolean–algebra, is–boolean–morphism⟩

BooleanAlgebras : ⟨generated–boolean–algebra, img⟩

NaturalNumbers : ⟨is–finite⟩

SetOperations : ⟨™Superscript⟩

A : ⟨#, $\bigsqcap$, $\bigsqcup$, O, I, '⟩

is–boolean–algebra[A] $\wedge$ is–finite[#] $\wedge$ ({$\overline{x}$} = #) $\wedge$ (|⟨$\overline{x}$⟩| = n) $\Rightarrow$

is–boolean–morphism[h, $\mathcal{B}$, A]

img[h, $\mathcal{B}$, A] = generated–boolean–algebra[{$\overline{x}$}]

$\nleftarrow$

n = card[#]

$\mathcal{B} = \left[\# \mapsto \mathcal{P}[\{0, 1\}^n],\ \bigsqcap \mapsto \bigcap,\ \bigsqcup \mapsto \bigcup\right]$

$$\text{pow} = \underset{a,j}{\lambda} \begin{cases} a & \Leftarrow\ j = 0 \\ a' & \Leftarrow\ j = 1 \end{cases}$$

$$h = \underset{S}{\lambda}\ \Omega_O^{\bigsqcup}{}_{e \in S}\ \Omega_I^{\bigsqcap}{}_{i=1,\ldots,n}\ \text{pow}[\langle \overline{x} \rangle_i, e_i]$$

Corollary20 $\rightleftharpoons \underset{n \in \mathbb{N}}{\forall}$

BooleanAlgebras : ⟨generated–boolean–algebra⟩

NaturalNumbers : ⟨number–segment, $\leq$, ™Superscript⟩

card[generated–boolean–algebra[number–segment[n]][#]] $\leq 2^{2^n}$

$$\text{Theorem21} \rightleftharpoons \underset{n\in\mathbb{N}}{\forall}\ \underset{B}{\exists}\ \underset{\overline{S}}{\exists}$$

§8–BooleanLattices–BooleanAlgebras : ⟨is–boolean–algebra, is–boolean–morphism⟩

$$\text{is–boolean–algebra}[B] \bigwedge (|\langle\overline{S}\rangle| = n) \bigwedge \underset{i=1,\ldots,n}{\forall} \langle\overline{S}\rangle_i \in \underset{B}{\#}$$

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

$$\text{is–boolean–algebra}[A] \bigwedge (|\langle\overline{x}\rangle| = n) \bigwedge \underset{i=1,\ldots,n}{\forall} \langle\overline{x}\rangle_i \in \underset{A}{\#} \Rightarrow$$

$$\underset{h}{\exists!} \left(\text{is–boolean–morphism}[h, B, A] \bigwedge \underset{i=1,\ldots,n}{\forall} h[\langle\overline{S}\rangle_i] = \langle\overline{x}\rangle_i\right)$$

Curriculum Vitae

Affiliation

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, A–4040 Linz, Austria
Tel. ++43 732 2468 9926
Fax ++43 732 2468 9930
Camelia.Rosenkranz@risc.uni-linz.ac.at

Personal Information

Name	Camelia Rosenkranz
Date and Place of Birth:	6 October 1979 in Cluj–Napoca / Romania
Nationality:	Romanian

Education

Oct 2003–Feb 2009	PhD Student, RISC, Johannes Kepler University, Austria.
Oct 2002–Jul 2003	MsC in Intelligent Systems, Computer Science, Babes-Bolyai University, Cluj-Napoca, Romania
Feb 2003–Jul 2003	Erasmus Exchange Student at Johannes Kepler University
Jun 2003	Final exam in Computer Science
Oct 1998–Jul 2002	BsC in Computer Science, Babes-Bolyai University, Cluj-Napoca, Romania
Jun 2002	Final exam in Computer Science
Jun 1998	Matura
Sep 1994–Jun 1998	Computer Science High School “Tiberiu Popoviciu”, Cluj–Napoca, Romania

Non–academic Professional Experience

Jun 2001–Oct 2001	“Centrul de Orientare Profesionala”, Babes-Bolyai University (Web programming and web design)
Oct 2002–Feb 2003	Voluntary work at Communication Center, Babes-Bolyai University, Cluj–Napoca, Romania.
Sep 2002–Jan 2003	Internship at InfoWorld (Java programming)

