Lecture Notes in Artificial Intelligence 4108

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Jonathan M. Borwein   William M. Farmer (Eds.)

Mathematical
Knowledge Management

5th International Conference, MKM 2006
Wokingham, UK, August 11-12, 2006
Proceedings

Springer

Volume Editors

Jonathan M. Borwein
Dalhousie University
Faculty of Computer Science
6050 University Avenue, Halifax, Nova Scotia B3H 1W5, Canada
E-mail: jborwein@cs.dal.ca

William M. Farmer
McMaster University
Department of Computing and Software
1280 Main Street West, Hamilton, Ontario L8S 4K1, Canada
E-mail: wmfarmer@mcmaster.ca

Library of Congress Control Number: 2006930246

CR Subject Classification (1998): I.2, H.3, H.2.8, I.7.2, F.4.1, H.4, C.2.4, G.4, I.1

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-540-37104-4 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-37104-5 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springer.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11812289 06/3142 5 4 3 2 1 0

Preface

This volume contains the proceedings of the 5th International Conference on Mathematical Knowledge Management (MKM 2006) held August 10-12, 2006 in Wokingham, UK. Previous international MKM conferences were at Hagenburg, Austria (September 2001), Bertinoro, Italy (February 2003), Białowieża, Poland (September 2004), and Bremen, Germany (July 2005).

Mathematical knowledge management (MKM) is an emerging interdisciplinary field of research in the intersection of mathematics, computer science, library science, and scientific publishing. Its main objective is to develop new and better ways of managing mathematical knowledge using sophisticated software tools. Two other important aims are to obtain a better understanding of the nature of mathematical knowledge and to investigate new modes of consuming and producing mathematical knowledge. The MKM conferences bring together mathematicians, software developers, users of mathematics, librarians, publishers, and educators who are interested in advancing the management of mathematical knowledge.

MKM 2006 received 22 submissions. Of these, 20 were reviewed by three Program Committee members or external reviewers and two were reviewed by two. The reviews were exceptionally positive, and as a result, the Program Committee decided to accept 20 papers for presentation at the conference and publication in this proceedings volume.

We would like to thank Gregory J. Chaitin and Abdou Youssef for agreeing to give invited talks at MKM 2006. This volume includes an abstract of Dr. Chaitin's talk and a full written version of Dr. Youssef's talk. We are also grateful to the Program Committee and the external reviewers for their review of the submissions and to the Conference Chairs, Andrew Adams and Paul Cairn, for taking care of the local arrangements. We used the EasyChair conference management system, developed by Andrei Voronkov, to facilitate the paper submission, the review process, and the preparation of the proceedings. EasyChair performed beautifully and saved us a huge amount of time.

June 2006

Jonathan M. Borwein
William M. Farmer

Conference Organization

Program Chairs

Jonathan M. Borwein, Dalhousie University, Canada
William M. Farmer, McMaster University, Canada

Program Committee

Andrew A. Adams, University of Reading, UK
Alessandro Armando, University of Genova, Italy
Paul Cairns, University College London, UK
Michiel Hazewinkel, CWI, Netherlands
Alejandro Jofre, University of Chile, Chile
Michael Kohlhase, International University Bremen, Germany
Dan Lozier, National Institute of Standards and Technology, USA
Robert Miner, Design Science, USA
Jim Pitman, University of California at Berkeley, USA
Andrzej Trybulec, University of Bialystok, Poland
Alf van der Poorten, Macquarie University, Australia
Stephen Watt, University of Western Ontario, Canada
Bernd Wegner, Technical University of Berlin, Germany
Freek Wiedijk, Nijmegen University, Netherlands

Conference Chairs

Andrew Adams, University of Reading, UK
Paul Cairns, University College London, UK

External Reviewers

Jacques Carette
Alessandro Coglio
Pierre Corbineau
Luís Cruz-Filipe
Chris Hamilton
Mason Macklem
Lionel Elie Mamane
Jacopo Mantovani
Andrew McCallum
Normen Müller
Immanuel Normann
Clare So
Dan Synek
Herre Wiersma
Jian Xu

Table of Contents

Invited Talks

The Omega Number: Irreducible Complexity in Pure Math
Gregory J. Chaitin 1

Roles of Math Search in Mathematics
Abdou Youssef 2

Contributed Papers

Structured Induction Proofs in Isabelle/Isar
Makarius Wenzel 17

Interpretation of Locales in Isabelle: Theories and Proof Contexts
Clemens Ballarin 31

A Dynamic Poincaré Principle
Manfred Kerber 44

A Proof-Theoretic Approach to Tactics
Kamal Aboul-Hosn 54

A Formal Correspondence Between OMDoc with Alternative Proofs and the $\overline{\lambda}\mu\tilde{\mu}$-Calculus
Serge Autexier, Claudio Sacerdoti-Coen 67

Proof Transformation by CERES
Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, Hendrik Spohr 82

Synthesizing Proof Planning Methods and Ω-Ants Agents from Mathematical Knowledge
Serge Autexier, Dominik Dietrich 94

Verifying and Invalidating Textbook Proofs Using Scunak
Chad E. Brown 110

Capturing Abstract Matrices from Paper
Toshihiro Kanahori, Alan Sexton, Volker Sorge, Masakazu Suzuki 124

Towards a Parser for Mathematical Formula Recognition
Amar Raja, Matthew Rayner, Alan Sexton, Volker Sorge 139

Stochastic Modelling of Scientific Terms Distribution in Publications
Rimantas Rudzkis, Vaidas Balys, Michiel Hazewinkel 152

Capturing the Content of Physics: Systems, Observables, and Experiments
Eberhard R. Hilf, Michael Kohlhase, Heinrich Stamerjohanns 165

Communities of Practice in MKM: An Extensional Model
Andrea Kohlhase, Michael Kohlhase 179

From Notation to Semantics: There and Back Again
Luca Padovani, Stefano Zacchiroli 194

Managing Informal Mathematical Knowledge: Techniques from Informal Logic
Andrew Aberdein 208

From Untyped to Polymorphically Typed Objects in Mathematical Web Services
William Naylor, Julian Padget 222

Managing Automatically Formed Mathematical Theories
Simon Colton, Pedro Torres, Paul Cairns, Volker Sorge 237

Authoring LeActiveMath Calculus Content
Paul Libbrecht, Christian Gross 251

Information Retrieval and Rendering with MML Query
Grzegorz Bancerek 266

Integrating Dynamic Geometry Software, Deduction Systems, and Theorem Repositories
Pedro Quaresma, Predrag Janičić 280

Author Index 295

The Omega Number: Irreducible Complexity in Pure Math

Gregory J. Chaitin

IBM Research, Yorktown Heights, NY 10598, USA
chaitin@us.ibm.com

Abstract. We discuss the halting probability Ω, whose bits are irreducible mathematical facts, that is, facts which cannot be derived from any principles simpler than they are. In other words, you need a mathematical theory with N bits of axioms in order to be able to determine N bits of Ω. This pathological property of Ω is difficult to reconcile with traditional philosophies of mathematics and with traditional views of the nature of mathematical proof and of mathematical knowledge. Instead Ω suggests a quasi-empirical view of math that emphasizes the similarities between mathematics and physics rather than the differences.

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, p. 1, 2006.

Roles of Math Search in Mathematics*

Abdou Youssef **

Department of Computer Science, The George Washington University,
Washington DC 20052, USA
ayoussef@gwu.edu
http://www.seas.gwu.edu/~ayoussef/

Abstract. Math-aware fine-grain search is expected to be widely available. A key question is what roles it can play in mathematics. It will be argued that, besides finding information, math search can help advance and manage mathematical knowledge. This paper will present the short-term goals and state of the art of math-aware fine-grain search. Afterwards, it will focus on how math search can help advance and manage mathematical knowledge, and discuss what needs to be done to fulfill those roles, emphasizing two key components. The first is similarity search, and how it applies to (1) discovering and drawing upon connections between different fields, and (2) proof development. The second is math metadata, which math search will surely encourage and benefit from, and which will be pivotal to mathematical knowledge management.

1 Introduction

Since the advent of the Worldwide Web, serious efforts have been undertaken to create digital libraries of mathematical contents, and to develop languages, tools, and systems for faster dissemination and processing of such contents [1,3,5,11,12,13,14,16,20,21,28,31,32,33,36,21,22]. For digital libraries of mathematics to serve their purpose fully, users need to be able to search easily and effectively, especially for equations, functions, structures, proof patterns, and other kinds of fine-grained mathematical constructs. Although text search has reached a high level of maturity [38,34], mathematical expressions are highly symbolic and structured, and are not curreently searchable by the available text-search systems.

Field-based search systems are now widely deployed in several mathematics databases and by many mathematical content providers, such as Zentralblatt's ZMATH and MathDi [40,23], the Jahrbuch Database [15], AMS's MathSCiNet [2], and various professional mathematical socities. These systems afford users more targeted search, such as search by author, subject, title, abstract, journal, series, reviewer, review text, and the like. Standard subject classifications, such

* This work was supported in part by The National Foundation Grant No. 0208818.

** Some of the work was done as part of the DLMF Project at the National Institute of Standards and Technology, USA.

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 2–16, 2006.

as MSC 2000 [30], helps to a considerable extent in focusing the search. Neverthless, like text-search systems, the current field-based search systems are neither meant nor able to provide access to fine-grained mathematical data.

It will probably be much more useful to the mathematical and scientific communities to have *math-aware fine-grain search* systems. The author has been conducting research and development on that kind of search [29,39]; much of that effort is part of the Digital Library of Mathematical Functions (DLMF) project [18,19,29]. The immediate goal of the research on math search is to create math-aware systems that (1) enable users to search not only for text, but also for fine-grain mathematical data, such as equations, functions, and structures; and (2) allow users to express math queries naturally and easily, using the notation and idiom of mathematicians and scientists.

Math-aware fine-grain search holds considerable promise for the short term and the long term. For the short term, it will help users fulfill momentary information needs. Whenever a user needs information about a specific mathematical item, s/he formulates and submits a query to the search system, which processes the query and returns a number of matching hits, ranked by relevance (or by some other user-specified criteria). The user will then browse through the returned hits, looking for the truly relevant ones which satify the need that prompted the search. At times, the user may have to refine their queries and repeat the search cycle. However, it is expected that the math-awareness of the search system is likely to identify much more relevant matches, and the fine-grain nature of the search leads to hits that point to **small-size** units of information. These two outcomes will greatly reduce the amount of time a user spends on searching and browsing through hits to find what is needed, and thus enable the user to return quickly to the main task at hand.

For the long term, math-aware fine-grain search holds promises that have potentially broader scope and greater impact. Specifically, it will be argued in this paper that such a search capability can contribute to the advancement and management of mathematical knowledge. For example, math-aware search can be used to find similarities between a piece of mathematics being developed, on the one hand, and proved theorems and well-developed theories in the same or different fields of mathematics, on the other hand, thus pointing the mathematician to fruitful methodological directions and interesting connections (note: two expressions or patterns are similar if some appropriately defined distance between their structures is below a certain threshold). Furthermore, through similarity search, it is possible to provide interactive *computer-aided proving* (CAP), either as a standalone system or as a complement to proof planning systems (e.g., λClam [6,9] and Omega [4,26,27]). In the standalone mode, a CAP system can, by constant monitoring of an evolving proof or at the prompting of a user, automatically search for similar proof patterns, and thus periodically suggest to the mathematician relevant strategies, tactics, and/or logic rules that can be applied to further the proof. In the other mode, as a component of a proof planning system (PPS), math-aware fine-grain search can help the user first find initial plans of proof (of "similar" theorems), and later in the proof process find refinement tactics, all through ongoing search for similar plans and tactics against either a standalone knowledge base or Web-accessible math repositories of appropriately marked up contents and proof patterns. These and other potentialities of math-aware fine-grain search will be discussed later in the paper.

With regard to contributing to mathematical knowledge management, math-aware fine-grain search can help classify manuscripts. The search, in a semi- or fully-automated classification environment, can be used to find similarities and associations between different manuscripts. Using the similarities and associations, a librarian or a system can classify and characterize (with metadata) a previously uncategorized document, by borrowing the classes and descriptive metadata of the search-discovered similar documents. Furthermore, in a radical departure from current practices, this process of classification and metadata-enrichment can and sould be done at fine ganularity — at the level of equations, functions, structures, proof patterns, and the like. This can be done **by** and **for** math-aware fine-grain search.

It is evident from the above that similarity-search and metadata are fundamental to those envisioned long-term roles of math search, and to the symbiotic relationship between search, management, and advancement of mathematical knowledge. Similarity-search, a fairly developed area in data mining applications [37], is a new area in math search, and will be discussed later in this paper. As for metadata, international, professional, and academic efforts towards developing math metadata have been initiated, such as the MathNet project in Germany [25], and the activities of the International Mathematics Metadata Task Force (and its affiliated American task force) [24]. The planned metadata of those efforts seem to be at a coarse-grained level: at the level of books, articles and manuscripts. The benefits of such efforts towards improved access, dissemination, and management of mathematical knowledge, will be considerable. They will be even greater if the metadata is at a fine-grain level. Of course, providing metadata at any level, but especially at a fine-grain level, is a daunting task. Therefore, automatic generation of metadata is indispensable.

This paper will address the short-term and long-term objectives, roles and capabilities of math-aware fine-grain search. Specifically, the paper will identify the main aspects and pertinent issues, present the state of the art, and, where possible, outline approaches to follow.

2 Math-Aware Fine-Grain Search

This section will address the basic objectives, and issues, and state of the art of math-aware fine-grain search.

As a result of the work on and experience with the development of math search on the DLMF, the author has identified some key objectives that math search systems ought to meet, at least to a significant extent. The next subsection describes those objectives.

2.1 Basic Objectives of Math Search

1. **Math-awareness:** Much of the mathematical knowledge is embodied in mathematical symbols, elaborate notations, and structures of various levels of complexity. So for math-search systems to be effective, they have to recognize mathematical symbols and structures.
2. **A natural math-query language:** A math search system must provide an intuitive yet expressive math query language. Users in the mathematical and scientific communities should be able to express their queries in the same way as they would write other mathematical expressions, such as in a Latex-like syntax. Table 2.1 shows several examples of queries and describes the corresponding matching records.
3. **Fine granularity of searchable and retrievable information units:** With the vast and fast-increasing amount of mathematical knowledge available for electronic access, it is desirable to search for the most targeted information, be it an equation, an integral, a differential equation, a Fourier transform of a function, a definition, a graph, a theorem, a proof technique, etc. If such is the size (granularity) of what a user needs in a given situation, it would be a waste of the user's precious time to provide him/her a larger amount of information and expect him/her to sift through it to locate the relatively tiny piece of interest. Therefore, an important objective of math search is to aford users the ability to search for and retrieve fine-grain targets. (A *target* or *record* is any searchable and retrievable information unit in a database.)
4. **Perfect recall:** Recall is a standard metric in all search systems; the recall per query is defined to be the ratio of the number of relevant hits to the total number of relevant targets in the database. It is a universal objective of search to maximize recall. That is, every target that matches a query must be included in the hitlist.
5. **Perfect precision:** Like recall, precision is another performance metric of all search systems; the precision per query is defined to be the ratio of the number of relevant hits to the number of hits in the hitlist. Every attempt should be made to maximize precision. That is, every hit in the hitlist must match the query; the hitlist should not contain any false hits.
6. **Perfect relevance-ranking:** Ideally, if hit A is more relevant than hit B, then A should appear before B in the hitlist. In particular, the most relvant hit(s) must appear on top of the hitlist, or at least near the top. This objective is particularly pressing because of the very large and ever increasing number of potentials hits.
7. **Useful highlighting:** Highlighting within a retrieved target should be done in a way that informs and justifies to the user why the target matched, and which specific parts matched. For very fine-grained targets, such as an equation or a graph, highlighting is not so critical, but for large-grained targets such as an article or manuscript, highlighting is very desirable to help the user identify quickly the more relevant parts of the hit.
8. **Minimum hit-redundancy:** In systems where targets at different levels of granularity are available, some targets may be subsets of other targets, such

as a separately accessible equation that is a part of a separately accessible section. In such environments, redundant hits are possible. For example, if target A is a subset of target B, and if B matches a query only because A matches the query, then presenting both A and B as two separate hits in the hitlist constitutes redundancy. Hit A should be presented, and B should be left out. The objective is to eliminate redundancy. If that is too costly, an attempt must be made to reduce the effect of redundancy; for example, have hit B appear much later than hit A in the hitlist. Note that if the targets are disjoint, no redundancy should arise; redundant hits would be a reflection of poor system design.

Table 1. Examples of Queries

Query	**Matching Records**
sin^2 x+cos^2 x	Those containing the expression $\sin^2 x + \cos^2 x$
`J_n(z)=`	Those containing the fragment "$J_n(z) =$"
`Gamma(1/2)=`	Those containing "$\Gamma(\frac{1}{2}) =$", for the values of $\Gamma(\frac{1}{2})$
sqrt(Ai^2+Bi^2)	Those containing the expression $\sqrt{Ai^2 + Bi^2}$
^(x+2)	Those containing $x + 2$ as an exponent
int^infinity	Those containing $\int^\infty$
int (sin x)/x dx	Those containing $\int \frac{\sin x}{x} dx$
DeMoivre and cos (n theta)	Those containing both "DeMoivre" and $cos(n\theta)$
"Fourier transform" and spheroidal	Those showing Fourier transforms of spheroidal functions, in addition to those containing the terms "Fourier transform" and "spheroidal"
Ai and Bessel	Those showing connections between Airy Ai and Bessel functions, in addition to those containing the terms "Ai" and "Bessel"
Ai = BesselK	Ideally, those containing equations expressing the Airy Ai function in terms of the Bessel function K

2.2 Issues and Policy Decisions

In meeting those objectives, several fundamental issues must be faced and some policies for resolving them must be implemented. Here are some of the more important issues and challenging policy decisions that have to be handled.

- **Target definition and granularity:** The designer must define what should be a searchable and retrievable target, and decide on the appropriate granularities of targets.
- **Literal vs. abstract understanding and weighting of query terms:** Mathematics is rife with abstraction and levels of abstraction. As a simple example, the name of a function argument is not to be taken literally, whereas the standard name of an elementary function or a special function should be taken literally. Another aspect is whether users can characterize rather than specify the terms that must occur in the matching targets. For example,

can users enter "trigonometric" to stand for any term that is the name of a trigonometric function?