Teaching Experience

Oct 2000–Aug 2002	Teaching Instructor for Computer Science, Babes-Bolyai University, Laboratories for Advanced Programming Methods, Computer Architecture, Algorithms and Programming and Operating Systems
Oct 2002–Feb 2003	Teaching Assistant for Computer Science, Babes-Bolyai University, Laboratories of Computer Architecture and Seminars of Mathematical Foundations of Computer Science

Service

2008	Workshop organization, ApCoA 2008, Member of the local organization team
2007	Summer School Organization, CoCoA 2007, Member of the local organization team
2006	Webpage development: GBImplementations
2004	Conference organisation AISC 2004, Member of the publicity team

Acknowledgments

As already noted in the Introduction, Bruno Buchberger provided seminal ideas for this thesis. But not only that—he created RISC and the *Theorema* group, where I found a lively environment for my research. Thanks also go to my advisor Tudor Jebelean for all his support and for our long and interesting discussions.

I would like to thank all my (past and present) colleagues from the *Theorema* group: Adrian Craciun, Alexander Zapletal, Besik Dundua, Florina Piroi, Gabor Kusper, Georg Regensburger, Judit Robu, Koji Nakagawa, Laura Kovacs, Loredana Tec, Martin Giese, Madalina Erascu, Madalina Hodorog, Mircea Marin, Nikolaj Popov, Robert Vajda, Temur Kutsia, and Wolfgang Windsteiger. It was a pleasure to work with you. Special thanks go to Florina Piroi: It was a beautiful collaboration, and I learned a lot in the process.

Among the many persons that influenced my perspective on mathematics, I want to thank Heinrich Rolletschek, who gave me deep insights into set theory and logic. Thank you for the many interesting discussions about mathematical foundations. To my professors from the Babes-Bolyai University in Cluj, especially to Doina Tatar and Florian Boian—thank you for guiding my undergraduate studies.

Many thanks also to my friends from RISC and elsewhere, especially to Nicoleta, Karoly and Anett, Corina, Dana, and Sylvia. Last but not least I want to thank Markus for all his support, patience, and love.

The work presented in this thesis was supported by the European project CALCULEMUS, by the FWF project SFB F1302, by the INTAS project 1000008-8144, by AEMTIA (with the Institute e-Austria Timisoara) funded through the Austrian Ministery of Science and Research, and by the Upper Austrian grant for RISC PhD studies.

References

[AbramowitzStegun65] M. Abramowitz and I.A. Stegun, *Handbook of Mathematical Functions (with Formulas, Graphs, and Mathematical Tables)*, Dover, 1965.

[ActiveMath] *ActiveMath*, http://www.activemath.org, last access February 2009.

[AhoEtAl75] A. A. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*,Addison–Wesley, October 1975.

[Anghelache04] R. Anghelache, Hermes—A Reliable Conversion from TeX to MathML. In: H. Becker, K. Stange and B. Wegner (eds.), *Proceedings of New Developments in Electronic Publishing of Mathematics* (4-th European Congress of Mathematics), 2004.

[AspertiEtAl07] A. Asperti, C. Sacerdoti Coen, E. Tassi and S. Zacchiroli, User Interaction with the Matita Proof Assistant. In: D. Aspinall and C. Lüth (eds.), *Journal of Automated Reasoning, Special Issue on User Interfaces for Theorem Proving*, vol. 39, no. 2, pp. 109–139, 2007.

[AspertiEtAl07a] A. Asperti, C. Sacerdoti Coen, E. Tassi and S. Zacchiroli. Crafting a Proof Assistant. In: T. Altenkirch and C. McBride (eds.), *Proceedings of TYPES 2006: Types for Proofs and Programs*, LNCS 4502, pp. 18–32, Springer, 2007.

[AspertiEtAl06] A. Asperti, F. Guidi, C. Sacerdoti Coen, E. Tassi, and S. Zacchiroli, A Content Based Mathematical Search Engine: Whelp. In: C. Paulin-Mohring, and B. Werner (eds.), *Proceedings of Types for Proofs and Programs International Workshop* (TYPES 2004), LNCS 3839, pages 17–32, Springer Verlag, 2006.

[AspertiEtAl00] A. Asperti, L. Padovani, C. Sacerdoti Coen and I. Schena. *An Hypertextual Electronic Library of Mathematics.* Talk at INRIA, Rocquencourt, June 5, 2000.

[AspertiEtAl00a] A. Asperti, L. Padovani, C. Sacerdoti Coen and I. Schena. *Content–centric Logical Environments*. Short presentation at LICS'2000, Santa Barbara, California, USA, June 2000.

[AspertiSelmi04] A. Asperti and M. Selmi, Efficient Retrieval of Mathematical Statements. In: A. Asperti, G. Bancerek and A. Trybulec (eds.), *Proceedings of MKM04*, Bialowieza, Poland, LNCS 3119, pages 17–32, Springer Verlag.

[AspertiTassi07] A. Asperti and E. Tassi, Higher–Order Proof Reconstruction from Paramodulation–based Refutations: The Unit Equality Case. In: M. Kauers, M. Kerber, R. Miner and W. Windsteiger (eds.), *Towards Mechanized Mathematical Assistants* (Proceedings of MKM and Calculemus 2007), LNCS 4573, pp. 146–160, Springer Verlag.

[AspertiZacchiroli04] A. Asperti and S. Zacchiroli, Searching Mathematics on the Web: State of the Art and Future Developments, In: H. Becker, K. Stange and B. Wegner (eds.), *New Developments in Electronic Publishing*, 2004.

[AutexierEtAl08] S. Autexier, J. Campbell, J. Rubio, V. Sorge, M. Suzuki and F. Wiedijk (eds.), *Intelligent Computer Mathematics*, Proceedings of the 9th International Conference, AISC 2008 and 15th Symposium, Calculemus 2008 and 7th International Conference, MKM 2008, Birmingham, LNCS 5144, Springer, 2008.

[BancerekKohlhase07] G. Bancerek and M. Kohlhase, *From Insight to Proof. Festschrift in honour of A. Trybulec.* Chapter Towards a Mizar Mathematical Library in OMDoc Format, pp. 265–277, University of Bialystok, vol. 10, no. 23, 2007.

[BancerekRudnicki03] G. Bancerek and P. Rudnicki, Information Retrieval in MML, In: A. Asperti, B. Buchberger and J. Davenport (eds.), *Proceedings of the Second International Conference on Mathematical Knowledge Management*, Springer, LNCS 2594, pp. 119–132, 2003 .

[Baraka06] R. Baraka, *A Framework for Publishing and Discovering Mathematical Web Services*, PhD Thesis, Research Institute for Symbolic Computation, Linz, Austria, August 2006.

[Barendregt97] H. Barendregt, The impact of the lambda calculus, *Bulletin of Symbolic Logic*, vol. 3, no. 2, pp. 181–215, 1997.

[BertotCasteran04] Y. Bertot and P. Castèran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*, Springer, 2004.

[Bruijn87] N.G. de Bruijn, The mathematical vernacular, a language for mathematics with typed sets. In: P. Dybjer et al. (eds.), *Proceedings of the Workshop on Programming Languages*, 1987, Marstrand, Sweden

[Buchberger08] B. Buchberger, Groebner Bases in Theorema Using Functors, In: J.C. Faugere and D. Wang (eds.), *Proceedings of SCC'08*, pp. 1–15, LMIB Beihang University Press, 2008.

[Buchberger04] B. Buchberger, Algorithm Supported Mathematical Theory Exploration: A Personal View and Strategy. In: B. Buchberger and J. Campbell (eds.),
Proceedings of AISC2004 (7th International Conference on Artificial Intelligence and Symbolic Computation), LNAI 3249, pp. 236–250, Springer, Berlin–Heidelberg, 2004.

[Buchberger04a] B. Buchberger, *Towards the Automated Synthesis of a Groebner Bases Algorithm*, RACSAM—Revista de la Real Academia de Ciencias (Review of the Spanish Royal Academy of Science), Serie A: Mathematicas vol. 98, no. 1, pp. 65–75, 2004.

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

[Buchberger03a] B. Buchberger, Algorithm Synthesis by Lazy Thinking: Examples and Implementation in Theorema. In: A. Asperti et al. (eds.), *Proceedings of the Mathematical Knowledge Management Workshop 2003*, Electronic Notes in Theoretical Computer Science, vol. 93, pp. 24-59, 2003.

[Buchberger02] B. Buchberger, *Theorema: A Formal Frame for Mathematics*. Invited talk at EACA–2002, Octavo Encuentro de Algebra Computational y Applicaciones, Penaranda de Duero, Universidad de Vallodolid, pp. 11–32, 2002.

[Buchberger01] B. Buchberger, Mathematical Knowledge Management in Theorema, In: B. Buchberger and O. Caprotti (eds.), *First International Workshop on Mathematical Knowledge Management (MKM 2001)*, September 2001.