- **Whether to return mathematically equivalent hits, and to what extent:** Many a mathematical concept or expression can be expressed in several equivalent forms. The question is whether or not documents that do not contain a literal match of a query expression but contain an equivalent expression should be returned as hits. If the search is for "$\sin(\frac{\pi}{2}-x)$", should the system return documents containing the equivalent expression "$\cos x$"? How about if the query is "$\frac{1}{x}$" and a document contains "x^{-1}"? Some equivalences are so commonplace that users may wish them to be detected and matched in search, while other less familiar equivalences would cause confusion if detected and matched. The extent of equivalence-awareness in search is a serious design decision. Of course, the implementatioin of "deep-equivalence" awareness is a major task that requires sophisticated mathematical reasoning algorithms.
- **Determination of the intented meaning of a user's query:** There is considerable "overloading" of names and notation, i.e., the same symbol referring to different things in different contexts. For example, the zeta symbol (ζ) can refer to the Jacobi zeta function, the Weierstrass zeta function, the Riemann zeta function, or a generic symbol with no specific denotation. If a user includes zeta in a query and has a specific context in mind (e.g., number theory) but that context is not communicated in the query, the system will have no way of determining which zeta occurrences to match, or how best to rank the hits, creating a likely situation of high user dissatisfaction with the results.

All but the first point above involve the extremely challenging problem of determining the user's intent and wishes, without soliciting too much information per query from the user. Decision policies are needed in order to make "educated guesses" about the user's intent and wishes from the limited information provided in the query, and, accordingly, to determine what targets are truly relevant and how to relevance-rank the various hits. For more accurate assessment of relevance, the context of the search must be determined, such as the user's field and level of expertise, and the area of interest at the time of the search. It is worth noting that relevance is a relatively old, open question in the general field of text information retrieval (IR) [35], and the issue of context-based search is a current research topic with considerable interest in the IR community [17].

2.3 State of the Art of Math-Aware Fine-Grain Search

As mentioned in the Introduction, all search systems deployed by the current mathematics databases and mathematical content providers are conventional coarse-grain field-based text search systems with little math-awareness. In math-aware search, some work has started to appear. Recently, Guidi et al published papers on a math query language MathQL [12] and related searching techniques [11], both of which are for RDF metadata repositories, where RDF is the XML-based metadata markup language standard. The MathQL syntax is a markup style that is advanced in its expressive power, and requires the users to be advanced mathematicians. An earlier effort in math-aware fine-grain search is the work by Einwohner and Fateman [10], which was limited to integral-lookup.

The most recent work on math-aware fine-grain search is the work on the DLMF search [29,39]. All the eight objectives presented in Subsection 2.1 have been met to a large extent. The resulting system, to be deployed in the near future, is fully math-aware and supports search and access to fine-grain targets such as equations, figures, tables, definitions, and named rules/theorems. It allows users to submit queries with Latex-like syntax. It achieves perfect precision and recall as far as term-occurrence search is concerned; also, through metadata enrichment, additional relevant hits are matched beyond literal occurence of terms. Relevance ranking is satisfactory, and is being improved. Small-grain targets (such as equations and figures) are highlighted when displayed within larger documents (such as sections). Finally, redundancy is greatly minimized, and when users restrict the search results to a specific type (such as equations or figures), no redundancy arises.

3 Objectives and Roles of Math Search in the Long Term

Beyond the conventional search for documents, it is envisioned that math search can fullfil higher-level and farther-reaching roles. This section will discuss some of those roles.

3.1 Discovery of Similarities Between Fields

Research in an evolving new field (or sub-field) may discover preliminary patterns and laws that happen to be similar to those in older, more established fields. Early discovery of such similarities may suggest new patterns, laws, and properties, which are well-established in the older fields, to explore in the context of the new field. Also, proven useful methodologies in the older fields may apply to the new field fruitfully. The bridging and borrowing apply to both broad methodologies and specific proof techniques & patterns.

Math search can help in the discovery of such similarities — as long as the content repositories are well-formatted, adequately marked up, and accessible. Section 4 will discuss methods of discovering and measuring mathematical similarities. (Recall from the Introduction that two expressions or patterns are similar if some appropriately defined distance between their structures is below a certain threshold. That is, the two expressions/patterns are similar if their structures are identical or near-identical.)

Suffice it to say at this point that a search-driven technology of similarity-discovery is likely to increase productive interdisciplinary activities, not only between mathematicians of different specialties, but also between mathematicians and researchers in the natural and even social sciences. Indeed, it is often the case that scientists, who are in other disciplines than Mathematics and happen to be engaged in some mathematical work related to their disciplines, need to know what mathematical theories and knowledge can help them advance their fields, and which mathematicians are doing such work and can thus be invaluable collaborators. A math-similarity search capability can help such scientists locate relevant mathematical work and potential collaborators.

3.2 Computer-Aided Proving

A second major role that math-aware fine-grain search can play is computer-aided proving (CAP). That can take at least two shapes: (1) a straightforward online computer-aided proving (O-CAP) role, and (2) a more elaborate interactive real-time computer-aided-proving (R-CAP) role. Both are discussed next.

Online computer-aided-proving. A user engaged in developing a proof for a theorem can, at various junctures of the proof development, submit expressions and Logical patterns (from the evolving proof) as queries. Matches may contain "identical" or similar proofs that are complete and valid; such proofs can then be mimicked, or learned from, to complete the proof at hand in an analogy-driven fashion. Also, atomic entities, expressions, and possible patterns from the premises (or conclusions) of the to-be-proved theorem can be submitted as queries, to search for similar theorems; the corresponding proofs may serve as a good aid to the proof at hand. Note that this O-CAP functionality is easy to have and use, for it is nothing more than straightforward math-aware fine-grain search.

Real-time computer-aided-proving. This is similar to O-CAP except that no explicit queries need be formulated and submitted by the user. Instead, a software system will, in the background and during the course of a proof-development, carry out the following steps:

1. monitor the evolving proof;
2. formulate&submit queries (from the expressions and logical patterns present in the partial proof);
3. search for similar expressions and logical patterns
4. evaluate, rank, and distill the returned matches; the distilling involves
   - identifying the known properties of entities (e.g., functions and operators) and of the premises/hypotheses; the entities and premises are those that are in the theorem or in the emerging proof.
   - identifying intermediate theorems/lemmas, as when the query consists of premises (from the original theorem or the emerging proof), and the matching hit is a theorem with the same premises; the conclusions of those matching theorems, and possibly their proofs, as well as bibliographic references to them, will be among the distilled materials presented to the user.
5. report the distilled results in real time as suggested directions (tactics) and intermediate sub-conclusions to the mathematician that is developing the proof;
6. repeat this cycle (steps 1-5) throughout the proof development, until the end of proof.

Note that in Step 3 of the R-CAP cycle, as well as in O-CAP, the search can be conducted not only against a local knowledge base, but also against all kinds of math repositories. For this to work, the math repositories must be well-formatted, adequately marked up, and indexed for searching. Such repositories are growing in size and number. They include: the DLMF [18,19,29]; MBase [16]; Mizar (at mizar.org); and so on.

An R-CAP implementation can be very much like *integrated development environments* (IDEs), which are very widely used by software developers in the computer science community. (Good Latex editors are small instances of IDEs). In an IDE, static and locational dynamic menus are available. The static menus offer fixed services and functionalities. Dynamic menus are menus whose items change depending on the context, and are populated by search systems working in the background; those menus pop up when the user mouses over certain words or commands in the file, or when the user types up the first few characters of certain patterns. An R-CAP IDE can behave in similar ways by popping up dynamic menus containing suggestions for new logical patterns/tactics/rules to follow, and those suggestions vary depending on where in the proof the user is, and what premises and intermediate conclusions have been put in the proof. The suggestions in the dynamic menus will be constantly gathered and updated by the R-CAP math search, which is working in the background. The search items that populate the dynamic menus in typical IDEs are usually obtained from search against an internal database as well as against the opened file. In an R-CAP IDE, however, the search can be extended beyond a local knowledge base (KB) and the opened file, to include Web-accessible knowledge bases; the user of the IDE would also have the configuration option of specifying which specific external KBs to make use of.

CAP as presented above bears a strong relation to proof assistants and proof planning systems in particular. Proof planning was introduced by Alan Bundy for inductive theorem proving [6,7], and was implemented in the systems Clam λClam, and IsaPlaner [8]. The Omega system [4,26,27] extended Clam's proof-planning paradigm to knowledge-based proof planning. Proof planning systems start with an abstract-level proof plan, and then "carry out" the abstract-level plan, interactively (with the user) and recursively when needed, by expanding the steps into concrete sequences of logical steps.

Therefore, the proof-planning approach (of Omega and Clam) to proving is primarily top down: from an abstract proof-plan to a final detailed proof. The search-driven R-CAP approach, described above, is fundamentally an incremental, bottom-up approach, driven mainly by the direction that the mathematician is taking in the proof, but at the same time helping the mathematician to further that direction along, or suggesting alternative tactics and patterns as a result of similarity search.

3.3 Learning Aid

In addition to its research-furthering roles, math search can be used by math & science educators and students for educational purposes: finding what they need, and learning from what they find. It is an obvious and natural role of any search system.

In the context of math education, however, some issues arise. One important issue is the relationship between the granularity of the retrieved information, on the one hand, and the information need and the educational level of the user, on the other hand. For example, if a physicist is seeking the value of an integral or the general solution to a specific differential equation, the search results should be at the level of an equation, rather than the title of a book about the subject. Likewise, if a novice wishes to learn about number theory, the search results should be books and perhaps articles about the subject, rather than stand-alone equations about the Riemann ζ function or the Euler φ function. The notions of relevance and context-based search mentioned earlier are pertinent here.

Another issue is how best to integrate math-aware search into math learning systems in a synergistic fashion. Like many of the ideas discussed in this paper, this integration topic is in its infancy, and will require considerable research.

3.4 Routing

Routing is the process of informing users (or subscribers) of the latest information that match a pre-determined query specified by the user, as soon as the information becomes available. Math search can be used to stream to a mathematician all articles and manuscripts that match the mathematician's pre-specified query (or queries) whenever and as soon as the information becomes available. The source of the information can be professional societies, publishers, or researchers posting their manuscripts on their institutions's Web sites. The system(s) to do the routing can be centralized systems on the information providers' Web servers, or federated systems that periodically "crawl" the Web (or at least certain specific sites) searching for newly posted information. Either way, math search must be a central component of the routing system, and the posted information must be formatted and marked up adequately, and indexed, in order for the background searching to take place and the search results to be routed to the appropriate users, each according to his or her pre-specified queries.

4 Methods for Discovering and Measuring Similarity

As seen throughtout the paper, similarity search is useful in many contexts. It is referred to sometimes as approximate search or fuzzy search. Before one can proceed further, a formal definition of similarity is called for.

Definition 1. *Given a distance metric d in the "space" of mathematical expressions or patterns, and given a threshold h, two expressions or patterns E_1 and E_2 are said to be h-*similar *if $d(E_1, E_2) < h$.*

Two remarks must be made. First, the distance d need not be a distance in the strict topological sense, nor the "space" of expressions or patterns need necessarily be a topological space. Rather, d should satisfy the two properties

$$d(E_1, E_2) \Leftrightarrow E_1 = E_2,$$

$$d(E_1, E_2) = d(E_2, E_1)$$

But the triangle inequality is not essential.

Second, the actual definition of the distance d must capture, to the extent possible, the intuitive subjective notion of (dis)similarity between expressions or patterns. For example, $x^2 + y^2$ and $\cos^2\theta + \sin^2\theta$ are intuitively similar expressions, whereas $x^2 + y^2$ and $\int xdx$ are quite dissimilar. Also, d must capture comparative information about similarity, that is, if E_1 is more similar to E_2 than F_1 is to F_2, then we should have $d(E_1, E_2) < d(F_1, F_2)$. For example, one would expect that $d(x^2 + y^2, u^2 + v^2) < d(x^2 + y^2, x^2 + y)$.

Ideally, the distance d should be sensitive to the notion of different levels of abstraction. Specifically, if an expression E is an abstraction of another expression F, and F is in turn an abstraction of G, then one should have:

$$d(E, F) < d(E, G), \text{ and } d(F, G) < d(E, G).$$

Furthermore, if E and F are mathematically equivalent expressions but have different structures, as is the case for the two expressions $(x+1)^2$ and x^2+2x+1, then one would expect that $d(E, F) = 0$. This expectation, however, assumes that the similarity system incorporates the detection of logical equivalence and value-equavalence, which is a rather difficult problem of symbolic computation and automated mathematical reasoning. Therefore, for pragmatic reasons, one may wish to leave out the requirement that $d(E, F) = 0 \Leftrightarrow E \equiv F$, although preserving it can lead to much more interesting similarity results, at the cost of much more computationally intensive similarity measurement.

With all those considerations in mind, one approach to quantifying similarity (or distance) between mathematical expressions and patterns is by modeling expressions as parse trees with node labels that represent the names of functions/operators in the expression. Similarity (or distance) can then be defined using the structures and node labels of the trees. The more nodes with like-labels in the two trees, the more similar the trees. Also, the more sub-trees of identical structures that the two trees share in common, the more similarity there is. Similarity between the internal (non-leaf) nodes in the two trees is more important than similarity between the labels of the leaves in the two trees, because leaves often represent arbitrary variable names, while non-leaves represent essential operational and structural information of a math expression. Also, structural and label similarities higher up the two trees are often more important than those further down the trees, because the "fundamental" structure of a formula/expression is reflected more near the root of the parse tree. These differences in importance suggest weighted measures of similarity, where higher nodes and higher subtrees are assigned more weight than the lower ones.

The precise development of those ideas of quantifuing similarity, and the development of algorithms for measuring similarity (or distance), are subjects of ongoing research in the author's research group.

One final note is that once one has an adequate definition of similarity (or distance) and a good algorithm for computing the distance between two expressions/patterns, it is straightforward to incoporate the distance and its algorithm into math-aware fine-grain search systems for performing similarity search, at whatever level of desirable similarity (as specified by the threshold h).

5 Approaches for Generating Fine-Grained Metadata

In most application, metadata is generated manually. In fact, in many instances, metadata is extrinsic to the object being described, such as the date and journal of a publication; therefore, such metadata cannot be derived in any other way but manually. Fortunately, extrinsic metadata is small in size, and need be enetered at the coarse-grain level (i.e., at the level of books, articles and manuscripts). In the case when the metadata describes something intrinsic, such as the properties of a certain function, the properties may be so complex and intricate that only the author or a domain expert is in a position to unearth them and state them explicitly. For the sake of fine-grain search, the metadata will have to be entered at the level of equations, definitions, functions, proof patterns, and the like. The metadata must also to be marked up properly so search systems can make use of them. Both the metadata generation & entry, and the marking up, are time-intensive tasks that few authors would be willing to do. Therefore, it is preferable to automate the math-metadata generation process.

Most mathematical functions and concepts enjoy many properties and fall under a hierarchy of mathematical categories. To illustrate, assume that an equation (or math file) E has the cosine function ”cos” in it. This function falls in the category of trigonometric functions, which is a subcategory of elementary functions, which in turn is a subcategory of special functions. It also enjoys the property of periodicity, among other things. Recall from earlier discussions that such properties are desirable to have as metadata. A user may wish to search for equations that have, among other things, periodic functions (or trigonometric functions, etc.). Clearly, even if the equation/file E does not contain explicitly any of those terms or phrases (”periodic functions” or ”trigonometric functions”), E is a relevant object and should be returned as a hit. But without metadata, this is not possible.

We have developed a knowledge-based approach to generating metadata. First, a knowledge base was compiled, consisting of standard math functions and operators, on the one hand, and associated metadata on the other hand. Specifically, for every function and operator in the KB, the corresponding metadata is a set of descriptive phrases that name the properties that the function/construct enjoys, and the mathematical categories that the function/construct falls under. Afterwards, we developed algorithms that, for each equation and math expression, generate from the KB a combined list of the descriptive phrases of all the functions and constructs that occur in that equation/expression, and treat that combined list as metadata for that equation/expression. The approach was enhanced further through using the context of a math expression to derive additional metadata phrases. Specifically, the titles of the containing sections/subsections, the captions of the containing tables, and similar headers, can be used as sources of additional metadata. However, care must be taken when using context information, because every item of information in the context applies to every equation or expression in that context. For example, if the title of a subsection is "Fourier and Laplace Transforms", and the subsection contains several equations, some being Fourier transforms, and others Laplace trasnforms, then latching the entire title of the subsection to each equation in the subsection leads to inaccuaries.

A more powerfull approach to automated fine-grain-metadata generation derives "higher-order" metadata from the structure of an equation/expression, not just from the functions and constructs that occur in it. For example, the Fourier transform has a recognizable expression structure; an algorithm can be written to search for such structures in equations and expressions, and wherever found, associate the metadata phrase "Fourier transform" with the containing equations/expressions. This higher-order metadata generation requires (1) defining structural patterns (and their characteristics) for a number of mathematical constructs, such as the Fourier transform, the Laplace transform, (partial/ordinary) differential equations, recurrence relations, and so on; and (2) developing algorithms for recognizing such structural patterns in math expressions so corresponding metadata can be associated with them. One method for expression-structure recognition is to use expression parse trees, specifically their structures and the internal (i.e., operation) node labels, as templates for pattern recognition & classification.

Acknowledgements

The author thanks the DLMF team at the National Institute of Standards and Technology, USA, especially Drs. Bruce Miller, Dan Lozier, Ron Boisvert, Frank Olver, and Charles Clark, for the many discussions and their valuable comments on various aspects of math search.