[Buchberger01a] B. Buchberger, Groebner Rings and Modules. In: S. Maruster, B. Buchberger, V. Negru and T.Jebelean (eds.), *Proceedings of SYNASC 2001*, pp. 22–25, Timisoara, Romania, 2001.

[Buchberger01b] B. Buchberger, The PCS Prover in Theorema. In: R. Moreno-Diaz, B. Buchberger and J.L. Freire (eds.), *Proceedings of EUROCAST 2001* (8th International Conference on Computer Aided Systems Theory—Formal Methods and Tools for Computer Science), LNCS 2178 , pp. 19–23, Springer, 2001.

[Buchberger00] B. Buchberger, Theory Exploration with Theorema, In T. Jebelean, V. Negru and A. Popovici (eds.), *Selected Papers of the 2nd International Workshop on Symbolic and Numeric Algorithms in Scientific Computing*, Analele Universitatii din Timisoara, Seria Matematica–Informatica, Timisoara, Romania, October 2000.

[Buchberger99] B. Buchberger, Theorem Proving Versus Theory Exploration. In: A. Armando and T. Jebelean (eds.), *Proceedings of the Calculemus'99 Workshop*, vol. 23, no. 3 of Electronic Notes in Theoretical Computer Science, Elsevier, 1999.

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

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

[Buchberger93] B. Buchberger, *Mathematica: Doing Mathematics by Computer?*, Invited Talk at DISCO'93, Gmunden, Austria, September 1993. Available as RISC Report 93-50, University of Linz, Austria, 1993.

[BuchbergerCraciun03] B. Buchberger and A. Craciun. Algorithm Synthesis by Lazy Thinking: Examples and Implementation in Theorema. In: F. Kamareddine (ed.), *Proceedings of the Mathematical Knowledge Management Workshop 2003*, Edinburgh, November, 2003, Electronic Notes in Theoretical Computer Science, vol. 93, pp. 24–59, February 2004.

[BuchbergerEtAl06] B. Buchberger, A. Craciun, T. Jebelean, L. Kovacs, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, Judit Robu, M. Rosenkranz and W. Windsteiger, Theorema: Towards Computer-Aided Mathematical Theory Exploration. In: *Journal of Applied Logic*, vol. 4, no. 4, pp. 470–504, 2006.

[BuchbergerEtAl03] B. Buchberger, G. Gonnet and M. Hazewinkel (Eds.), *Mathematical Knowledge Management*, Special Issue of Annals of Mathematics and Artificial Intelligence, volume 38, Kluwer Academic Publisher, Dordrecht, Netherlands, May 2003.

[BuchbergerEtAl00] B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru and W. Windsteiger, The Theorema Project: A Progress Report, In: M.Kerber and M.Kohlhase (eds.), *Symbolic Computation and Automated Reasoning*, Proceedings of the Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus'00), pp. 98–113, A.K. Peters, 2000.

[BuchbergerEtAl97] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta and D. Vasaru, A Survey of the Theorema project. In: W. Kuechlin (ed.), *Proceedings of ISSAC 1997* (International Symposium on Symbolic and Algebraic Computation), pp. 384–391, ACM Press, 1997.

[BundyEtAl98] A. Bundy, S. Colton and T. Walsh, HR – A System for Machine Discovery in Finite Algebras, *Proceedings of the Machine Discovery Workshop ECAI 1998*, Brighton, England, 1998.

[BuswellEtAl03] S. Buswell, O. Caprotti, and M. Dewar, *Mathematical Service Description Language: Final Version*, Monet Deliverables, March 2003.

[CaprottiEtAl04] O. Caprotti, J.H. Davenport, M. Dewar and J. Padget, Mathematics on the (Semantic) NET, In: C. Bussler, J. Davies, D. Fensel and R. Studer (eds.), *Proceedings of ESWS 2004, First European Semantic Web Symposium*, LNCS 3053, pp. 213–224, 2004.

[Calculemus] *The CALCULEMUS Project*, http://www.eurice.de/calculemus, last access February 2009.

[CarlsonEtAl99] A.J. Carlson, C.M. Cumby, J.L. Rosen and D. Roth, *SNoW User's Guide*. UIUC Tech Report UIUC-DCS-R-99-210.

[ChangLee73] C.L. Chang and R.C. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.

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

[Colton02] S. Colton, Making Conjectures about Maple Functions. In *Proceedings of AISC/Calculemus'02*, LNAI 2385, Springer, 2002.