References

1. The ActiveMath Project, http://www.mathweb.org/activemath/
2. MathSciNe. American Mathematical Society (AMS). `http://www.ams.org/mathscinet`
3. Asperti, A. et al: Mathematical Knowledge Management in HELM [Italy]. First International Workshop on Mathematical Knowledge Management, Schloss Hagenberg, Austria, September 24-26, 2001.
4. Benzmüller, C. et al: Ω: Towards a Mathematical Assistant, Conference on Automated Deduction, 1997.
5. Buchberger, B.: Mathematical Knowledge Management Using Theorema. First International Workshop on Mathematical Knowledge Management, Schloss Hagenberg, Austria, September 24-26, 2001.
6. Bundy, A.: The Use of Explicit Plans to Guide Inductive Proofs. R. Lusk and R. Overbeek (eds) Proceedings of the 9th Conference on Automated Deduction (CADE 9), Springer-Verlag, (1988) 111–120

7. Bundy, A.: Proof Planning. B. Drabble (ed) Proceedings of the 3rd International Conference on AI Planning Systems, (1996) 261–267.
8. Dixon, L. and Fleuriot, J. D.: IsaPlanner: A prototype proof planner in Isabelle. In F. Baader (ed), CADE 19, volume 2741 of LNCS, 279–283. Springer 2003.
9. Dixon, L., Jamnik, M., and Pollet, M.: Proof planning: Comparing Ωmega, λClam and Isa-Planner. In B. Bennett (ed) ARW 11, 50–52. University of Leeds, School of Computing, 2004. Held in Association with the AISB'04 Convention.
10. Einwohner, T. H. and Fateman, R.: Searching techniques for integral tables. International symposium on Symbolic and algebraic computation, ACM, 1995. (http://torte.cs.berkeley.edu:8010/tilu)
11. Guidi., F.: Searching and Retrieving in Content-based Repositories of Formal Mathematical Knowledge. Ph.D. Thesis in Computer Science, University of Bologna, March 2003. Technical report UBLCS 2003-06.
12. Guidi, F. and Schena, I.: A Query Language for a Metadata Framework about Mathematical Resources. The 2nd International Conf. Mathematical Knowledge Management, Bertinoro, Italy, Feb. 2003.
13. Hardin, T.: Mathematical Knowledge Management in FOC [France]. First International Workshop on Mathematical Knowledge Management, Schloss Hagenberg, Austria, September 24-26, 2001.
14. An Hypertextual Electronic Library of Mathematics, http://helm.cs.unibo.it/.
15. Jahrbuch Database. http://www.emis.de/MATH/JFM/JFM.html
16. Kohlhase, M.: MBase: Representing Knowledge and Context for the Integration of Mathematical Software Systems. Journal of Symbolic Computation 23:4 (2001) pp. 365 – 402
17. Leake, D. B. and Scherle, R.: Towards Context-Based Search Engine Selection. IUI01, January 14-17, 2001, Santa Fe, New Mexico, USA.
18. Lozier, D. W.: The DLMF Project: A New Initiative in Classical Special Functions. International Workshop on Special Functions - Asymptotics, Harmonic Analysis and Mathematical Physics. Hong Kong, June 21-25, 1999.
19. Lozier, D.W., Miller, B.R., and Saunders, B.V.: Design of a Digital Mathematical Library for Science, Technology and Education. Proceedings of the IEEE Forum on Research and Technology Advances in Digital Libraries; IEEE ADL '99, Baltimore, Maryland , May 1999.
20. MathWeb.org, http://www.mathweb.org/
21. The OpenMath Standard, 1998, http://www.openmath.org/
22. MathML 2.0, a W3C Recommendation, October, 2003, http://www.w3.org/Math/
23. MathDi (Mathematics Didactics Database). http://www.emis.de/MATH/DI/
24. Mathematics Metadata. http://www.mathmetadata.org/
25. The MathNet Project. http://www.math-net.de/project/
26. Melis, E. and Siekmann, J.: Knowledge-Based Proof Planning. J. Artificial Intelligence, 1999.
27. Melis, E. and Meier, A.: Proof Planning with Multiple Strategies. First International Conference on Computational Logic, CL-2000 , 2000.
28. Melis, E. et al: ActiveMath: A Generic and Adaptive Web-Based Learning Environmen. Artifical Intelligence in Education,(12)4 Winter 2001.
29. Miller, B. and Youssef, A.: Technical Aspects of the Digital Library of Mathematical Functions. Annals of Mathematics and Artificial Intelligence, **Vol. 38** (2003) 121–136
30. Mathematical Subject Classification (MSC2000). American Mathematical Society. http://www.ams.org/msc/

31. MathWeb.org, http://www.mathweb.org/
32. The Omega Group, http://www.ags.uni-sb.de/õmega/
33. Rudnicki, P. and Trybulec, A.: Mathematical Knowledge Management in MIZAR. First International Workshop on Mathematical Knowledge Management, Schloss Hagenberg, Austria, September 24-26, 2001
34. Salton, G. and McGill, M. J.: Introduction to Modern Information Retrieval. McGraw Hill, New York (1993)
35. Saracevic, T.: Relevance: A Review of and a Framework for the Thinking on the Notion. Journal of the American Society of Information Science, **26(4)** (1975) 321–343
36. Theorist Interactive LiveMath, http://www.livemath.com/
37. Tan, P.-N., Steinbach, M., and Kumar, V. : Introduction to Data Mining. Addison-Wesley (2006)
38. Baeza-Yates, R. and Ribeiro-Neto, B.: Modern information retrieval, Reading, MA: Addison-Wesley, (1999)
39. Youssef, A.: Information Search And Retrieval of Mathematical Contents: Issues And Methods. The proceedings of the ISCA 14th International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE-2005), July 20-22, 2005, Toronto, Canada.
40. Zentralblatt MATH database at European Mathematical Information Service (EMIS). http://www.emis.de/ZMATH/

Structured Induction Proofs in Isabelle/Isar

Makarius Wenzel

Technische Universität München
Institut für Informatik, Boltzmannstraße 3, 85748 Garching, Germany
http://www.in.tum.de/~wenzelm/

Abstract. Isabelle/Isar is a generic framework for human-readable formal proof documents, based on higher-order natural deduction. The Isar proof language provides general principles that may be instantiated to particular object-logics and applications. We discuss specific Isar language elements that support complex induction patterns of practical importance. Despite the additional bookkeeping required for induction with local facts and parameters, definitions, simultaneous goals and multiple rules, the resulting Isar proof texts turn out well-structured and readable. Our techniques can be applied to non-standard variants of induction as well, such as co-induction and nominal induction. This demonstrates that Isar provides a viable platform for building domain-specific tools that support fully-formal mathematical proof composition.

1 Motivation

1.1 The Isar Philosophy

Isabelle/Isar [15, 16, 7, 17] is intended as a generic framework for developing formal mathematical documents with full proof checking. The Isabelle/Isar system is well integrated with existing theorem prover interface technology [1] and document preparation based on PDF-LaTeX.[1] The main objective is the design of a human-readable structured proof language, which is called the "primary proof format" in Isar terminology.

Such a primary proof language is somewhere in the middle between the extremes of primitive proof objects and actual natural language. In this respect, Isar is a bit more formalistic than Mizar [12, 10], using explicit logical connectives for certain reasoning schemes where Mizar would prefer English words; see [19, 18] for further comparisons of these systems. We argue that any effort of building a library of formalized mathematics heavily depends on a reasonable notion of structured proofs — the Mizar Mathematical Library [6] provides some empirical evidence for this.

So Isar challenges the traditional way of recording informal proofs in mathematical prose, as well as the common tendency to see fully formal proofs directly as objects of some logical calculus (e.g. λ-terms in a version of type theory). In fact, Isar is better understood as an interpreter of a simple block-structured language for describing data flow of local facts and goals, interspersed with occasional invocations of proof methods. Everything is reduced to logical inferences internally, but these steps are somewhat marginal compared to the overall bookkeeping of the interpretation process. Thanks to

[1] In fact, the present paper has been prepared as an Isabelle/Isar theory document, which means that the proofs and proof outlines encountered here have been checked by the system.

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 17–30, 2006.

careful design of the syntax and semantics of Isar language elements, a formal record of Isar instructions may actually appear as an intelligible text to the attentive reader.

Consuming well-structured Isar proofs is less demanding than producing them in the first place. The main effort is to get started with new formalizations, while modifying existing proofs is much easier. With sufficient background material available, a proof author will first consume previous developments, and then produce some derivative work. This general procedure is taken for granted in science, but is not quite established yet in formal theory development — despite the relative success of the Mizar Mathematical Library.

1.2 Generic Natural Deduction

Isabelle/Isar is based on the existing logical framework of Isabelle/Pure [9], which provides a generic platform for higher-order natural deduction. The generic approach of Pure inference systems is extended by Isar towards actual proof texts. Applications require another intermediate layer: an object-logic. Presently Isabelle/HOL [8] (simply-typed higher-order logic) is being used most of the time. Isabelle/ZF is less extensively developed, although it would probably fit better for classical mathematics.

The Isar proof language offers various common principles that are parameterized by entities of the object-logic and application context, including plug-in proof methods. The most basic method is called *rule* and refers to a generic natural deduction step of the underlying framework (essentially combined forward-backward chaining). The application context provides declarations of canonical introduction and elimination rules for various kinds of connectives and derived operations, covering the object-logic $\wedge$, $\vee$, $\forall$, $\exists$, ..., set-theory $\cap$, $\cup$, $\bigcap$, $\bigcup$, ..., and any application specific notions such as operators of lattice theory, topology, etc. With reasonable rule declarations in the context, proofs involving common elements of discourse proceed in an implicit manner, where rules are determined by the structure of propositions stated explicitly. For example:

```
assume x ∈ A and x ∈ B
then have x ∈ A ∩ B by rule
```

Abbreviations like "**..**" for "**by** *rule*" make this less formalistic. Similarly **proof** ... **qed** already initiates a certain reasoning pattern from the context, unless an alternative is named explicitly. Note that beyond selecting single rules by the syntactic structure of goals and facts, which works by means of higher-order unification, Isabelle/Isar never appeals to hidden automated reasoning. Any such proof tools need to be invoked explicitly, usually in terminal position such as "**by** *simp*".

The subsequent example performs basic natural deduction with an explicit rule applied in a purely backwards manner; the arising sub-proofs are completed accordingly.

```
fix n :: nat
have P n
proof (rule nat-induct)
  show P 0 ⟨proof⟩
next
```

```
    fix $n$
    assume $P\ n$
    show $P\ (Suc\ n)$  $\langle proof \rangle$
  qed
```

This is already a structured induction proof in Isabelle/Isar, so in theory we could conclude the present paper just now. In practice, though, various issues arise in presenting an inductive statement $P\ n$ properly — slightly annoying auxiliary structure can occur here. We shall introduce a significantly improved version of the Isar *induct* method that enables extraneous logical bookkeeping to be suppressed from the proof text.

1.3 Case-Study: Complete Induction with Local Definitions

Consider the following problem of recreational mathematics: "Given some function f on natural numbers, such that $f\ (f\ n) < f\ (n + 1)$ for all n, show that f is the identity." Here we formalize only the easier part: $n \leq f\ n$ for arbitrary n. The proof is by complete induction on $f\ n$, along the $<$ relation on natural numbers.

We shall work in Isabelle/HOL, which uses fairly standard mathematical notation. Special attention needs to be paid to the "three arrows" $\Rightarrow / \bigwedge / \Longrightarrow$ of Isabelle/Pure (cf. §2.1), which occur whenever abstract syntax or inference rules of the logical framework need to be introduced explicitly. Note that Isar proofs may reference previous facts either by name (e.g. *asm*) or proposition (e.g. $\langle n = m + 1 \rangle$). Furthermore "**.**" stands for "**by** *this*", i.e. immediate composition of facts without any rule in between.

```
lemma example:
  fixes $f :: nat \Rightarrow nat$
  assumes asm: $\bigwedge n.\ f\ (f\ n) < f\ (n + 1)$
  shows $n \leq f\ n$
proof (induct $k = f\ n$ fixing: $n$ rule: less-induct)
  case (less $k$)
  then have IH: $\bigwedge m.\ f\ m < f\ n \Longrightarrow m \leq f\ m$ by simp
  show ?case
  proof cases
    assume $n = 0$
    then show ?thesis by simp
  next
    assume $n \neq 0$
    then obtain $m$ where $n = m + 1$ by (cases $n$) simp-all
    from asm have $f\ (f\ m) < f\ n$ unfolding $\langle n = m + 1 \rangle$ .
    with IH have $f\ m \leq f\ (f\ m)$ .
    also note $\langle f\ (f\ m) < f\ n \rangle$
    finally have $f\ m < f\ n$ .
    with IH have $m \leq f\ m$ .
    also note $\langle f\ m < f\ n \rangle$
    finally have $m < f\ n$ .
    then show $n \leq f\ n$ using $\langle n = m + 1 \rangle$ by simp
  qed
qed
```

The general structure of our proof is as follows: (i) the main statement (lines 1–4), (ii) initiating the induction (lines 5–8), (iii) splitting the proof body into two cases and solving the trivial one (lines 9–12), (iv) finish the interesting second case with two appeals to the induction hypothesis (lines 13–23).

Part (i) already starts the proof by giving a structured Isar statement, which consists of several proof context elements (**fixes**, **assumes**) followed by the main conclusion (**shows**). Thus we may commence the actual reasoning immediately, without decomposing the problem into its constituent parts first. The final result will be extracted from the initial statement as a closed formula, namely $(\bigwedge n. f\ (f n) < f\ (n + 1)) \Longrightarrow n \leq f n$.

Part (ii) shall be our main technical concern. We have used the *induct* method, providing some additional information about the precise mode of reasoning to be performed. The arguments include the induction variable k, which is locally defined as $k = f n$. We also fix n as an auxiliary induction parameter now, because the induction step will require different instances of $f n$. We further need to specify the underlying induction principle *less-induct*, because the default rule for natural numbers would merely consider zero and immediate successor cases. The subsequent **case** element augments the proof context by additional parameters and assumptions stemming from the induction step (rule *less-induct* has only one such case, namely *less*). From this we conclude the induction hypothesis *IH* in a convenient form — the raw hypothesis would still mention the local definition of $k = f n$, but we prefer the expanded version here. Then we are ready to work on the inductive conclusion: "**show** *?case*" (the abbreviation of *?case* for the recurrent proposition $n \leq f n$ also stems from the *less* context).

Part (iii) is not very special, but note that the *cases* method applies *tertium-non-datur*.

Part (iv) is most interesting from the mathematical viewpoint. Technically, we merely conduct a few steps of calculational reasoning [3] in Isar (with "glue statements" **also** and **finally**), while results are composed in a mostly trivial manner; we rarely invoke automated reasoning support here. In line 14, syntactic case analysis converts between $m + 1$ and *Suc m*, which is the preferred representation in the Isabelle/HOL library.

In informal mathematical practice, one would probably just present the main body of part (iv), and include some hints about the induction. Compared to that, our proof is almost twice as long. In particular, part (ii) requires further investigation: the 4 lines being spent here are not that bad, as we shall see now.

The following version performs the induction without any specific support. In order to present the problem as a closed inductive proposition, the local definition and variable generalization is simulated within the object-logic as $\forall n.\ k = f n \longrightarrow n \leq f n$.

```
lemma example:
  fixes f :: nat ⇒ nat
  assumes asm: $\bigwedge n. f\ (f n) < f\ (n + 1)$
  shows $n \leq f n$
proof –
  { fix k have $\forall n.\ k = f n \longrightarrow n \leq f n$
    proof (rule less-induct)
      fix k assume less: $\bigwedge m.\ m < k \Longrightarrow \forall n.\ m = f n \longrightarrow n \leq f n$
      show $\forall n.\ k = f n \longrightarrow n \leq f n$
      proof
11              fix $n$
12              show $k = f\,n \longrightarrow n \le f\,n$
13              proof
14                assume $k = f\,n$
15                with $\mathit{less}$ have $\mathit{IH}$: $\bigwedge m.\ f\,m < f\,n \Longrightarrow m \le f\,m$ by $\mathit{simp}$
16                show $n \le f\,n$ $\langle \mathit{proof} \rangle$
17              qed
18            qed
19          qed
20        } then show $\mathit{?thesis}$ by $\mathit{simp}$
21      qed
```

Here we have spent 21 lines without doing anything useful, since the proof of $n \le f\,n$ in line 16 has been left out! (Lines 9–23 of the previous version may be pasted here.)

Dealing with compound formulas such as $\forall n.\ k = f\,n \longrightarrow n \le f\,n$ imposes extra inconveniences in our framework, because the object-language needs to be decomposed into Pure first. Even worse, this extra work is multiplied in induction proofs: (i) reformulate the original problem (line 6), (ii) decompose both the induction hypothesis and conclusion in the proof body (lines 8–15), (iii) apply the modified result to solve the main problem (line 20). Some parts have been automated in an ad-hoc fashion using "**by** *simp*".

One could argue now that Isar should not insist on Pure rule composition, but let the user transform the logical structure of a pending problem more directly. This would approximate the Mizar language, with separate language elements to dig into connectives of first-order logic. On the other hand, it would not really solve the problem at hand. Turning $n \le f\,n$ into $\forall n.\ k = f\,n \longrightarrow n \le f\,n$ already demands efforts that are irrelevant to the application. Since interactive proof development usually requires some experimentation to figure out the induction parameters in the first place, we would like to reduce logical clutter as much as possible.

As a rule of thumb, Mizar is better at decomposing statements of first-order logic, but Isar does not require any such decomposition, if we manage to represent a problem in Pure form. The basic idea of our enhanced *induct* method is to make complex induction proof patterns appear as a native part of the Isar framework. This is achieved by internalizing portions of Isar proof context into the object-logic, and reverse the effect before handing over to the user to finish the induction cases.

Subsequently, we shall present further details of the Isar framework in §2, describe the *induct* method in §3, and review common induction patterns in §4.

2 Foundations

Isabelle/Isar consists of three main layers: The Isabelle/Pure framework for primitive natural deduction, Isar proof contexts for managing various kinds of local parameters, assumptions, facts, abbreviations etc., and the Isar/VM ("virtual machine") interpreter that implements an incremental model of structured proof composition.

2.1 The Pure Framework

The Pure logic [9] is an intuitionistic fragment of higher-order logic. In type-theoretic parlance, there are three levels of λ-calculus with corresponding arrows: $\Rightarrow$ for syntactic function space (terms depending on terms), $\bigwedge$ for universal quantification (proofs depending on terms), and $\Longrightarrow$ for implication (proofs depending on proofs).

Term syntax provides explicit notation for abstraction $\lambda x :: \alpha.\ b(x)$ and application t u, while types are usually implicit thanks to type-inference; terms of type *prop* are called propositions. Logical statements are composed via $\bigwedge x :: \alpha.\ B(x)$ and $A \Longrightarrow B$. Primitive reasoning operates on judgments of the form $\Gamma \vdash \varphi$, with standard introduction and elimination rules for $\bigwedge$ and $\Longrightarrow$ that refer to fixed parameters $x_1, \ldots, x_m$ and hypotheses $A_1, \ldots, A_n$ from the context Γ. The corresponding proof terms are left implicit.

An object-logic introduces another layer: type o for object propositions, term constants $Tr :: o \Rightarrow prop$ as (implicit) object-truth judgment and connectives like $\wedge :: o \Rightarrow o \Rightarrow o$ or $\forall :: (\alpha \Rightarrow o) \Rightarrow o$, and axioms for object rules such as *conj-I*: $A \Longrightarrow B \Longrightarrow A \wedge B$ or *all-I*: $(\bigwedge x.\ B\ x) \Longrightarrow \forall x.\ B\ x$. Derived object rules are represented as theorems of Pure.

Since Pure propositions may be nested arbitrarily, the resulting natural deduction framework extends Gentzen's version [5] such that rules may take arbitrary rules as assumptions [11]. For example, the induction rules $P\ 0 \Longrightarrow (\bigwedge n.\ P\ n \Longrightarrow P\ (Suc\ n)) \Longrightarrow P\ n$ and $(\bigwedge n.\ (\bigwedge m.\ m < n \Longrightarrow P\ m) \Longrightarrow P\ n) \Longrightarrow P\ n$ both fit nicely into this framework.

2.2 Isar Proof Contexts

In judgments $\Gamma \vdash \varphi$ of the primitive framework, Γ essentially acts like a proof context. Isar elaborates this idea towards a higher-level notion, with separate information for type-inference, term abbreviations, local facts, and generic hypotheses parameterized by discharge rules. For example, the context element **assumes** A introduces a hypothesis with $\Longrightarrow$ introduction as discharge rule; **notes** $a = b$ defines local facts; **defines** $x = a$ and **fixes** $x :: \alpha$ introduce local terms.