[ColtonEtAl06] S. Colton, P. Torres, P. Cairns and V. Sorge, Managing Automatically Formed Mathematical Theories, In *Proceedings of the 5th International Conference on Mathematical Knowledge Management*, LNAI volume 4108, Springer 2006.

[ColtonEtAl02] S. Colton, R. McCasland, A. Bundy and T. Walsh, Automated Theory Formation for Tutoring Tasks in Pure Mathematics. In: *Proceedings of the CADE'02 Workshop on the Role of Automated Deduction in Mathematics*, Copenhagen, Denmark, 2002.

[ColtonEtAl00] S. Colton, A. Bundy and T. Walsh. On the Notion of Interestingness in Automated Mathematical Discovery, In: *IJHCS: International Journal of Human-Computer Studies*, Academic Press, 2000.

[CONNEXIONS] *Connexions: Rhaptos Software Development*, http://www.rhaptos.org, last access February 2009.

[CoQ] *The Coq proof assistant*, http://coq.inria.fr, last acess February 2009.

[Craciun08] A. Craciun, *Lazy Thinking Algorithm Synthesis in Gröbner Bases Theory*, PhD Thesis, Research Institute for Symbolic Computation (RISC), Linz, Austria, 2008.

[CraciunHodorog07] A. Craciun, M. Hodorog. Decompositions of Natural Numbers: From A Case Study in Mathematical Theory Exploration, In: Dana Petcu, V. Negru, D. Zaharie and T. Jebelean (eds.), *Proceedings of the 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC07)*, pp. 1–8, West University of Timisoara, Romania, September 2007.

[CruzFilipe04] L. Cruz-Filipe, *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*, PhD Thesis. University of Nijmegen, April 2004.

[DLMF] *NIST Digital Library of Mathematical Functions*, http://dlmf.nist.gov/, last access February 2009.

[EMIS] *The European Mathematical Information Service*, http://www.emis.de/, last access February 2009.

[Farmer04] W.M. Farmer, MKM: A New Interdisciplinary Field of Research, *SIGSAM Bulletin*, vol. 38 no. 2, pp. 47–52, June 2004.

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

[GAMS] *NIST Guide to Available Mathematical Software*, http://gams.nist.gov/, last access February 2009.

[GieseBuchberger07] M. Giese, B. Buchberger. Towards Practical Reflection for Formal Mathematics, extended abstract. In: T. Kutsia and M. Marin (eds.), *Proceedings of Austria-Japan Workshop on Symbolic Computation and Software Verification*, pp. 30–34, RISC Technical report no. 07–09, 2007.

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

[GimenezCasteran06] E. Giménez and P. Castéran, *A Tutorial on Recursive Types in CoQ*, Revision July 2006. Available online at: http://www.labri.fr/perso/casteran/RecTutorial.pdf, last access February 2009.

[Graf96] P. Graf, *Term indexing*, LNCS 1053, Springer Verlag, 1996.

[Graf94] P. Graf, *Substitution Tree Indexing*, 1994MPI–I–94–251, Saarbruecken. Available online at: http://citeseer.ist.psu.edu/graf94substitution.html, last access February 2009.

[GoguadzeEtAl02] G. Goguadze, E. Melis, A. Asperti, MOWGLI Report D1.b. *Structure and Metastructure of Mathematical Documents*, 2002. Available online at: http://mowgli.cs.unibo.it/html_yes_frames/deliverables/requirement-analysis/d1b.html, last access February 2009.

[Gonthier08] G. Gonthier, Formal Proof—The Four Color Theorem, *Notices of the AMS*, vol. 55, no. 11, pp. 1382–1392, December 2008.

[GuidiCoen03] F. Guidi, and C. Sacerdoti Coen, Querying Distributed Digital Libraries of Mathematics, In: T. Hardin and R. Rioboo (eds.), *Proceedings of Calculemus 2003*, pp. 17–30, Aracne Editrice S.R.L, Italy, 2003.

[Harrison06] J. 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] K. Aboul-Hosn. *A Proof-Theoretic Approach to Mathematical Knowledge Management*, Ph.D. Thesis, Cornell University, January 2007.

[HosnAndersen05] K. Aboul-Hosn and T. Damhøj Andersen, A Proof-Theoretic Approach to Hierarchical Math Library Organization, In: M. Kohlhase (ed.), *Proceedings of the 4th International Mathematical Knowledge Management Conference*, pp. 1–16, Bremen, October 2005.