Top-level theorem statements may refer directly to such Isar elements to establish a conclusion **shows** B within a certain context; the final result will be in discharged form. There are separate Isar commands to build contexts within a proof body, notably **fix**, **assume**, **obtain** etc. Using Isar proof notation, we can explain the latter three formally:

```
{                        {                        {
  fix x                    assume A                 obtain x where A x ⟨proof⟩
  have B x ⟨proof⟩          have B ⟨proof⟩            have B ⟨proof⟩
}                        }                        }
note ‹⋀x. B x›           note ‹A ⟹ B›             note ‹B›
```

Here **note** is used to indicate the fact resulting from each proof block. The **obtain** above involves a proof of $\bigwedge C.\ (\bigwedge x.\ A\ x \Longrightarrow C) \Longrightarrow C$, which happens to be the body of $\exists$ elimination in first-order logic; it then acts like **fix** x **assume** $A\ x$ using that discharge rule. See [16, §5.3] for more on generalized elimination in Isar.

Isar also supports named context segments called *cases*: the language element **case** c expands to **fix** $x_1 \ldots x_m$ **assume** $A_1 \ldots A_n$ according to a given definition of c in the context; the variant **case** $(c\, y_1 \ldots y_k)$ specifies explicit names for fixed variables introduced here. Cases may also provide term abbreviations, typically *?case* for the conclusion. Isar proof methods are entitled to define cases for the current proof body.

2.3 Isar/VM Transitions

A structured Isar proof text consists of a sequence of proof commands, which are interpreted as transitions of the Isar virtual machine. The basic idea is analogous to evaluating algebraic expressions on a stack machine: $(a + b) \cdot c$ then corresponds to a sequence of single transitions for each symbol $($, a, $+$, b, $)$, $\cdot$, c. In Isar the algebraic values are local facts or goals, and the operations are logical inferences.

The Isar/VM state maintains a stack of nodes, each node contains the local proof context, an optional pending goal, and the linguistic mode. The latter determines the type of transition that may be performed next, it essentially alternates between forward and backward reasoning. For example, in *state* mode Isar acts like a mathematical scratch-pad, which accepts various context declarations like **fix**, **assume**, and goal statements like **have** and **show**. A goal changes the mode to *prove*, which means that we may now refine the problem via **unfolding** or **proof**. Then we are again in *state* mode of a proof body, which may issue **show** statements to solve pending subgoals. A concluding **qed** will return to the original *state* mode one level upwards. Isar provides convenient abbreviations, such as "**by** *meth*$_1$ *meth*$_2$" (terminal proof) for "**proof** *meth*$_1$ **qed** *meth*$_2$".

In the following sequence of Isar/VM transitions we indicate block structure and mode:

have $A \longrightarrow B$	*open*	*state* → *prove*
proof		*prove* → *state*
assume A		
show B	*open*	*state* → *prove*
⟨*proof*⟩	*close*	*prove* → *state*
qed	*close*	

In structured Isar proof texts, *state* mode is active most of the time, while *prove* mode is left immediately after 1 or 2 refinement steps steps (say to unfold a definition and apply a standard rule). In contrast, unstructured tactic scripts would stay in *prove* mode indefinitely, applying more and more backward refinements until the goal is solved.

3 The *induct* Proof Method

The *induct* proof method of Isar is a sophisticated wrapper for the basic *rule* method. Arguments provided in the proof text are interpreted as directions for local modifications of the proof context, induction rule, and goal, culminating in the actual logical refinement step. The resulting proof body will contain several subgoals corresponding to the inductive cases of the instantiated rule after refinement, together with named cases to enable abbreviations **case** c **show** *?case* for each induction step, instead of explicit **fix** $x_1 \ldots x_m$ **assume** $A_1 \ldots A_n$ **show** B. The general method invocation looks like this:

$$\begin{array}{l} \mathit{facts} \\ (\mathit{induct}\ \mathit{insts}\ \ \mathit{fixing: vars}\ \ \mathit{rule: rule}) \end{array}$$

Here *facts* refers to the implicit facts being passed to any Isar method according to the proof structure, as indicated by **then** or **using** in the text; *insts* is a list of instantiations, either x (variable) or $x = a$ (defined variable); *vars* refers to a list of fixed variables; *rule* refers to the underlying induction principle.

Any of these arguments are optional, and there are sensible defaults. For example, giving $n :: \mathit{nat}$ as *insts* determines the rule *nat.induct*; giving $\vdash x \in A$ as *facts* determines both the instantiation x and rule $A.\mathit{induct}$ (for some inductively defined set A in the context). Since mutual induction is also supported, arguments *insts*, *vars*, *rule* may be repeated, using the separator **and**. For example, $(\mathit{induct}\ t :: \mathit{term}$ **and** $\mathit{ts} :: \mathit{term\ list})$ could refer to the mutual induction principle stemming from a nested datatype in HOL.

3.1 Generic Induction Rules

We support a variety of induction rules, with minimal assumptions about the logical presentation. Following canonical Pure formulas, rules represent the outermost $\bigwedge$ prefix as schematic variables. The remainder is an iterated implication $\Longrightarrow$, with arbitrary "major premises" in front, followed by the inductive cases, concluded by some predicate expression $\mathit{Tr}\ (P\ x_1 \ldots x_m)$; recall that the object judgment Tr is usually implicit.

The inductive cases may not introduce further schematic variables, apart from those already present in the major premise and the conclusion. Thus inductive cases are fully determined after instantiating the other parts of the rule. Occurrences of the predicate in the cases is restricted to $\mathit{Tr}\ (P\ a_1\ \ldots\ a_m)$ for arbitrary term arguments, but with the outer object judgment still intact. In other words, $P\ a_1\ \ldots\ a_m$ needs to be surrounded by $\bigwedge/\Longrightarrow$ connectives. Thus the predicate may essentially represent object rules.

For example, $(\bigwedge n.\ (\bigwedge m.\ m < n \Longrightarrow P\ m) \Longrightarrow P\ n) \Longrightarrow P\ n$ seen before qualifies as induction rule. The equivalent version $(\bigwedge n.\ \forall m{<}n.\ P\ m \Longrightarrow P\ n) \Longrightarrow P\ n$ in Isabelle/HOL does not work, because $\forall m < n.\ P\ m$ hides the predicate inside a closed object-formula. A rule for some inductive set A looks like $x \in A \Longrightarrow \mathit{cases} \Longrightarrow P\ x$, but not $\mathit{cases} \Longrightarrow \forall x \in A.\ P\ x$ as is occasionally seen in set-theory texts. Here the restricted conclusion has been split to exhibit the major premise $x \in A$ in frontmost position.

The object-logic and application context already declare common rules for inductive sets and types in proper form, so the user only needs to take care in unusual situations.

3.2 Internal Operations

Assume for the moment that only one goal and one induction rule is involved. Then the *induct* method performs operations on the proof context, rule, and goal as follows:

1. context: declare local *defs* for any defined induction variables $x = a$
2. rule: apply *insts* according to the positions in the inductive predicate $P\ x_1 \ldots x_n$
3. rule: expand *defs* in major premises (accommodate the original statement, which mentions the expanded expressions)

4. rule: consume a prefix of *facts* according to the number of major premises (the remainder is a pure induction rule $cases \Longrightarrow P\ x_1\ \ldots\ x_n$)
5. goal: strengthen by inserting the remaining *facts*, and all *defs*
6. goal: strengthen by fixing *vars*, using rule $(\bigwedge x.\ B\ x) \Longrightarrow B\ a$ of the framework
7. goal: internalize $\bigwedge/\!\!\Longrightarrow$ into the object-logic, using $(\bigwedge x.\ Tr\ (B\ x)) \equiv Tr\ (\forall x.\ B\ x)$ and $(Tr\ A \Longrightarrow Tr\ B) \equiv Tr\ (A \longrightarrow B)$ of the framework
8. rule: unify conclusion against goal (absorb any internalized auxiliary structure, including *defs*; result is a fully-instantiated induction rule)
9. rule: carefully recover internalized $\bigwedge/\!\!\Longrightarrow$ structure in the inductive cases, using the above rules backwards
10. context: extract inductive cases from rule (enable **case** abbreviations in the proof body; cases consist of elements stemming from both the rule and the goal)
11. context: discharge *defs*
12. goal: apply the fully instantiated rule

Simultaneous induction may involve both multiple goals and multiple rules (e.g. from mutually inductive sets or types). Isar already supports simultaneous goal statements, e.g. **shows** A **and** B. The *induct* method is able to pass this structure through the inductive process, using a local version of Pure conjunction $A\ \&\ B$ internally.

Mutual induction uses packs of rules that share common inductive cases, but deviate in the major premises and conclusion. The rule preparations above essentially work the same with parallel copies of method arguments *insts* and *vars* given for each *rule*. The pack is joined by & introduction before unifying against the goal. The inner conjunctive structure of the goal is internalized using $(Tr\ A\ \&\ Tr\ B) \equiv Tr\ (A \wedge B)$. The reverse step shuffles & outermost and eliminates it via $(A\ \&\ B \Longrightarrow C) \equiv (A \Longrightarrow B \Longrightarrow C)$. Extracted cases consist of a shared context (from the rule) and separate sub-cases (from the goals).

4 Common Induction Patterns

We briefly review common proof patterns supported by our generic *induct* method, using the induction rule of Peano natural numbers as representative example. The proof outlines illustrate augmented contexts via literal facts, e.g. **note** $\langle\bigwedge x.\ A\ n\ x \Longrightarrow P\ n\ x\rangle$. In practice, these facts would just be used in their proper place without further notice.

4.1 Local Facts and Parameters

Augmenting a problem by additional facts and locally fixed variables is a bread-and-butter method in many applications. This is where unwieldy object-level $\forall$ and $\longrightarrow$ used to occur in the past. Now we are able to use primary means of the language, notably **using** in the proof text and *fixing* in the method specification.

```
lemma
  fixes n :: nat and x :: 'a
  assumes A n x
  shows P n x using ‹A n x›
proof (induct n fixing: x)
  case 0
```

```
    note prem = ‹A 0 x›
    show P 0 x ⟨proof⟩
  next
    case (Suc n)
    note hyp = ‹⋀x. A n x ⟹ P n x›
      and prem = ‹A (Suc n) x›
    show P (Suc n) x ⟨proof⟩
  qed
```

4.2 Local Definitions

Here the idea is to turn sub-expressions of the problem into a defined induction variable. This is often accompanied with fixing of auxiliary parameters in the original expression, otherwise the induction step would refer invariably to particular entities. This combination essentially expresses a partially abstracted representation of inductive expressions.

```
  lemma
    fixes a :: 'a ⇒ nat
    assumes A (a x)
    shows P (a x) using ‹A (a x)›
  proof (induct n = a x fixing: x)
    case 0
    note prem = ‹A (a x)›
      and def = ‹0 = a x›
    show P (a x) ⟨proof⟩
  next
    case (Suc n)
    note hyp = ‹⋀x. A (a x) ⟹ n = a x ⟹ P (a x)›
       and prem = ‹A (a x)›
       and def = ‹Suc n = a x›
    show P (a x) ⟨proof⟩
  qed
```

Observe how the local definition $n = a\ x$ recurs in the inductive cases as $0 = a\ x$ and $Suc\ n = a\ x$, according to underlying induction rule. Here we cannot apply definitional expansions immediately, in contrast to the purer induction rule encountered in §1.3.

4.3 Simple Simultaneous Goals

The most basic simultaneous induction operates on several goals one-by-one, where each case refers to induction hypotheses that are duplicated according to the number of conclusions.

```
  lemma
    fixes n :: nat
    shows P n and Q n
  proof (induct n)
    case 0 case 1
    show P 0  ⟨proof⟩
  next
    case 0 case 2
    show Q 0  ⟨proof⟩
  next
    case (Suc n) case 1
    note hyps = ‹P n› ‹Q n›
    show P (Suc n)  ⟨proof⟩
  next
    case (Suc n) case 2
    note hyps = ‹P n› ‹Q n›
    show Q (Suc n)  ⟨proof⟩
  qed
```

Note that induction with mutual rules is usually even simpler than this, because the collection of simultaneous goals will be consumed by the given pack of rules. The resulting proof body will not require additional nesting or duplication of cases, although separate projections of the mutual induction predicates may be mentioned.

4.4 Compound Simultaneous Goals

The following pattern illustrates the slightly more complex situation of simultaneous goals with individual local assumptions. In compound simultaneous statements like this, local assumptions need to be included into each goal, using $\Longrightarrow$ of the Pure framework. In contrast, local parameters do not require separate $\bigwedge$ prefixes here, but may be moved into the common context of the whole statement.[2]

```
lemma
  fixes n :: nat
   and x :: 'a and y :: 'b
  shows A n x ⟹ P n x
   and B n y ⟹ Q n y
proof (induct n fixing: x y)
  case 0
  { case 1
    note prem = ‹A 0 x›
    show P 0 x  ⟨proof⟩ }
  { case 2
    note prem = ‹B 0 y›
    show Q 0 y  ⟨proof⟩ }
next
  case (Suc n)
  note hyps = ‹⋀x. A n x ⟹ P n x› ‹⋀y. B n y ⟹ Q n y›
  then have some-intermediate-result  ⟨proof⟩
  { case 1
    note prem = ‹A (Suc n) x›
```

[2] Vacuous quantification does not impose any restriction in the Pure framework — unlike more complex versions of dependent type-theory.

```
      show P (Suc n) x ⟨proof⟩ }
    { case 2
      note prem = ‹B (Suc n) y›
      show Q (Suc n) y ⟨proof⟩ }
  qed
```

Here *induct* provides again nested cases with numbered sub-cases. Unlike the flattened version of §4.3, we are more careful here to share common parts of the body context. In typical applications, there could be a long intermediate proof of general consequences of the induction hypotheses, before finishing each conclusion separately.

5 Conclusion and Related Work

The present infrastructure for structured induction proofs has emerged from experience with applications of Isabelle/HOL, involving non-trivial reasoning about inductive types, sets and functions. Many Isar proofs in the Isabelle library benefit from this already (in post-2005 development versions). The method implementation provides sufficiently general building blocks to support non-standard variants of induction as well.

Co-induction is the logical dual of induction, and is both like and unlike standard induction in practical use. Instead of introducing a conclusion $P\ n$, co-induction eliminates a fact $\vdash P\ n$ (often written as set-membership). Our technique for *fixing* inductive parameters, essentially universal elimination applied backwards, becomes existential introduction applied forwards. Then the method invocation (*coinduct n fixing*: *x*) would correspond to the common technique in co-induction proofs to single-out certain expressions as existential parameters, which then recur in the co-induction step in eliminated form. The present *coinduct* method of Isabelle/Isar does not yet implement this elaborate scheme, but there are some examples available in *HOL*/*Library*/ *Coinductive-List* of the Isabelle distribution that illustrate common proof techniques.

Nominal induction [14, 13] augments traditional structural induction by specific means to reason about datatypes involving local binders (e.g. λ-calculus, Π-calculus, functional programming languages). The corresponding induction rules involve a "freshness context" as additional parameter for the inductive predicate. This fits well into our infrastructure for producing complex induction predicates conveniently. We merely need to support an additional "*avoiding*" argument, similar to the present "*fixing*". The resulting *nominal-induct* method has been used by Urban for some key induction proofs of the POPLmark challenge[3]. The complex pattern of §4.4 reflects the structure of a key induction proof in this application.

Other interactive provers provide surprisingly little support for complex inductions.

In Mizar [12, 10], induction is performed as a forward step, using basic "scheme" application. Thus the inductive cases need to be spelled out beforehand in full detail. (Isar is able to produce symbolic **case** elements due to backward refinement of a known statement and rule.) Mizar lacks specific support for compound inductive predicates.

In Coq [2], the default induct tactic passes *all* of the current proof context through the inductive process. Thus users may have to prune unwanted parts first. This corresponds

[3] http://fling-l.seas.upenn.edu/~plclub/cgi-bin/poplmark

to our internal treatment of $\bigwedge/\Longrightarrow$, only that Isar specifies wanted elements positively, instead of unwanted ones negatively. Coq lacks any further induction support beyond that, e.g. the user needs to encode defined inductive entities manually.

Other tactical provers, notably Isabelle/HOL [8], have traditionally avoided too elaborate induction tactics, because the exact portion of context to participate here is hard to delimit in unstructured goal states. Even worse, automated tools easily fail due to overly general induction hypotheses. Users are asked to present their problem in object-logic form, and manage the additional logical connectives by means of ad-hoc automation.

Isabelle/Isar provides a framework for checking structured proof documents, but the user needs to produce such texts manually. Proof planning techniques have recently been applied as additional means to assist proof authors in this process. The "proof-centric approach" of [4] aims to automate Isar proof construction, while preserving the ability of the author to intervene. The resulting IsaPlanner system focuses on induction and rippling, although it uses a less sophisticated version of Isar induction so far.

References

[1] D. Aspinall. Proof General: A generic tool for proof development. In *European Joint Conferences on Theory and Practice of Software (ETAPS)*, 2000.

[2] B. Barras et al. *The Coq Proof Assistant Reference Manual, version 8*. INRIA, 2006.

[3] G. Bauer and M. Wenzel. Calculational reasoning revisited — an Isabelle/Isar experience. In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in Higher Order Logics: TPHOLs 2001*, LNCS 2152, 2001.

[4] L. Dixon and J. D. Fleuriot. A proof-centric approach to mathematical assistants. *Journal of Applied Logic: Special Issue on Mathematics Assistance Systems*, To appear.

[5] G. Gentzen. Untersuchungen über das logische Schließen. *Math. Zeitschrift*, 1935.

[6] Mizar mathematical library. http://www.mizar.org/library/.

[7] T. Nipkow. Structured proofs in Isar/HOL. In H. Geuvers and F. Wiedijk, editors, *Types for Proofs and Programs (TYPES 2002)*, LNCS 2646. Springer, 2003.

[8] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. LNCS 2283. Springer, 2002.

[9] L. C. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*. Academic Press, 1990.

[10] P. Rudnicki. An overview of the MIZAR project. In *1992 Workshop on Types for Proofs and Programs*. Chalmers University of Technology, Bastad, 1992.

[11] P. Schroeder-Heister. A natural extension of natural deduction. *Journal of Symbolic Logic*, 49(4), 1984.

[12] A. Trybulec. Some features of the Mizar language. Presented at a workshop in Turin, 1993.

[13] C. Urban and S. Berghofer. A recursion combinator for nominal datatypes implemented in Isabelle/HOL. In *International Joint Conference on Automated Reasoning, IJCAR 2006*, LNCS. Springer, 2006.

[14] C. Urban and C. Tasson. Nominal techniques in Isabelle/HOL. In *Conference on Automated Deduction, CADE 2005*, LNCS 3632, 2005.