[Isabelle] *Logics of Isabelle*, http://www.cl.cam.ac.uk/research/hvg/Isabelle/logics.html, last access February 2009.

[Jackson94] P. B. Jackson. *The Nuprl Proof Development System, Version 4.1* Introductory Tutorial. Unpublished manuscript, Cornell University, 1994. Available online at http://www.cs.cornell.edu/Info/Projects/NuPrl/tutorial/tutorial.ps, last access November 2007.

[JSTOR] *The Scholarly Journal Archive*, http://www.jstor.org/, last access February 2009.

[Kohlhase06] M. Kohlhase, *OMDoc. An Open Markup Format for Mathematical Documents (Version 1.2)*, LNAI 4180, Springer, Heidelberg, 2006.
Updated version available online at http://www.omdoc.org/pubs/omdoc1.2.pdf.

[KohlhaseFranke01] M. Kohlhase and A. Franke, MBase: Representing Knowledge and Content for the Integration of Mathematical Software Systems, In: *Journal of Symbolic Computation*, vol. 32 no. 4, pages 365–402, 2001.

[KohlhaseFranke00] M. Kohlhase and A. Franke. System Description: MBase, an Open Mathematical Knowledge Base. In: D. McAllester (ed.), *CADE-17 Proceedings*, Springer, LNAI 1831, p.455–459, 2000.

[KohlhaseSucan07] M. Kohlhase and I. Sucan, *System Description: MathWebSearch 0.3: A Semantic Search Engine*, preprint, 2007.

[KohlhaseSucan06] M. Kohlhase and I. Sucan, A Search Engine for Mathematical Formulae. In: T.Ida, J.Calmet and D. Wang (eds.), *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*, Springer Verlag, pp. 241-253, 2006.

[KovacsEtAl05] L. Kovacs, N. Popov and T. Jebelean, Verification Environment in Theorema. In: *Annals of Mathematics, Computing and Teleinformatics (AMCT)* vol. 1, no. 3, pp. 27–34, 2005.

[Kutsia03] T. Kutsia, Equational Prover of Theorema. In: R. Nieuwenhuis (ed.), *Proceedings of the 14th International Conference on Rewriting Techniques and Applications* (RTA'03), LNCS 2706, pp. 367-379, Springer, 2003.

[Kutsia02] T. Kutsia, Theorem Proving with Sequence Variables and Flexible Arity Symbols. In: M.Baaz and A.Voronkov (eds.), *Logic in Programming, Artificial Intelligence and Reasoning. Proceedings of the 9th International Conference LPAR'02*, LNAI 2514, pp. 278–291, Springer, 2002.

[KutsiaNakagawa01] T. Kutsia and K. Nakagawa, An Interface between Theorema and External Automated Deduction Systems. In: Steve Linton and Roberto Sebastiani (eds.), *Proceedings of 9th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning* (Calculemus '01), pp. 178–182, Siena, Italy, June 2001.

[Lange08] C. Lange: SWiM – A Semantic Wiki for Mathematical Knowledge Management. In: S. Bechhofer, M. Hauswirth, J. Hoffmann and M. Koubarakis (eds.), *Proceedings of the European Semantic Web Conference 2008*, Demo track, LNCS 5021, Springer, 2008.

[Larson06] C.E. Larson, *A Updated Survey of Research in Automated Mathematical Conjecture–Making*, preprint, 2006.

[Larson05] C.E. Larson, A Survey of Research in Automated Mathematical Conjecture–Making. In: *Proceedings of DIMACS Workshop on Automated Discovery in Chemistry and Mathematics*, 2005.

[LibbrechtMelis06] P. Libbrecht and E. Melis, Methods for Access and Retrieval of Mathematical Content in ActiveMath. In: N. Takayama, A. Iglesias and J. Gutierrez (eds.), *Proceedings of ICMS–2006*, LNCS 4151, Springer, 2006.

[Lucene] *Apache Lucene*, http://lucene.apache.org, last acces February 2009.

[MACE] *Mace 4*, http://www.cs.unm.edu/~mccune/mace4/, last acces February 2009.

[MacLaneBirkhoff67] S. MacLane and G. Birkhoff, *Algebra*, MacMillan, New York, 1967.

[MathML] *The MathML 2.0 Recommandation*, http://www.w3.org/TR/MathML/, last acces February 2009.

[MathSciNet] *AMS Mathematical Reviews on the Web*, http://www.ams.org/mathscinet, last acces February 2009.

[MathWebSearch] *Math Web Search*, http://www.mathweb.org/mws/, last acces February 2009.