[15] M. Wenzel. Isar — a generic interpretative approach to readable formal proof documents. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Thery, editors, *Theorem Proving in Higher Order Logics: TPHOLs'99*, LNCS 1690, 1999.

[16] M. Wenzel. *Isabelle/Isar — a versatile environment for human-readable formal proof documents*. PhD thesis, Institut für Informatik, TU München, 2002. http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2002/wenzel.html.

[17] M. Wenzel and L. C. Paulson. Isabelle/Isar. In F. Wiedijk, editor, *The Seventeen Provers of the World*, LNAI 3600. Springer, 2006.

[18] F. Wiedijk. Comparing mathematical provers. In *Second International Conference on Mathematical Knowledge Management*, LNCS 2594, 2003.

[19] F. Wiedijk and M. Wenzel. A comparison of the mathematical proof languages Mizar and Isar. *Journal of Automated Reasoning*, 29(3-4), 2002.

Interpretation of Locales in Isabelle: Theories and Proof Contexts

Clemens Ballarin

Fakultät für Informatik
Technische Universität München
85748 Garching, Germany
http://www4.in.tum.de/~ballarin

Abstract. The generic proof assistant Isabelle provides a landscape of specification contexts that is considerably richer than that of most other provers. Theories are the level of specification where object-logics are axiomatised. Isabelle's proof language Isar enables local exploration in contexts generated in the course of natural deduction proofs. Finally, locales, which may be seen as detached proof contexts, offer an intermediate level of specification geared towards reuse. All three kinds of contexts are structured, to different extents. We analyse the "topology" of Isabelle's landscape of specification contexts, by means of development graphs, in order to establish what kinds of reuse are possible.

1 Introduction

Locales are Isabelle's [12] emerging mechanism to support abstract reasoning. They are modules whose specification and theorems can be transported to other contexts by either import or interpretation. Thus specification and/or theorems can be reused.

Three kinds of contexts can be distinguished in Isabelle. These are theories, locales and proofs. Theories are the outermost level of specification, at which object-logics like HOL (Higher-Order Logic) and ZF (Zermelo-Fränkel Set Theory) are axiomatised. Locales provide a setting for specifications within these logics. Compared to theories, locales are more restricted but provide more flexible means of reuse. The third kind of contexts are the contexts of Isabelle's proof language Isar. Currently, theories are the setting of numerous formalisations, which range from abstract, like lattice theory, to concrete, like models for Java-like languages. With the facilities of locales improving — for example, packages for the definition of inductive sets and recursive functions becoming available — it is to be expected that many of these developments will move to locales in future.

Interpretation of locales in locales is the topic of [3]. There we have argued that, in order to support interactive proof, a network of import and interpretation relations need to be maintained explicitly, so that newly proved theorems can be propagated. This can be achieved with development graphs [7]. In the present paper we discuss the extension of these mechanisms to theories and proofs.

Type constructors may only be declared in theories, not locales. Interpretation of locales in theories enables to map theorems valid relative to an abstract locale specification onto concrete types of a theory. For example, the type of natural numbers can be

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 31–43, 2006.

uniformly endowed with lattice theorems, which are proved in the context of a locale for lattices, for both the lattice induced by the order relation "$\leq$" and the lattice induced by the divisibility relation "$|$". This cannot be achieved with type classes, because there the operations are fixed [16].

Interpretation in proofs is more interesting. This is a true extension of the proof language. It can raise the level of abstraction in proofs, for it enables to reason at the level of concepts rather than theorems inside a proof itself. It is also an extension of the facilities of locales since it enables to employ locales for reasoning about arbitrary *families* of structures, say groups. The inheritance mechanisms of locales themselves only enable to combine specifications in a finitary way — for example, by importing the specification of groups twice to the specification of group homomorphisms.

2 Development Graphs

Development graphs were introduced by Hutter to manage dependencies between theories in verification settings where theories are repeatedly modified, and postulated relations between theories needs to be maintained — that is, formally proved, see [7]. Development graphs can reduce the number of proof obligations caused by such changes by carefully keeping track of dependencies in the development. The following exposition of development graphs follows [7], but takes into account that in locales they are also used to propagate proved theorems. This concept was introduced in [3].

Definition 1. *A* consequence relation *is a pair $(S, \vdash)$ where S is a set of sentences and $\vdash \subseteq \mathrm{Fin}(S) \times S$, where $\mathrm{Fin}(S)$ denotes the set of finite subsets of S, is a binary relation such that*

$$\{\phi\} \vdash \phi, \qquad \text{(reflexivity)}$$

$$\varDelta \vdash \phi \textit{ and } \{\phi\} \cup \varDelta' \vdash \psi \textit{ implies } \varDelta \cup \varDelta' \vdash \psi \textit{ and} \qquad \text{(transitivity)}$$

$$\varDelta \vdash \psi \textit{ implies } \{\phi\} \cup \varDelta \vdash \psi. \qquad \text{(weakening)}$$

A consequence relation induces a *closure operation* on sets of sentences $\varPhi \subseteq S$ defined by $\varPhi^{\vdash} = \{\phi \mid \varDelta \vdash \phi$ for some finite $\varDelta \subseteq \varPhi\}$. This is the set of all sentences *derivable* from $\varPhi$.

Definition 2. *A* morphism *of consequence relations from $(S_1, \vdash_1)$ to $(S_2, \vdash_2)$ is a function $\sigma : S_1 \to S_2$ such that $\varDelta \vdash_1 \phi$ implies $\sigma(\varDelta) \vdash_2 \sigma(\phi)$.*

Labelled directed graphs will be used to represent known dependencies between modules. A graph $G = (N, L)$ consists of nodes $n \in N$, which are module names, and links $n \xrightarrow{\sigma} m \in L$, which are labelled with consequence morphisms. There may be several links between from one node to another node, provided the morphisms are different.

As usual, reachability in graphs is defined along paths. The relation is enriched by the consequence morphism obtained from composing the labels along the path.

Definition 3. *Let $G = (N, L)$ be a labelled directed graph, where each link $n \xrightarrow{\sigma} m \in L$ is labelled with a consequence morphism σ. A node m is* reachable *from n via a consequence morphism σ, $n \xrightarrow{\sigma}_* m \in G$ for short, if $n = m$ and σ is the identity morphism* id, *or there is a $n \xrightarrow{\tau} k \in L$, and $k \xrightarrow{\rho}_* m \in G$, with $\sigma = \rho \circ \tau$.*

We will sometimes denote a graph by its set of links and write $n \xrightarrow{\sigma}_* m \in L$. The set of nodes will then be clear from the context. Likewise, we will write $n \xrightarrow{\sigma} m \in G$ instead of $n \xrightarrow{\sigma} m \in L$.

A development graph represents the import hierarchy of modules. Its links are called *definition links* and denote import relations.

Definition 4. *A* development graph *G is a finite labelled directed acyclic graph* (N, L)*. Each node* n *has an associated consequence relation* $(S_G(n), \vdash^n_G)$*, and finite sets of sentences* $\mathrm{A}_G(n), \mathrm{F}_G(n) \subseteq S_G(n)$*. For each node* $n \in N$*,* $\mathrm{A}_G(n)$ *is the set of* local axioms *of* n *and* $\mathrm{F}_G(n)$ *the set of* local proved theorems *in* n*. Links in L are called* definition links *and are denoted* $n \xrightarrow{\sigma}_{\mathrm{D}} m$*. The label* σ *is a morphism of the consequence relations from* $(S_G(n), \vdash^n_G)$ *to* $(S_G(m), \vdash^m_G)$*.*

The sets of proved theorems are not present in Hutter's definition. They are needed to model the propagation of theorems and the control information attached to them.

The proof theoretic semantics of a development graph is given by the sentences derivable at each module node.

Definition 5. *Let G be a development graph and n be a module node. The sets of* global axioms $\mathrm{A}^*_G(n)$ *and* global proved theorems $\mathrm{F}^*_G(n)$ *of n wrt. to G are defined by*

$$\mathrm{A}^*_G(n) = \bigcup_{k \xrightarrow{\sigma}_* n \in G} \sigma(\mathrm{A}_G(k)) \qquad \textit{and} \qquad \mathrm{F}^*_G(n) = \bigcup_{k \xrightarrow{\sigma}_* n \in G} \sigma(\mathrm{F}_G(k)).$$

The theory $\mathrm{Th}_G(n)$ *of* n *is the set of all sentences derivable from the global axioms;* $\mathrm{Th}_G(n) = \mathrm{A}^*_G(n)^{\vdash^n_G}$.

The theory of a node depends on its local axioms and of the axioms of imported nodes. Import is transitive. Proved theorems need to be derivable. We extend the definition of development graphs and demand that $\mathrm{F}_G(n) \subseteq \mathrm{Th}_G(n)$ for all $n \in N$ in a development graph (N, L). This implies that $\mathrm{F}^*_G(n) \subseteq \mathrm{Th}_G(n)$.

Interpretation relations between modules are consequences of a development graph. They are modelled by means of *theorem links*.

Definition 6. *Let* G *be a development graph and* n *and* m *be nodes in* G*. The graph implies a* global theorem link*, denoted* $G \vdash n \xrightarrow{\sigma}_{\mathrm{T}} m$*, if* $\mathrm{Th}_G(m) \vdash^m_G \sigma(\phi)$ *for all* $\phi \in \mathrm{Th}_G(n)$*. It implies a* local theorem link*, denoted* $G \vdash n \xrightarrow{\sigma}_{\mathrm{t}} m$*, if* $\mathrm{Th}_G(m) \vdash^m_G \sigma(\phi)$ *for all* $\phi \in \mathrm{A}_G(n)$*.*

Theorem links are properties of the development graph. We will use phrases like "a theorem link has been proven" or "established" to indicate that it is implied by the development graph under consideration.

Global theorem links require the image of the entire theory of the source node to be derivable. By transitivity of the consequence relation it is sufficient if the global axioms of the source node are derivable. Local theorem links only require the local axioms to be derivable. We observe that a development graph with definition link $n \xrightarrow{\sigma}_{\mathrm{D}} m$ implies the global theorem link $n \xrightarrow{\sigma}_{\mathrm{T}} m$. This, in turn, implies the local theorem link $n \xrightarrow{\sigma}_{\mathrm{t}} m$. The following lemma, which is due to Hutter, says how a global theorem link can be decomposed into a set of local theorem links.

Lemma 1. *Let G be a development graph. Then $G \vdash n \xrightarrow{\sigma}_{\mathrm{T}} m$ if and only if $G \vdash k \xrightarrow{\sigma \circ \tau}_{\mathrm{t}} m$ for all k and τ with $k \xrightarrow{\tau}_{*} n \in G$.*

When an interpretation — that is, a global theorem link — is asserted by the user, proof obligations are generated, and it is analysed which of these follow from import or existing interpretations. The lemma enables this analysis to be at the level of links, not sentences.

3 Isabelle: Theories, Proofs and Locales

Isabelle's meta-logic is based on an intuitionistic fragment of higher-order logic with polymorphic types. The polymorphism is schematic — that is, the type system is a quantifier-free first-order language. Terms of a special type *prop* are called *propositions*. These involve connectives for universal quantification and implication: $\bigwedge x.\, \phi$ and $\phi \Longrightarrow \psi$. *Theorems* are judgements of the form

$$\{\phi_1, \dots, \phi_n\} \vdash \phi,$$

where $\phi, \phi_1, \dots, \phi_n$ are propositions. The ϕ_i are called *meta-assumptions*. We abbreviate a judgement without meta-assumptions by $\vdash \phi$. In Isabelle and some of the related literature the judgements are written in reverse notation: $\phi\ [\phi_1, \dots, \phi_n]$. Isabelle's kernel contains functions that implement the rules of a natural deduction calculus on theorems; see [14] for details. We denote the resulting derivability relation by $\Vdash$. This is a consequence relation on the set of theorems. The rules of the calculus imply that also $\vdash$ is a consequence relation, on propositions.

The difference between both relations is subtle and rooted in the nature of schematic polymorphism. Let ϕ be a proposition containing a type variable α. One may obtain $\phi[\tau/\alpha]$ by substituting a type τ for α in ϕ. While $(\vdash\ \phi) \Vdash (\vdash\ \phi[\tau/\alpha])$ holds for theorems, the direct statement on propositions $\phi \vdash \phi[\tau/\alpha]$ does not hold in general. On the other hand, the canonical map from propositions to theorems, $\iota : \phi \mapsto (\vdash\ \phi)$, is a consequence morphism from $\vdash$ to $\Vdash$. Well-typed substitutions are consequence endomorphisms for both $\Vdash$ and $\vdash$.

3.1 Theories

Theories encompass declarations of language, namely constants and type constructors with their syntax, and theorems over that language. Theorems are either declared or derived, and the former are axioms of the theory. Theories may import other theories. They are not parametric, and import is literal. Constants and type constructors cannot be renamed. Instead, a system of qualified names resolves name conflicts by prefixing with the theory name, which must be unique. Theorems (including axioms) in theories have no meta-assumptions.

The import hierarchy of theories is a development graph $G = (N, L)$ where N is the set of *theory nodes*. The consequence relation of each node $n \in N$ is $\Vdash$; more precisely, the subrelation on theorems of the form $\vdash\ \phi$. All definition links in L are embeddings.

Theorems in theories may be named, and name spaces are also relevant to these names. In addition, theorems may be decorated with attributes. These encode hints that control the automatic use of theorems. For example, the attribute [simp] makes the theorem a rewrite rule automatically used by the simplifier, [simp del] removes a theorem from the rewrite rules, and [induct set: S] declares an induction rule for the inductively defined set S.

3.2 Proofs

Isar is Isabelle's language of human-readable formal proof documents. It was developed by Wenzel [17,18] and is radically different from tactic scripts. It is based on the principle that every object referred to in the proof text must have been introduced previously. It is not possible to refer to objects in the proof state that were generated by tactic applications, and whose meaning can only be understood by replaying the proof.

A full introduction to Isar is beyond the scope of this paper. Interested readers may consult the tutorial [11]. Here, we are only concerned with *proof contexts*. A theorem $\{\phi_1, \ldots, \phi_n\} \vdash \phi$ of the meta-logic can be seen as defining a context for the proposition ϕ. This context is given by parameters $x_1, \ldots, x_m$, which are the free variables occurring in the assumptions, and the assumptions $\phi_1, \ldots, \phi_n$ themselves. Isar refines this idea. Its proof contexts contain, in particular, proved local propositions as additional information, and decorate assumptions by discharge rules that are applied when leaving a context. Commands for the explicit construction of contexts within a proof are provided: **fix** x introduces a new parameter, which is discharged by $\bigwedge$-introduction, and **assume** ϕ introduces a new assumption with $\Longrightarrow$-introduction as discharge rule.

The following illustrate the main building blocks of Isar texts. The resulting propositions are displayed underneath. Note that these may depend on parameters and assumptions of surrounding contexts.

```
{
  fix $x$
  have $B\,x$ <proof>
}

$\bigwedge x.\, B\,x$
```

```
{
  assume $A$
  have $B$ <proof>
}

$A \Longrightarrow B$
```

The keyword **have** introduces a local goal and is followed by a proof. It yields a local proposition — more precisely, a theorem with meta-assumptions from the context. Local propositions may be named and have attributes. It is, for example, possible to declare local rewrite rules — that is, the proof context maintains, for example, a set of rewrite rules. A $\{\cdots\}$ block can be seen as a local kind of theory. Intermediate propositions may be proved and referred to in proofs. As already indicated, blocks may be nested, and they may be used to discharge complex goals involving $\bigwedge$ and $\Longrightarrow$. In such blocks **show** is used instead of **have** for the final inner goal. Isar then ensures that the block matches the proposition of the surrounding goal.[1]

[1] In fact, Isar is more liberal: higher-order unification takes place here.

3.3 Locales

Locales were designed by Kammüller [8,9] as a sectioning concept for tactic-based proofs. In Wenzel's reimplementation of locales for Isar [2] the focus has shifted, and there is now a stronger emphasis on locales as parametric theory modules, which are distinct from Isabelle's theories.

Locales may be seen as detached proof contexts with fixed sets of parameters and assumptions. These are identified by name. Assumptions are the module axioms. Isabelle's judgement operator $\vdash$ is a consequence relation. The set of theorems of the meta-logic is closed under substitution of types and terms for type and term variables. That is, well-typed substitutions are endomorphisms of the consequence relation. This enables to maintain locales in a structured way. A development graph $G = (N, L)$ represents the import relations between locales. (This is distinct from the graph of theories, but to simplify notation, we do not distinguish this for the moment.) The nodes $n \in N$ are *locale nodes*, and their consequence relations are $\vdash$ on the set of propositions. Locale parameters may have mixfix syntax. The list of parameters of locale n is denoted with $\mathrm{P}^*(n)$. Parameters are typed; $\mathrm{T}^*(n)$ denotes the type environment that maps the parameters of n to their types. Type variables occurring in $\mathrm{T}^*(n)$ are the *type parameters* of n.

Consistent with notation introduced in Section 2, $\mathrm{A}_G(n)$ denotes the local axioms (also called assumptions) and $\mathrm{F}_G(n)$ the local proved theorems (which are propositions) of locale n. The global assumptions are $\mathrm{A}^*_G(n)$. Interpretation relations between locales may be declared (and, of course, must be proved). A set of global theorem links T is maintained. The extended set of proved theorems of a locale

$$\mathrm{F}^*_{G,T}(n) = \bigcup_{k \xrightarrow{\sigma}_* n \in L \cup T} \sigma(\mathrm{F}_G(k)).$$

is obtained by accumulating proved theorems through any conceivable path of definition and theorem links. In [3] we have shown that this set is finite if consequence morphisms are restricted to renamings of term parameters.

4 Interpretation in Theories

Many formalisations in Isabelle are concerned with properties of objects that live in theories. In the object-logic Isabelle/HOL, in particular, many of these objects are types. Many share algebraic properties, and in order to facilitate reuse of theorems, axiomatic type classes were introduced to Isabelle [16]. These are restricted to algebraic structures with a single type parameter, and constants are shared, that is, overloaded. Locales permit more flexible forms of reuse since they may have an arbitrary number of parameters and type parameters. In addition, locales are useful in all object-logics while axiomatic classes are not.

In order to enable reuse of locale theorems in theories, we distinguish disjoint sets of theory nodes N_{T} and locale nodes N_{L} in the development graph. Likewise, the definition links in L_{T} are between theory nodes, and the links in L_{L} between locale nodes. The set T_{L} is a set of global theorem links implied by the locale development subgraph

$(N_{\mathrm{L}}, L_{\mathrm{L}})$. On the other hand, Isabelle's theory module does not support interpretations between theories, hence there are no theorem links in the theory development subgraph $(N_{\mathrm{T}}, L_{\mathrm{T}})$.

The canonical morphism $\iota : \phi \mapsto (\vdash \phi)$ enables to introduce theorem links from the locale development graph to the theory development graph. This is the set T_{T}, which is a set of local theorems links. The reason for this will become apparent in Section 4.2. The command

interpretation $l : n\ [p_1 \dots p_i]$ *<proof>*

declares a global theorem link from locale n to the current theory.[2] The arguments $p_1, \dots, p_i$ are terms over the language of the theory and determine the morphism of the theorem link: the kth parameter of n, which is the kth entry of $\mathrm{P}^*(n)$ is mapped to p_k. The substitution of type parameters is inferred. By means of Lemma 1 proof obligations are generated for local theorem links that are not implied by the development graph and existing theorem links. These are discharged by the user. Interpreted theorems are then added to the theory — but only those related to new local theorem links. This avoids unnecessary duplications. The label l is an optional theorem name prefix. It may be used to disambiguate theorems from different interpretations.

Maintaining development graph and theorem links is not only instrumental in discharging proof obligations. It also enables to propagate newly proved theorems of locales to theories that "subscribe" — that is, interpret — the locale.

4.1 Examples

A few examples are in order. The declarations below are of locales for partial orders, semi-lattices and linear orders; *partial_order* is imported by both *semi_lattice* and *linear_order*. The keyword **fixes** indicates parameters, **assumes** indicates axioms. Theorems may be added to these locales by a version of the **theorem** command.

```
locale partial_order =
  fixes le (infixl ⊑ 50)
  assumes refl: x ⊑ x
    and anti_sym: x ⊑ y ∧ y ⊑ x ⟹ x = y
    and trans: x ⊑ y ∧ y ⊑ z ⟹ x ⊑ z

locale semi_lattice = partial_order +
  fixes meet (infixl ⊓ 70)
  assumes le1: x ⊓ y ⊑ x and le2: x ⊓ y ⊑ y
    and least: x ⊑ y ∧ x ⊑ z ⟹ x ⊑ y ⊓ z

locale linear_order = partial_order +
  assumes linear: x ⊑ y ∨ y ⊑ x
```

[2] The implementation of the command permits a *locale expression* e in place of n. This is analogous to the interpretation of locales in locales with the command **interpretation** $m \subseteq e$ *<proof>*. See [3].

Table 1. Some interpretations of locales in theories

Locale	Morphism			Theory
semi_lattice	$\alpha \mapsto$ int	*le* $\mapsto$ $\mid$	*meet* $\mapsto$ gcd	Integer
semi_lattice	$\alpha \mapsto$ int	*le* $\mapsto$ $\leq$	*meet* $\mapsto$ min	Integer
linear_order	$\alpha \mapsto$ int	*le* $\mapsto$ $\leq$		Integer
semi_lattice	$\alpha \mapsto \alpha$ set	*le* $\mapsto$ $\subseteq$	*meet* $\mapsto$ $\cap$	Set
semi_lattice	$\alpha \mapsto \alpha$ set	*le* $\mapsto$ $\lambda xy.\, y \subseteq x$	*meet* $\mapsto$ $\cup$	Set
linear_order	$\alpha \mapsto$ unit set	*le* $\mapsto$ $\subseteq$		Set

Theorems of these locales may be reused in a theory *Integer* that declares a type *int* of integers. The upper half of Table 1 shows examples. The type is a semi-lattice via divisibility relation "|" and greatest common divisor. It is also a semi-lattice via "$\leq$" and minimum. Assume that these interpretations have been asserted and proved. Adding the interpretation that *int* is a linear order via "$\leq$" does not require to reprove that it is a partial order, since this information is stored in the set T_{T} of theorem links. Neither are the theorems of *partial_order* added to *Integer* in duplicate.

4.2 Subsumption

Lemma 1 is a general characterisation of the decomposition of global theorem links into local ones with respect to a development graph. The required notion of consequence morphism — that is, substitution — is now analysed further.

The lower half of Table 1 shows interpretations of the order and lattice locales in a theory *Set*, which introduces a type constructor *set* with the usual operations. Set inclusion "$\subseteq$" and intersection "$\cap$" form a semi-lattice. So does reversed set inclusion and union "$\cup$". Standard formalisations of anti-symmetric relations in Isabelle do not provide a connective for the reversed relation. This is also the case for set inclusion. A second connective "$\supseteq$" would increase redundancy without increasing expressiveness. Since Isabelle is a higher-order framework, the reverse inclusion can be specified by $\lambda xy.\, y \subseteq x$. Application of a consequence morphism is modulo β- and η-reduction — that is, λ-terms are normalised after applying the substitution.

The last line of the table interprets set of a particular type, while the other two interpretations map the parameter type to a polymorphic set type. The type *unit* is the type with a single element. This is trivially a linear order.[3] If the previous interpretations have been established before then partial_order $\xrightarrow{\alpha\mapsto\alpha\ \mathrm{set}}_{\mathrm{t}}$ Set $\in T_{\mathrm{T}}$. This obviously subsumes partial_order $\xrightarrow{\alpha\mapsto\mathrm{unit\ set}}_{\mathrm{t}}$ Set $\in T_{\mathrm{T}}$, which need not be reproved. On the other hand, the local theorem link linear_order $\xrightarrow{\alpha\mapsto\mathrm{unit\ set}}_{\mathrm{t}}$ Set $\in T_{\mathrm{T}}$, is not subsumed. For this reason, global theorem links are decomposed into local ones in T_{T}.

In summary, the calculus based on Lemma 1 that infers which local theorem links of a new interpretation are implied by existing ones is over higher-order terms, and subsumption testing is required. Higher-order unification is not decidable in general,

[3] The example is indeed not a particularly interesting one in terms of the interpretation. However, it serves well as an example of a subtype occurring in an interpretation. Examples of useful subtypes occur with axiomatic type classes.

but unification of higher-order patterns, which are λ-terms where the arguments of free variables occurring in the term are η-equivalent to bound variables. See [10]. This implies that restricting consequence morphisms to substitutions that replace term variables by higher-order patterns rather than arbitrary λ-terms ensures decidability of the calculus. We have not encountered examples in practice where full λ-terms would have been required.

5 Interpretation in Proofs

Interpretation raises the level of abstraction in formal developments. It is an operation at the level of concepts (mathematical theories, or — in our case — locales) rather than theorems. The idea is also found in paper proofs and follows this pattern: an object is constructed in some context, and properties of it are proved. It may turn out that the object is an instance of groups, say. It is then common to refer to theorems from that mathematical theory in the sequel of the proof, or to apply proof techniques from that theory.

This is a form of interpretation and is used by mathematicians informally — that is, without ever referring to it as interpretation. It is desirable to add this to the proof language, and Isar's proof contexts enable to do so.

5.1 Proofs as Development Graphs

Isar proofs are built in a top-down manner by continuous refinement of goals to subgoals until eventually a goals are solved. Each subgoal is reflected explicitly as a context in the proof text. The proof is a development graph $(N_{\mathrm{P}}, L_{\mathrm{P}})$, distinct from previous developments graphs for locales and theories, whose nodes are the nested contexts. The graph is in fact a tree, since it reflects the structure of the proof. Its root represents the goal statement. Definition links point from contexts of one level to the contexts of the next level. The consequence relation of the nodes is uniformly $\vdash$. The assumptions of the goal statement are the axioms of the root node. Along each path, contexts are only lever extended, by parameters and/or assumptions. The morphisms attached to the definition links are thus embeddings. Additional assumptions introduced in a context are the local axioms of that node. The local theorems are the propositions that are proved in the context.

Consider as an example the proof of the theorem that the subgroups of a group are a complete lattice. This has been formalised in Isabelle/HOL, and is available on the author's web page at http://www4.in.tum.de/~ballarin/isabelle/SubgrpLattice.thy. We will analyse a branch of the proof tree in more detail. The statement can be formalised as follows:

$$\bigwedge G.\, \mathrm{group}\, G \implies \mathrm{complete\_lattice}\, (\!|\mathrm{carrier} = \{H.\, \mathrm{subgroup}\, H\, G\}, \sqsubseteq\, = \mathrm{subgrp}\, G|\!) \tag{1}$$

Thick parentheses $(\!|\cdots|\!)$ denote records. The lattice $(\!|\mathrm{carrier} = \{H.\, \mathrm{subgroup}\, H\, G\}$, $\sqsubseteq\, = \mathrm{subgrp}\, G|\!)$ will be abbreviated by L in the sequel.

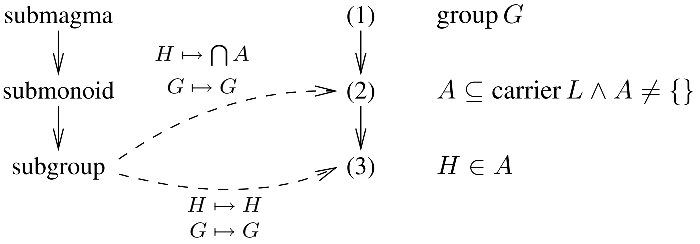


Fig. 1. Locale hierarchy and a proof that interprets subgroup locales

The context of this statement is given by parameter G and assumption group G. One proof of the statement involves showing the existence of a greatest lower bound (infimum) of each non-empty subset A of the carrier $\{H.\, \text{subgroup}\, H\ G\}$. The subgoal is:

$$\bigwedge A.\, A \subseteq \text{carrier}\, L \wedge A \neq \{\} \Longrightarrow \exists I.\, \text{greatest}\, L\ I\ (\text{Lower}\ L\ A) \qquad (2)$$

Note that this is relative to the context of the previous goal. The proof of this goal requires, amongst others, that the intersection of the groups in A is a subgroup of each $H \in A$:

$$\bigwedge H.\, H \in A \Longrightarrow \bigcap A \sqsubseteq_L H \qquad (3)$$

The path of the development graph corresponding to Subgoals (1) to (3) is depicted in Figure 1. To the right of the nodes, assumptions of the goals are displayed. In development graph terminology these are local axioms. Note that by Definition 5 they have to be read incrementally.

Completing the proof of (3) requires to show that $\bigcap A$ is a subgroup of H in G. This follows, because both $\bigcap A$ and H are subgroups of G. An Isar proof of this is reproduced in Figure 2(a). It is required to show that the set $\bigcap A$ is closed with respect to group operations of G. The proof is complicated considerably, because the notion of H being a subgroup of G is specified incrementally through the hierarchy of locales shown in Figure 1 on the left. It is a realistic assumption that an algebraic library is structured in that way.

5.2 Theorem Links

Interpretation enables to abstract away from these library structure details in the proof text. The command

interpret $l : n\ [p_1 \ldots p_i]$ *<proof>*

is analogous to the corresponding command for interpretation in theories, but is available in proofs.[4] It maintains a set of local theorem links T_{P} from the locale development subgraph to the proof development subgraph. Links are persistent in the scope of the proof only. They facilitate reuse of interpretations, for example if a hierarchy of locales is interpreted incrementally.

[4] The name is different because this is required, for technical reasons, by the front end of the proof assistant.

```
show $\bigcap A \sqsubseteq_L H$
  proof (simp, rule_tac subgrpI)
    show $H \subseteq$ carrier $G$
      by (rule submagma.subset [OF subgroup.is_submagma, OF subgroupH])
  next
    show $\bigwedge xy.\, x \in \bigcap A \wedge y \in \bigcap A \implies x \otimes y \in \bigcap A$
      by (rule submagma.m_closed [OF subgroup.is_submagma, OF Int_subgroup])
  next
    show $1 \in \bigcap A$
      by (rule submonoid.one_closed [OF subgroup.is_submonoid, OF Int_subgroup])
  next
    show $\bigwedge x.\, x \in \bigcap A \implies$ inv $x \in \bigcap A$
      by (rule subgroup.m_inv_closed [OF Int_subgroup])
  qed
```

(a) Proof without interpretation.

```
show $\bigcap A \sqsubseteq_L H$
  proof (simp, rule_tac subgrpI)
    show $H \subseteq$ carrier $G$ by (rule H.subset)
  next
    show $\bigwedge xy.\, x \in \bigcap A \wedge y \in \bigcap A \implies x \otimes y \in \bigcap A$ by (rule Int.m_closed)
  next
    show $1 \in \bigcap A$ by (rule Int.one_closed)
  next
    show $\bigwedge x.\, x \in \bigcap A \implies$ inv $x \in \bigcap A$ by (rule Int.m_inv_closed)
  qed
```

(b) Proof with interpretation.

```
show $\bigcap A \sqsubseteq_L H$ by (auto intro: subgrpI)
```

(c) Interpretation makes a more concise proof possible.

Fig. 2. Comparison of proofs without and with interpretation

In the proof described in the previous section, interpretation can be applied fruitfully twice.

In Proof Context (2): **interpret** Int: subgroup [$\bigcap A\ G$] <*proof*>
In Proof Context (3): **interpret** H: subgroup [$H\ G$] <*proof*>

The result is a simpler proof of Goal (3), which is shown in Figure 2(b). This is much cleaner, because manual interpretations of theorems by means of "[OF ...]" are no longer necessary. If, finally, the corresponding locale theorems were declared as rewrite rules with the attribute [simp] in the locale, the proof can be collapsed to a single line. See Figure 2(c).

6 Conclusion

The facilities for the interpretation of locales in theories and proofs described here have been implemented in Isabelle and are available with the release of Isabelle 2005. More information on the commands **interpretation** and **interpret** can be found in the corresponding Isar Reference Manual [15].

Interpretation is implemented in a hand full of provers, namely IMPS [6], PVS [13] and Coq [5], and development graphs are implemented in the development graph manager Maya [1]. In these systems only a single notion of context prevails, namely that of a theory. A comparison of the facilities of these systems and locales was given in our paper on interpretation of locales in locales [3].

The present work goes beyond that in that interpretation is extended to other contexts. Development graphs provide a framework making it possible to describe all three kinds of contexts available in Isabelle, namely theories, proofs and locales, in a uniform way. The result is a development graph where theory nodes, proof nodes and locale nodes are distinguished, and where theorem links, which represent interpretations, may only start from locale nodes but may point to any kind of node.

The distinction of theories and locales is rooted both in Isabelle being a framework for the specification of logics, and in types not being first-class citizens of higher-order logic. Interpretation of type constructors, which can be declared in theories but not in locales, would either require to apply context morphisms to proof objects, or to give up the concept of an LCF-style kernel. Given this distinction, the interpretation of locales in theories is useful. Maintaining theorem links explicitly makes it possible for theories to subscribe to locales, so that theorems become available there, whenever added to locales.

Interpretation in proof contexts is most exciting in combination with human-readable proofs. It enables proof writers to mimic a technique common in mathematical texts, namely to insert theorems from different mathematical theories in a proof as required. We have implemented interpretation in Isar proofs. Our experiment, a proof that the subgroups of a group form a complete lattice, shows that this can make proofs considerably more concise. The experiment also shows how locales can be applied to a family of objects, here the family of subgroups of a group, within a proof, while structured specifications cannot deal with that directly.

Consequence morphisms may map parameters to higher-order terms. By restricting these to higher-order patterns, the problem whether new interpretations are implied by existing ones can be shown to be decidable for interpretations in theories and proofs.

References

1. S. Autexier, D. Hutter, T. Mossakowski, and A. Schairer. The development graph manager Maya. In H. Kirchner and C. Ringeissen, editors, *Algebraic Methodology and Software Technology, AMAST 2002, Saint-Gilles-les-Bains, Reunion Island, France*, LNCS 2422, pages 495–501. Springer, 2002.
2. C. Ballarin. Locales and locale expressions in Isabelle/Isar. In Berardi et al. [4], pages 34–50.
3. C. Ballarin. Interpretation of locales in Isabelle: Managing dependencies between locales. Technical Report TUM-I0607, Technische Universität München, 2006.
4. S. Berardi, M. Coppo, and F. Damiani, editors. *Types for Proofs and Programs, TYPES 2003, Torino, Italy*, LNCS 3085. Springer, 2004.
5. J. Chrzaszcz. Modules in Coq are and will be correct. In Berardi et al. [4], pages 130–136.

6. W. M. Farmer, J. D. Guttman, and F. J. Thayer. Little theories. In D. Kapur, editor, *Automated deduction, CADE-11: Saratoga Springs, NY, USA*, LNCS 607, pages 567–581. Springer-Verlag, 1992.
7. D. Hutter. Management of change in structured verification. In *Automated Software Engineering, ASE 2000, Grenoble, France*, pages 23–31. IEEE Computer Society, 2000.
8. F. Kammüller. *Modular Reasoning in Isabelle*. PhD thesis, University of Cambridge, Computer Laboratory, Aug. 1999. Also Technical Report No. 470.
9. F. Kammüller, M. Wenzel, and L. C. Paulson. Locales: A sectioning concept for Isabelle. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *Theorem Proving in Higher Order Logics: TPHOLs'99, Nice, France*, LNCS 1690, pages 149–165. Springer, 1999.
10. T. Nipkow. Functional unification of higher-order patterns. In *Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 64–74, 1993.
11. T. Nipkow. Structured proofs in Isar/HOL. In H. Geuvers and F. Wiedijk, editors, *Types for Proofs and Programs (TYPES 2002)*, LNCS 2646, pages 259–278. Springer, 2003.
12. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. LNCS 2283. Springer, 2002.
13. S. Owre and N. Shankar. Theory interpretation in PVS. Technical Report CSL-01-01, SRI, Apr. 2001.
14. L. C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.
15. M. Wenzel. The Isabelle/Isar reference manual. Part of the Isabelle distribution, available at http://isabelle.in.tum.de.
16. M. Wenzel. Type classes and overloading in higher-order logic. In E. L. Gunter and A. Felty, editors, *Theorem proving in higher order logics: 10th International Conference, TPHOLs '97, Murray Hill, NJ, USA, August 19–22, 1997: proceedings*, LNCS 1275, pages 307–322. Springer, 1997.
17. M. Wenzel. *Isabelle/Isar — a versatile environment for human-readable formal proof documents*. PhD thesis, Technische Universität München, 2002. Internet publication, http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2002/wenzel.html.
18. M. Wenzel. Structured induction proofs in Isabelle/Isar. MKM, 2006.

A Dynamic Poincaré Principle

Manfred Kerber

School of Computer Science
The University of Birmingham
Birmingham B15 2TT
England
http://www.cs.bham.ac.uk/~mmk

Abstract. Proofs contain important mathematical knowledge and for mathematical knowledge management it is important to represent them adequately. They can be given at different levels of abstraction and writing a proof is typically a compromise between two extremes. On the one hand it should be in full detail so that it can be checked without using any intelligence, on the other hand it should be concise and informative. Making everything fully explicit is not adequate for most mathematical fields since easy parts do not need any communication. In particular in traditional proofs, computations are typically not made explicit, but a reader is expected to check them for him- or herself. Barendregt formulated a principle, the Poincaré Principle, which allows to separate reasoning and computation. However, should any computation be hidden? Or only easy computations? What is easy? How can we be sure that computations are correct? In this contribution, relevant notions are discussed and a principle is introduced which allows for checkable proofs which give a choice to see on request two different types of argument. The first type of argument states why any computation of this kind is correct. The second type states a (typically lengthy) detailed low-level proof of a trace of the computation.