[MBase] *MBase*, http://mbase.mathweb.org:8080/mbase/, last acces March 2007.

[McCaslandBundy06] R. McCasland and A. Bundy, MATHsAiD: a Mathematical Theorem Discovery Tool. In: T. Jebelean and V. Negru (eds.), Proceedings of SYNASC' 06, pp. 17–22, IEEE Computer Society Press, 2006.

[McCaslandEtAl07] R. McCasland, A. Bundy, and S. Autexier, Automated discovery of inductive theorems. In R. Matuszewski and P. Rudnicki (eds.), *From Insight to Proof: Festschrift in Honor of Andrej Trybulec*, Bialystok, 2007.

[McCaslandEtAl06] R. McCasland, A. Bundy, and P. Smith, *Ascertaining Mathematical Theorems*, Electronic Notes in Theoretical Computer Science, vol. 151 no. 1, pp. 21–38, Elsevier, 2006.

[McCune03] W.W. McCune, *Mace4 Reference Manual and Guide*, Argonne National Laboratory no. ANL/MCS-TM-264, 2003.

[MelisEtAl05] E. Melis, P. Kärger and M. Homik, Interactive Concept Mapping in ActiveMath (iCMap), In: J. M. Haake, U. Lucke, and D. Tavangarian (eds.), *3. Deutsche eLearning Fachtagung Informatik* (Delfi 2005), LNI 66, September 2005, pp. 247–258, Rostock, Germany.

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

[MillerYoussef03] B. Miller and A. Youssef, Technical Aspects of the Digital Library of Mathematical Functions, *Annals of Mathematics and Artificial Intelligence*, vol. 38, pp. 121–136, 2003.

[Metamath] *Metamath*, http://metamath.org/, last access February 2009.

[MinerTopping03] R. Miner and P. Topping, *Math on the web status report*, Design Science Inc. 2003,
Available online at: http://www.dessci.com/en/reference/webmath/status/, last access February 2009.

[Mizar] *Mizar*, http://www.mizar.org, last access February 2009.

[MKMNet] *MKM Network*, http://monet.nag.co.uk/mkm/index.html, last access February 2009.

[MKMNet4.1] Mathematical Knowledge Management Network (MKMNet), *Survey of existing tools for Formal MKM*, Deliverable 4.1.

[MONET] *MONET*, http://monet.nag.co.uk/cocoon/monet/index.html, last access February 2009.

[MOWGLI] *MOWGLI*, http://mowgli.cs.unibo.it/, last access February 2009.

[NieuwenhuisRubio01] R. Nieuwenhuis and A. Rubio, Paramodulation-Based Theorem Proving, *Handbook of Automated Reasoning* I(7), Elsevier Science and MIT Press, 2001.

[Noether21] E. Noether, *The Theory of Ideals in Ring Domains*, Mathematische Annalen, vol. 83, pp. 24–66, 1921.

[Omega] *Ωmega*, http://www.ags.uni-sb.de/~omega/omega/index.php, last access February 2009.

[OMDoc] *OMDoc*, http://www.omdoc.org/, last access February 2009.

[OpenMath] *OpenMath*, http://www.openmath.org/, last access February 2009.

[Otter] *Otter*, http://www-unix.mcs.anl.gov/AR/otter/, last access February 2009.

[Piroi04] F. 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.

[PiroiBuchberger05] F. Piroi and B. Buchberger, Label Management in *Theorema.* In: M. Kohlhase (ed.), *Informal Proceedings of the 4th International Conference on Mathematical Knowledge Management*, Bremen, Germany, July 2005.

[PiroiBuchberger04] F. Piroi and B. Buchberger, *An Environment for Building Mathematical Knowledge Libraries.* In: W. Windsteiger and Christoph Benzmueller (eds.), Proceedings of the Workshop on Computer-Supported Mathematical Theory Development, Second International Joint Conference (IJCAR), pp. 19–29, Cork, Ireland, July 2004.

[PiroiEtAl08] F. Piroi, B. 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] F. Piroi, B. Buchberger, Camelia Rosenkranz, T. Jebelean. *Organisational Tools for MKM in Theorema*. Technical report no. 07-11 in RISC Report Series, University of Linz, Austria, 2007.

[Pitts03] A. M. Pitts, *Nominal Logic, a First Order Theory of Names and Bindings*, Information and Computation vol. 186, pp. 165–193, Elsevier, USA, 2003.