1 Introduction

Proof is a multicoloured concept which plays an important role in mathematics at least back to the days of Aristotle (384BC–322BC), who can be considered as the father of logic, and Euclid (325BC–265BC), who advanced the field of geometry by achieving a rigour which was unattained before and was not surpassed in two thousand years.

Why are proofs so important in mathematics? Leibniz answered this question in 1686 as follows:

> *I've noticed that the reason why we get it so often wrong outside of mathematics and the [mathematicians] are so lucky with their conclusions, is only that in geometry and other fields of abstract mathematics it is possible to carry through . . . proofs, not only about the final statement, but also for each and every moment and for every step which is done starting from the premisses . . .*

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 44–53, 2006.

> *The only means to improve our conclusions is to make them as evident as those of the mathematicians are ... and when there is a dispute one needs only say: "Let's calculate."*
>
> *[12, p.16]*

Leibniz' idea was most influential in the rapid development of logic which started around 200 years later in the mid 1850ies by Boole in *An Investigation of The Laws of Thought* [4] and by others. However, it turned out later that there is an important difference between proof and computation. We will come back to that in the next section.

Frege distinguished the way how we detect a mathematical theorem and how we establish it by proof. He claims that the first is an individual process while the latter is best done in a more definite form through a logical proof.

> *In apprehending a scientific truth we pass, as a rule, through various degrees of certitude. Perhaps first conjectured on the basis of an insufficient number of particular cases, a general proposition comes to be more and more securely established by being connected with other truths through chains of inferences, whether consequences are derived from it that are confirmed in some other way or whether, conversely, it is seen to be a consequence of propositions already established. Hence we can inquire, on the one hand, how we have gradually arrived at a given proposition and, on the other, how we can finally provide it with the most secure foundation. The first question may have to be answered differently for different persons; the second is more definite, and the answer to it is connected with the inner nature of the propositions considered. The most reliable way of carrying out a proof, obviously, is to follow pure logic ... Everything necessary for a correct inference is expressed in full ... nothing is left to guesswork.*
>
> *[6] quoted from [8, p.6f].*

In this contribution, we are interested only in the question how to convey a proof and not how to find it in the first place. We will argue, however, that even this may be answered differently for different persons. While traditionally – using a passive medium such as paper – an author of a proof has to commit to a particular proof well-suited for a particular audience, the technological development makes it possible to adapt proofs and give choices to a reader as well.

Of course, there is a tension between "*Everything necessary for a correct inference is expressed in full ... nothing is left to guesswork*" and the need to be concise.

While rarely any mathematician, who is not very interested in the logical foundations of mathematics (and many mathematicians seem not to be too concerned about the foundations of mathematics), gives very precise proofs which are fully explicit, the arrival of proof development environments changed this picture. Suddenly it became possible (and necessary) to be fully precise in order to get a machine checkable proof and de Bruijn re-shaped Leibniz' old dream:

> *As a kind of dream I played (in 1968) with the idea of a future where every mathematician would have a machine on his desk, all for himself, on which he would write mathematics and which would verify his work. But, by lack of experience in such matters, I expected that such machines would be available in 5 years from then. But now, 23 years later, we are not that far yet.*
>
> *N.G. de Bruijn in [13, p.210]*

De Bruijn's idea can be viewed as a programme and manifesto for a whole field. As he says progress was slower than anticipated. This is partly due to the fact that things are in practice not as simple as Leibniz, Frege, Hilbert, and de Bruijn thought. In particular there is a need to be concise. One way to achieve this is to separate proof and computation. We will discuss this in the next section in more detail.

2 The Poincaré Principle

The invention of a tactic and tactical theorem proving was a great advance compared to proof checkers which do not use tactics, since tactics allow to reduce the number of user interactions in the generation of proofs. However, they do not reduce the length of the proof which is to be checked by a proof checker or by a human being, since each tactic has to be expanded to a low level proof which makes use only of calculus level rules.

If we expand a proof which is generated from a tactic, we typically do not get a proof which mathematicians would enjoy checking. There are proofs, which are concise and a pleasure to read, and there are proofs, which are lengthy, tedious and boring. Some theorems have proofs of the latter kind only (see [10]), however, whenever there is a shorter proof the advice of Hardy [7, p.29] would be very clear:

> *In both theorems [– the existence of an infinity of prime numbers and the irrationality of $\sqrt{2}$ –] (and in the theorems, of course, I include the proofs) there is a very high degree of* unexpectedness, *combined with* inevitability *and* economy. *The arguments take so odd and surprising a form; the weapons used seem so childishly simple when compared with the far-reaching results; but there is no escape from the conclusions. There are no complications of detail—one line of attack is enough in each case; and this is true too of the proofs of many much more difficult theorems, the full appreciation of which demands quite a high degree of technical proficiency. We do not want many 'variations' in the proof of a mathematical theorem: 'enumeration of cases', indeed, is one of the duller forms of mathematical argument. A mathematical proof should resemble a simple and clear-cut constellation, not a scattered cluster in the Milky Way.*

In one important aspect all these proofs are equivalent: They establish the theorem. However, in another they are not. Typically the more concise a proof is the more intelligence on part of the reader is required. How often is the "as easily can be seen" not so easy after all? Fully explicit proofs, however, require a lot of endurance. Human readers, in particular mathematically skilled ones, typically prefer the challenge over the boredom. This is different when we build computer systems which should check proofs. It is much easier to write systems which do the duller type of checking.

It is an important aspect of making proofs interesting to free them from computation by adding for each computable function f an expression $F(x)$ and postulate axiomatically that for arbitrary expressions t, $f(t)$ can be evaluated by applying F to the evaluation of t. Since the idea goes back to Poincaré it is called the Poincaré Principle in [2]. Barendregt and Cohen discuss it in detail in [3]:

> *Since computing is important for proving, one would like that if f is a computable function (on a freely generated algebra $\mathcal{A}$), then there is a formal expression $F(x)$ such that for all $a, b \in \mathcal{A}$*
>
> $$f(a) = b \quad \Leftrightarrow \quad \vdash F(\underline{a}) = \underline{b}, \tag{1}$$
>
> *for some representation $a \mapsto \underline{a}$ of elements of $\mathcal{A}$ in the theory.*
>
> *The most efficient way (from the point of proving) to ensure (1) is to add for each computable function f an expression $F(x)$ and postulate axiomatically that for arbitrary $a \in \mathcal{A}$*
>
> $$\vdash F(\underline{a}) = \underline{f(a)}. \tag{2}$$

The Poincaré Principle is in conflict with another principle which Barendregt and Cohen state as an important property of proof checkers, the de Bruijn principle [3]: "A proof assistant satisfies the *de Bruijn criterion* if it has a proof checker that is small enough to be verified by hand. Proof assistants that have proof objects that are stored have the advantage of the possibility of *independent checking*." The principle is important since a skeptical reader of a proof can take full control by checking the checker.

Let us assume that addition on natural numbers is defined inductively as $\forall n.\ n + 0 = n$ and $\forall n, m.\ n + s(m) = s(n + m)$ and we want to prove that $P(3 + 1)$ and $\neg P(4)$ are contradictory. There is first a representational issue. 1, 3, and 4 can be represented as successors of 0, that is, as $s(0)$, $s(s(s(0)))$, and $s(s(s(s(0))))$, respectively. Without a Poincaré principle a proof would be:

1.	$\forall n.\ n + 0 = n$	premise
2.	$\forall n, m.\ n + s(m) = s(n + m)$	premise
3.	$P(s(s(s(0))) + s(0))$	premise
4.	$\neg P(s(s(s(s(0)))))$	premise
5.	$\forall m.\ s(s(s(0))) + s(m) = s(s(s(s(0))) + m)$	$\forall$e$[n \mapsto s(s(s(0)))]$ 2
6.	$s(s(s(0))) + s(0) = s(s(s(s(0))) + 0)$	$\forall$e$[m \mapsto 0]$ 5
7.	$s(s(s(0))) + 0 = s(s(s(0)))$	$\forall$e$[n \mapsto s(s(s(0)))]$ 1
8.	$s(s(s(0))) + s(0) = s(s(s(s(0))))$	$=$subst $7, 6$
9.	$P(s(s(s(s(0)))))$	$=$subst $8, 3$
10.	$\bot$	$\neg$e $9, 4$

A similar logical level proof would have to be much longer if we wanted to make a similar argument for $P(1{,}000{,}000 + 1{,}000{,}000)$ and $\neg P(2{,}000{,}000)$. Such a trivial computation would take more than a million proof steps in a formal environment if one followed a naïve approach. This is not appropriate, in particular since it is easy to construct examples which are way beyond any computation power of a simple proof checker. This is not only extremely inconvenient for proof presentation, but can also be disastrous for proof construction. Since these steps are clearly unwanted in a proof, they should be excluded. Surely nobody would want to see a proof with more than a million steps, in particular, not for establishing a relationship which he or she is convinced to be true anyway. Typically nobody would want to see a proof with any sort of trivial computation included, since the trivial computations are taken for granted and are in some way outside of the argument. In order to make this formal, Barendregt has introduced the Poincaré principle, which allows to take computations out of the reasoning process and the proofs.

The argument above should be abbreviated to something like (indiscriminately of whether we compute $3 + 1$ or $1{,}000{,}000 + 1{,}000{,}000$):

1.	$P(3 + 1)$	premise
2.	$\neg P(4)$	premise
3.	$P(4)$	calculation 1
4.	$\bot$	$\neg$e $3, 2$

or even

1.	$P(3 + 1)$	premise
2.	$\neg P(4)$	premise
3.	$\bot$	$\neg\mathrm{e}_{\mathrm{calculation}}$ $1, 2$

The computations used in this example are very simple, but in real mathematics they may be much more advanced, for instance, involving the computations of integrals such $\int \sin^2(x)dx$, or checking a tautology by a decision procedure.

The Poincaré principle is important since no one wants to be bored by an uninformative proof, which contains almost no information. Furthermore it is appropriate since it takes steps out of the reasoning which human mathematicians would not consider part of a proof. Computations are assumed to be correct, algorithmic and checking them is on a different level than following a proof argument. That is, the Poincaré principle makes it possible to provide a guarantee for the correctness of a theorem at the *appropriate* level.

In a significant point, the Poincaré principle as presented in this section is ad hoc, however, since it assumes a fixed notion of what is "trivial" and what needs an explicit argument. While computations can be assumed to be trivial, in many aspects they are not always. While the sum of 1,000,000 and 1,000,000 is trivial for most people, the differentiation of $\frac{de^{\sin(\sqrt{x})}}{dx}$ may require some explanation, although computer algebra systems typically do not provide any.

Furthermore the notion what is "trivial" and what not is different for beginners and experts. Such a balanced view is, however, not possible assuming a fixed notion of what requires an argument and what not. In order to remedy this shortcoming of the Poincaré principle it is made dynamic in the next section by an explicit notion of reason for the validity of a computation.

3 A Dynamic Poincaré Principle

The Poincaré Principle is adequate for a fixed set of functional procedures for which the correctness can be checked once and for all. It can be postulated axiomatically to hold henceforth. In this section we propose a principle, called Dynamic Poincaré Principle, which is a generalization of the Poincaré Principle, since it allows a high-level explanation of proofs, in which inessential parts are omitted, but the computations are not postulated axiomatically, but can be checked to be correct by a skeptical reader in two fundamentally different ways.

The first expansion is a general argument as to why this type of step is correct, that is, in case of a computation, a proof that the computation algorithm is correct (in case of a tactic, a proof that the tactic is correct). The second expansion gives a detailed argument for the correctness of the particular argument, which – in case of a computation – corresponds directly to a detailed trace of the computation. Each of the arguments is sound and only one is needed. However, each amounts to a different type of understanding.

As Ayer pointed out [1, p.85f]

> *The power of logic and mathematics to surprise us depends, like their usefulness, on the limitations of our reason. A being whose intellect was infinitely powerful would take no interest in logic and mathematics. For he would be able to see at a glance everything that his definitions implied, and, accordingly could never learn anything from logical inference which he was not fully conscious of already. But our intellects are not of this order. It is only a minute proportion of the consequences of our definitions that we are able to detect at a glance. Even so simple a tautology as "$91 \times 79 = 7189$" is beyond the scope of our immediate apprehension. To assure ourselves that "7189" is synonymous with "91×79" we have to resort to calculation, which is simply a process of tautological transformation – that is, a process by which we change the form of expressions without altering their significance. The multiplication tables are rules for carrying out this process in arithmetic, just as the laws of logic are rules for the tautological transformation of sentences expressed in logical symbolism or in ordinary language.*

While for a being with an infinitely powerful intellect there is no need for any reasoning at all, the intellectual capabilities and experiences of individuals vary (in absolute terms and with particular mathematical fields). Somebody very familiar with a particular type of argument (such as mathematical induction) will understand many particular instances without any explanation, while a beginner will need a detailed argument.

This difference is true also for computational or algorithmic parts of a proof. Let us first look at a simple computation example in form of a division algorithm, **DIV**, which is supposed to compute the integer part of a simple division. Concretely, let us assume that two number `x` and `y` are given, the program is to calculate the quotient `a` and rest `b` of the division of `x` by `y`. If only the algorithm is given then it will produce a result but no argument which could be checked in order to convince us that the result is actually correct (you cannot ask a pocket calculator why 15 divided by 3 is 5). The argument why an algorithm is correct is typically on a level different from the logical level.[1]

The program can be given in any type of programming language such as Java for programs like integer division, or advanced specialized systems such as computer algebra systems for specialized computations like differentiation or indefinite integration. Note that in any of these cases it is important to state a semantic relationship between the function symbols in the reasoning system and the corresponding functions in the programming language. For instance, addition for natural numbers corresponds closely to addition in computer algebra systems, but not to addition in Java, which corresponds to addition in $\mathbf{Z}/p\mathbf{Z}$. Note, that for a rigorous argument, we need to have a correctness proof of the program.

One possible way to specify programs is given by Hoare logic through so-called Hoare triples. In the concrete example this can be done by proving the validity of

$$\{x \geq 0 \wedge y > 0\}\mathbf{DIV}\{a \cdot y + b = x \wedge 0 \leq b < y\}.$$

The Hoare calculus allows to prove properties of programs formally. If for instance **DIV** is given by

```
a := 0 ;
b := x ;
while b >= y do
    b := b - y;
    a := a + 1
  end
```

the correctness can be shown by a suitable loop invariant and the sequence rule. Note that even if we have a correctness proof, the correctness of the computation still depends on the correctness of the compiler with respect to the primitives used in the program (that is, here $+$, $-$, and $\geq$).

If we want to prove

Hyp $P(6/3)$ Hyp
Thm $P(2)$???

then there is mainly one standard argument. Namely we argue that we use a calculation to compute $6/3$.

[1] There are systems based on constructive logic (as, for instance, Nuprl [5] or Coq [9]) which allow to write internal algorithms which can be written and used in a verified way. While this is a particularly nice usage of the system for achieving correctness on different levels, conceptually the argument why an algorithm is correct, is a priori on a different level.

Hyp	$P(6/3)$	Hyp
Thm	$P(2)$	calculation

Since 'calculation' is not a primitive explanation, it must be possible to expand it (that is, to ask why it is correct). For this expansion there are now *two* standard explanations. First, we can present a correctness proof of the algorithm (for instance, in form of a Hoare calculus proof as discussed above). This option would be preferred if, for instance, we had to compute something like 1,000,000/2. Alternatively, we can present an argument why the concrete computation is correct, an argument which is relatively easy in the case of $6/3 = 2$. The latter can in principle be generated from a trace protocol of the computation.

Different proofs have different advantages. A high-level proof will capture the essential parts, it is more concise and may be easily generalized. On this level, the proof

Hyp	$P(6/3)$	Hyp
Thm	$P(2)$	calculation

is the same as the proof

Hyp	$P(1{,}000{,}000/2)$	Hyp
Thm	$P(500{,}000)$	calculation

while the explicit descriptions on a low logical level are quite different (for instance, they have significantly different lengths).

As described in [11], it is possible to decorate algorithms in a way that they produce tactics which explain concrete computations. This would be for a computation such as $6/3 = 2$ a tactic level proof:

Hyp	$P(6/3)$	Hyp
1	$6/3 = s((6-3)/3)$	DefDiv
3	$6/3 = s(s(((6-3)-3)/3))$	DefDiv
2	$6/3 = s(s((3-3)/3))$	DefMinus
4	$6/3 = s(s(0/3))$	DefMinus
5	$6/3 = s(s(0))$	DefDiv
Thm	$P(2)$	Subst=

This is not a logic level proof yet, since the tactics DefDiv and DefMinus may be further expanded.

The Dynamic Poincaré Principle frees up proofs from computations, not with a fixed a priori notion of what constitutes a computation, but in an extensible flexible way. It can be stated as follows. A proof follows the Dynamic Poincaré Principle if:

> *At any point in a proof it is possible to introduce an argument* "calculation", *which a reader can deal with in three different ways:*
>
> – *Believe that the computation is correct.*
> – *Check a proof that the algorithm performing the computation is correct.*
> – *Check that a trace of the concrete computation is correct.*

The Dynamic Poincaré Principle is an extension of the Poincaré Principle, since it adds flexibility for the writer of proofs by extending the notion of computation, and adds flexibility for the reader of proofs since it allows for skepticism in proof reading without enforcing it.

The Dynamic Poincaré Principle has been presented here to deal appropriately with computations in logical arguments. It should, however, be useful more broadly such as in the re-representation of mathematical expressions, for instance, in the transition from $((a+b)+((-b)+(-a)))+a = c$ to $a+b-b-a+a = c$ for an associative operator $+$.

4 Conclusion

We propose in this contribution a new principle, the Dynamic Poincaré Principle which allows to build and check proofs at an appropriate level. A main advantage of the Dynamic Poincaré Principle is that proofs can be represented in a concise form. Computations are not assumed to be axiomatically correct, but they can be expanded. A human reader of the proof can decide whether he or she believes the computation, wants to see an argument why the computation is always correct, or wants to get an expansion of a computation. An expansion of the computation has the advantage that it can be checked by standard proof checkers. This is, however, potentially at the price that the proof is prohibitively long. A general argument why the computation is correct requires still some trust that the actual computation has been performed correctly (for instance, that the compiler works perfectly and that the hardware did not fail). This seems for many types of computations acceptable.