[Prevosto05] V. Prevosto, Certified Mathematical Hierarchies: the FoCal System. In *Mathematics, Algorithms, Proofs*, 2005.

[RiazanovVoronkov99] A. Riazanov and A. Voronkov. Vampire. In H. Ganzinger (ed.), Proceedings of the 16th International Conference on Automated Deduction (CADE'99), LNAI 1632, pp. 292–296, Springer, Trento, Italy, 1999.

[RosenkranzEtAl09] C. Rosenkranz, B. Buchberger and T. Jebelean, Knowledge Archives in Theorema: A Logic-Internal Approach, Submitted to the *MCS Special Issue on Authoring, Digitalization and Management of Mathematical Knowledge*. Available as: Technical report no. 09-01 in RISC Report Series, University of Linz, Austria, January 2009.

[RudnickiTrybulec01] P. Rudnicki and A. Trybulec, Mathematical Knowledge Management in Mizar, In B. Buchberger and O. Caprotti (eds.), *Proceedings of MKM2001*, 2001.

[Urban06] J. Urban, MizarMode–Integrated Proof Assistance Tools for the Mizar Way of Formalizing Mathematics. In: C. Benzmueller (ed.), *Journal of Applied Logic*, vol. 4, no. 4, 2006.

[Urban05] J. Urban, MOMM - Fast Interreduction and Retrieval in Large Libraries of Formalized Mathematics, In *International Journal on Artificial Intelligence Tools*, 2005.

[Urban05a] J. Urban, MPTP–Motivation, Implementation, First Experiments, *Journal of Automated Reasoning*, 2005

[Theorema] *Theorema*, http://www.theorema.org, last access February 2009.

[TPS] *TPS*, http://gtps.math.cmu.edu/tps.html, last access February 2009.

[Youssef07] A. Youssef, Methods of Relevance Ranking and Hit-content Generation in Math Search. In: M. Kauers, M. Kerber, R. Miner and W. Windsteiger (eds.), *The 6th International Conference on Mathematical Knowledge Management*, pp. 393–406, Springer, Hagenberg, Austria, June 2007,.

[YoussefShatnawi06] A. Youssef and M. Shatnawi, Math Search with Equivalence Detection Using Parse-tree Normalization, The *4th International Conference on Computer Science and Information Technology*, April 2006, Amman, Jordan.

[Youssef05] A. Youssef, Information Search And Retrieval of Mathematical Contents: Issues And Methods. *The ISCA 14th International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE-2005)*, Toronto, Canada, July 20–22, 2005.

[Youssef04] A. Youssef, Search Systems for Math Equations, Invited talk at the *IMA Workshop on "Special Functions in the Digital Age"*, University of Minnesota, Minneapolis, MN, July, 2004.

[ZentralblattMath] *Zentralblatt MATH - ZMATH Online Database*, http://www.zentralblatt-math.org/zmath/en/, last access February 2009.

[ZimmerEtAl02] J. Zimmer, A. Franke, S. Colton and G. Sutcliffe. Integrating HR and tptp2X into MathWeb to Compare Automated Theorem Provers. *Proceedings of the CADE'02 Workshop on Problems and Problem Sets*, Copenhagen, Denmark, 2002.

[ZimmerAutexier06] J. Zimmer and S. Autexier. The MathServe Framework for Semantic Reasoning Web Services. In: U. Furbach and N. Shankar (eds.), *Proceedings of IJCAR'06*, LNCS 4130, pp. 140–145, Springer, 2006.

[WDML] *Göttingen Digitalisierung Zentrum*, http://gdz.sub.uni-goettingen.de/, last access February 2009.

[Wiedijk09] F. Wiedijk, *The mathematical vernacular*, unpublished note. Available online at: http://www.cs.kun.nl/~freek/notes/mv.ps.gz, last access December 2007.

[Wiedijk08] F. Wiedijk. *Mizar: an impression*, note, Available online at: http://www.cs.ru.nl/~freek/mizar/mizarintro.ps.gz, last access December 2008.

[WIRIS] *WIRIS Editor*, http://www.wiris.com/content/view/20/3/, last access February 2009.

[Wolfram03] S. Wolfram, *The Mathematica Book*, Wolfram Media Inc., 5th edition, 2003.

[WongSaunders05] Q. Wong and B. Saunders, Web-Based 3D Visualization in a Digital Library of Mathematical Functions, In *Proceedings of Web3D 2005 (10th International Conference on 3D Web Technology)*, ACM SIGGRAPH, 2005.