References

1. Alfred Jules Ayer. *Language, Truth and Logic.* Victor Gollancz Ltd, London, UK, 2nd edition, 1951 edition, 1936.
2. Henk Barendregt. The impact of the lambda calculus in logic and computer science. *Bull. Symbolic Logic*, **3**(2):181–215, 1997.
3. Henk Barendregt and Arjeh M. Cohen. Electronic communication of mathematics and the interaction of computer algebra systems and proof assistants. *Journal of Symbolic Computation*, **32**(5):3–22, 2001.
4. George Boole. *An Investigation of The Laws of Thought.* Macmillan, Barclay, & Macmillan, Cambridge, UK; reprinted by Dover Publication, New York, USA, 1958, 1854.
5. Robert L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1986.
6. Gottlieb Frege. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle, 1879. reprint in: Begriffsschrift und andere Aufsätze, J. Angelelli, editor, Hildesheim. Also in Logiktexte, Karel Berka, Lothar Kreiser, editors, pages 82–112.
7. Godfrey Hardy. *A Mathematician's Apology.* Cambridge University Press, London, UK, 1940.

8. Jean van Heijenoort, editor. *From Frege to Gödel – A Source Book in Mathematical Logic, 1879-1931.* Havard University Press, Cambridge, Massachusetts, USA, 1967.
9. Gérard Huet et al. The Coq Theorem Prover (Version 8.0). `http://coq.inria.fr/doc-eng.html`.
10. Manfred Kerber. Why is the Lucas-Penrose argument invalid? In Uli Furbach, editor, *Proceedings of the 28th German Conference on Artificial Intelligence – KI 2005*, pages 380–393, Koblenz, Germany, 2005. Springer, LNAI 3698.
11. Manfred Kerber, Michael Kohlhase, and Volker Sorge. Integrating computer algebra into proof planning. *Journal of Automated Reasoning*, **21**(3):327–355, 1998.
12. Gottfried Wilhelm Leibniz. Projet et essais pour arriver à quelque certitude pour finir une bonne partie des disputes et pour avancer l'art d'inventer. In Karel Berka and Lothar Kreiser, editors, *Logiktexte*, chapter I.2, pages 16–18. Akademie-Verlag, german translation, 1983, Berlin, Germany, 1686.
13. Rob Nederpelt, Herman Geuvers, and Roel de Vrijer, editors. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, Amsterdam, The Netherlands, 1994.

A Proof-Theoretic Approach to Tactics

Kamal Aboul-Hosn

Department of Computer Science
Cornell University
Ithaca, New York, USA
kamal@cs.cornell.edu

Abstract. *Tactics* and *tacticals*, programs that represent and execute several steps of deduction, are fundamental to theorem provers providing automated tools for creating proofs quickly and easily. The language used for tactics is usually a full-scale programming language, separate from the language used to represent proofs. Consequently, there is also a separation between the use of theorems in proofs and the use of tactics. Our goal is to represent tactics in a way that allows them to be treated at the same level as proofs and theorems. We also want a representation that allows us to formally translate tactics into the proof steps they represent. We extend a system presented in [1,2] to represent tactics at the same level as theorems and move freely from tactics to proof steps and provide an example of its usefulness.

1 Introduction

Theorem provers such as Coq, NuPRL, and Isabelle provide extensive tools for users to create proofs quickly with automated methods. Fundamental to these systems is the use of *tactics* and *tacticals*, programs that represent and execute several steps of deduction. The language used for tactics is typically a full-scale programming language, separate from the language used to represent proofs. Consequently, there is also a separation between the use of theorems in proofs and the use of tactics.

Giunchiglia and Traverso succinctly state the properties desired for a tactic language: the tactics should be expressions of a logical language in order to facilitate reasoning about them; and, there should be a correspondence between the tactics as represented in this logical language and the programs that implement the tactics [3]. Syme calls the correspondence *justifications*, hints about the proof manually specified for the automated tactics [4].

The ML-like languages for tactics in Coq, NuPRL, and Isabelle are generally well suited to this task [5,6,7]. The tactics found in these systems are built from basic inference rules into complex programs that can apply rules, choose between tactics to apply, and analyze the current structure of a proof. These powerful constructs can automate a great deal of the theorem proving process.

Appel and Felty have looked at implementing tactics in higher-order logic programming languages such as Lambda Prolog and Twelf [8,9]. They use the

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 54–66, 2006.

power of backtracking in these systems to facilitate the creation and execution of tactics.

All of these approaches share one thing in common: they implement tactics at a system level that is separate from the level of proofs and theorems. The system-level implementation allows the tactics to be more powerful than the underlying logic. The power is particularly important in the automated search for proofs, where the user creates tactics and tacticals so that a theorem prover can complete a proof with little or no interaction. In fact, it is this desire for automation that drives the decision to separate tactics from proofs.

Despite being implemented at different levels, theorems and tactics have much in common. Both store and repeat proof steps. They represent generalized proving techniques used often within the theory in which they exist. Moreover, both provide guidance and hints to a user regarding the completion of a proof; proofs that share a few tactics or theorems are likely to share more. Nevertheless, work in the area of tactics and tacticals focuses on developing automated proof steps at the system level, separate from the underlying logic in which they work.

The power of the separate tactics language comes at a price, as explained by Delahaye [10]. Separating the two languages requires a user to learn two languages when creating proofs and the developer to create a separate infrastructure for debugging and validating tactics.

The separation between tactics and theorems also inhibits our flexibility in proof representation. While tactics may be used to automate proof steps, they are not represented in the completed proof; the tactics merely apply a sequence of elementary inference rules that a user would perform manually without the tactics. There are times when formally representing the tactics at the same level as proofs can be useful, particularly when transferring a proof to a paper. If a step in the proof is repeated several times by a tactic, we may want to perform the step explicitly the first time and then say that the step is "repeated several times in the same way." We want to be able to represent such a statement formally in the proof itself.

Our goal is to represent tactics in a way that allows them to be treated at the same formal level as proofs and theorems, independent of their system-level implementation. Many very useful tactics on commonly used algebras only require simple constructs that can be represented easily in the same way as theorems, not needing Turing-complete languages used in theorem provers. For example, a tactic for substitution of equals for equals require congruence rules for each operation in the algebra, the ability to iterate through several steps of using different congruence rules, and the ability choose the appropriate congruence rule at each step.

We also want a representation that allows us to easily translate tactics into the proof steps they represent using proof-theoretic, formal rules. Such a representation gives us the flexibility to make proofs more general by using the tactics in the representation or more specific by using some or all of the individual proofs steps.

Finally, the representation should be independent of search techniques and algorithms used to implement automated proof search. While these issues are important for a theorem prover, they are system-level decisions orthogonal to the choices made in representing tactics.

In this paper, we propose such a representation. We extend a system presented in [1,2] to represent tactics at the same level as theorems and move freely from tactics to proof steps. We formalize several common tactics and propose a way to represent them in our proof system. We then provide formal rules for creating and manipulating tactics and their use in proofs. Finally, we provide an extended example for creating a simple tactic and using it.

2 A Motivating Example

Consider reasoning about a Boolean algebra (B, $\vee$, $\wedge$, $\neg$, 0, 1). Boolean algebra is an equational theory, thus contains the axioms of equality:

$$\mathsf{ref} : \forall x. \quad x = x \tag{1}$$

$$\mathsf{sym} : \forall x, y. \quad x = y \ \rightarrow \ y = x \tag{2}$$

$$\mathsf{trans} : \forall x, y, z. \quad x = y \ \rightarrow \ y = z \ \rightarrow \ x = z \tag{3}$$

$$\mathsf{cong}_\wedge : \forall x, y, z. \quad x = y \ \rightarrow \ (z \wedge x) = (z \wedge y) \tag{4}$$

$$\mathsf{cong}_\vee : \forall x, y, z. \quad x = y \ \rightarrow \ (z \vee x) = (z \vee y) \tag{5}$$

$$\mathsf{cong}_\neg : \forall x, y, z. \quad x = y \ \rightarrow \ \neg x = \neg y \tag{6}$$

$$\mathsf{idemp}_\wedge : \forall x. \quad x \wedge x = x \tag{7}$$

Let us look at a particular form of tactic. It is easy to see that the axiom $\mathsf{idemp}_\wedge$ allows us to prove

$$\forall a. \quad a \wedge a \wedge a = a \tag{8}$$

Once we have the proof of (8), we can use it to prove

$$\forall a. \quad a \wedge a \wedge a \wedge a = a \tag{9}$$

in the following way. From (8) and $\mathsf{cong}_\wedge$ with the substitution $[x/a \wedge a \wedge a, y/a, z/a]$, we can deduce

$$a \wedge a \wedge a \wedge a = a \wedge a \tag{10}$$

We then use $\mathsf{idemp}_\wedge$ to get

$$a \wedge a = a \tag{11}$$

Finally, we apply trans to (10) and (11) with the substitution $[x/a \wedge a \wedge a \wedge a, y/a \wedge a, z/a]$ to conclude

$$a \wedge a \wedge a \wedge a = a$$

which is true for arbitrary a, yielding our desired conclusion (9). We can continue to prove a theorem like this for $n + 1$ occurrences of a using the proof for n occurrences of a.

The form of this proof is typical: an inductive argument where we use the result from one proof to prove a step in the next proof. We wish to generalize this kind of proof as a tactic that allows one to represent the execution of several steps of the proof either with the tactic itself or with the individual proof steps.

The need to recover the steps is important, particularly for presentation. Imagine one proves a theorem such as (9). Given that the proof steps are similar and repeated, one may wish to state the proof step explicitly once and then capture the rest of the iterations with one statement.

3 Tactic Representation

For representing tactics, we extend a proof representation system designed to create, remember, and reuse proofs from [1,2]. The papers present a *publish-cite* system, which uses proof rules with an explicit library to formalize the representation and reuse of theorems and lemmas. The system uses universal Horn equational logic, and we do as well, since it is a good vehicle for illustrating the organization and reuse of theorems. There is no inherent limitation in the system that requires the use of this logic; it could be extended to work with more complex deductive systems.

We build theorems from terms and equations. Consider a set of *individual variables* $X = \{x, y, \ldots\}$ and a first-order signature $\Sigma = \{f, g, \ldots\}$. An *individual term* $s, t, \ldots$ is either a variable $x \in X$ or an expression $ft_1 \ldots t_n$, where f is an n-ary function symbol in Σ and $t_1 \ldots t_n$ are individual terms. An equation $d, e, \ldots$ is between two individual terms, such as $s = t$.

A *theorem* is a universally quantified Horn formula of the form

$$\forall x_1, \ldots x_m.\quad \varphi_1 \;\rightarrow\; \varphi_2 \;\rightarrow\; \cdots \;\rightarrow\; \varphi_n \;\rightarrow\; \psi \tag{12}$$

where the φ_i are equations representing *premises*, ψ is an equation representing the conclusion, and $x_1 \ldots x_m$ are the variables that occur in the equations $\varphi_1, \ldots, \varphi_n, \psi$. A formula may have zero or more premises. These universally quantified formulas allow arbitrary specialization through term substitution. An example of this is the use of $\mathsf{cong}_\wedge$ with substitutions to get (10).

Next, we must define a proof term. For simplicity, we use the model presented in [1]. Let $\mathcal{P}$ be a set of *proof variables* $p, q, \ldots$. A proof of a theorem is a λ-term abstracted over both the proof variables for each premise of a theorem proven by the proof and the individual terms that appear in the proof. A *proof term* is:

- a variable $p \in \mathcal{P}$
- a constant, referring to the name of a theorem
- an application $\pi\tau$, where π and τ are proof terms
- an application πt, where π is a proof term and t is an individual term
- an abstraction $\lambda p.\tau$, where p is proof variable and τ is a proof term
- an abstraction $\lambda x.\tau$, where x is an individual variable and τ is a proof term

When creating proof terms, we have the typing rules seen in Table 1. These typing rules are what one would expect for a simply-typed λ-calculus. The

Table 1. Typing rules for proof terms

$$\frac{}{\Gamma, p : e \vdash p : e} \qquad \frac{}{\Gamma, c : \varphi \vdash c : \varphi}$$

$$\frac{\Gamma \vdash \pi : e \;\rightarrow\; \varphi \quad \Gamma \vdash \tau : e}{\Gamma \vdash \pi\ \tau : \varphi} \qquad \frac{\Gamma \vdash \pi : \forall x.\varphi}{\Gamma \vdash \pi\ t : \varphi[x/t]}$$

$$\frac{\Gamma, p : e \vdash \tau : \varphi}{\Gamma \vdash \lambda p.\tau : e \;\rightarrow\; \varphi} \qquad \frac{\Gamma \vdash \tau : \varphi}{\Gamma \vdash \lambda x.\tau : \forall x.\varphi}$$

typing environment Γ maps variables and constants to types. According to the Curry-Howard Isomorphism, the type of a well-typed λ-term corresponds to a theorem in constructive logic and the λ-term itself is the proof of that theorem [11]. For example, a theorem such as (12) viewed as a type would be realized by a proof term representing a function that takes an arbitrary substitution for the variables x_i and proofs of the premises φ_i and returns a proof of the conclusion ψ.

We use the following notation throughout the rest of the paper:

- $\overline{x}$ is a set of elements $\{x_1, \ldots, x_n\}$.
- $\varphi[\overline{x}/\overline{t}]$ means for all i, replace $x_i \in \overline{x}$ in φ with $t_i \in \overline{t}$.
- $\overline{\pi : \varphi}$ is the list of proof term typing statements $\pi_1 : \varphi_1, \ldots, \pi_n : \varphi_n$.

In order to use tactics, we introduce a few new proof terms:

- A case *statement*,

$$\begin{array}{ll} \mathsf{case}\ \delta\ \mathsf{of} & =\varphi_1 \Rightarrow \pi_1 \\ & \ldots \\ & =\varphi_n \Rightarrow \pi_n \\ & \psi_1 \Rightarrow \tau_1 \\ & \ldots \\ & \psi_m \Rightarrow \tau_m \end{array}$$

  where $\delta, \varphi_1, \ldots, \varphi_n, \psi_1, \ldots, \psi_m$ are formulas and $\pi_1, \ldots, \pi_n, \tau_1, \ldots, \tau_m$ are proof terms. The case statement is very similar to the one in Standard ML. We look at the structure of δ and match it against the types in the body of the statement. There are two kinds of matches that can occur. We can exactly match the type δ with a type φ_i, signified by the $=$, or we match a type δ against a possible unification, ψ_j. The difference is that a type δ matches a case $=\varphi_i$ if $\delta = \varphi_i$, whereas it matches a case ψ_j if there exists a substitution such that $\delta = \psi_j[\overline{x}/\overline{t}]$. The proof to the right of the $\Rightarrow$ of the matched case is a proof of the type δ, as enforced by the type system.

  We use the notation $\overline{=\varphi \Rightarrow \pi}$ to represent $=\varphi_1 \Rightarrow \pi_1 \ldots =\varphi_n \Rightarrow \pi_n$ and $\overline{\psi \Rightarrow \tau}$ to represent $\psi_1 \Rightarrow \tau_1 \ldots \psi_n \Rightarrow \tau_m$.

- A *formula variable* X, representing a quantified or unquantified formula
- A *formula abstraction* $\lambda X.\pi$, where X is a formula variable and π is a proof term. We need this proof term in order to abstract over the δ found in the case statement.

To support tactics, we extend formulas with *recursive types* [12],[13, Ch. 20]. We require the addition of three types:

- A *formula variable* X.
- A *recursive formula* $\mu X.\varphi$, where X is a formula variable and φ is a formula.
- A *sum formula* $\{\delta : \ =\varphi_1 + \ldots + =\varphi_n + \psi_1 + \ldots + \psi_m\}$ where δ,$\varphi_1, \ldots, \varphi_n$,$\psi_1, \ldots, \psi_m$ are formulas. We use the notation $\{\delta : \overline{=\varphi} + \overline{\psi}\}$ to represent $\{\delta : \ =\varphi_1 + \ldots + =\varphi_n + \psi_1 + \ldots + \psi_m\}$. The sum formula is closely related to the case statement, as will be apparent when examining the typing rules. In fact, we refer to an individual $=\varphi_i$ or ψ_j in a sum formula as a case.

The typing rules for the new proof terms are in Table 2. With the presence of abstraction over type variables, we need to type the formulas with kinds [13, Ch. 29]. The kinds primarily provide information for matching a formula with a case in a sum formula. Kinds are built from a base kind $*$ and the first-order signature $\Sigma = \{f, g, \ldots\}$. A *kind term* s_*, t_* is a *base kind* $*$ or an expression $f\ t_{1*} \ldots t_{n*}$ where f is an n-ary function symbol in Σ and $t_{1*}, \ldots, t_{n*}$ are kind terms. A kind equation d_*, e_* is between two kind terms, such as $s_* = t_*$.

For the most part, kind information is implicit; the kind $s_* = t_*$ of an equation $s = t$ is formed by replacing all variables in s and t with $*$. However, we may want to be explicit about kind information when the kind is more specific than the type. For example, the type $x = y$ implicitly has the kind $* = *$. If we mean for it to represent a more specific kind, say, $*\vee* = *\wedge*$ in our Boolean algebra example, we would have to specify the kind explicitly with the notation $(x = y : *\vee* = *\wedge*)$. A type's explicit kind can never be less specific than its implicit kind, i.e., $x\wedge y = y$ cannot have the kind $* = *$. We use the explicit kinds to match formulas with cases in the sum formula.

Table 2. Typing rules for new proof terms

$$\frac{\Gamma \vdash \pi_1 : \varphi_1 \quad \ldots \quad \Gamma \vdash \pi_n : \varphi_n \quad \Gamma \vdash \tau_1 : \psi_1 \quad \ldots \quad \Gamma \vdash \tau_m : \psi_m}{\Gamma \vdash \mathsf{case}\ X\ \mathsf{of}\ \overline{=\varphi \Rightarrow \pi}, \overline{\psi \Rightarrow \tau} : \{X : \ \overline{\varphi} + \overline{\psi}\}}$$

$$\frac{\Gamma \vdash \pi : \{\delta : \ \overline{=\varphi} + \overline{\psi}\}}{\Gamma \vdash \pi : \delta}\ \varphi_i = \delta$$

$$\frac{\Gamma \vdash \pi : \{\delta : \ \overline{=\varphi} + \overline{\psi}\}}{\Gamma \vdash \pi : \delta}\ \begin{array}{l}\psi_i[\overline{x}/\overline{t}] = \delta \text{ or} \\ \delta : e_*, \psi_i : e_*\end{array}$$

$$\frac{\Gamma \vdash \pi : \psi}{\Gamma \vdash \lambda X.\pi : \forall X.\psi} \qquad \frac{\Gamma \vdash \pi : \forall X.\psi}{\Gamma \vdash \pi\ \varphi : \psi[X/\varphi]}$$

$$\frac{\Gamma \vdash \lambda p.\pi : \mu X.\varphi}{\Gamma \vdash \pi[p/\lambda p.\pi] : \mu X.\varphi} \qquad \frac{\Gamma \vdash \pi[p/\lambda p.\pi] : \mu X.\varphi}{\Gamma \vdash \lambda p.\pi : \mu X.\varphi}$$

The type of a case statement with a formula variable X is the sum formula formed from the types of the proofs in the body of the statement. The second and third typing rules allow us to be more specific about a proof with a sum formula type. The type of a proof with a formula δ is δ if either δ is equal to one of the φ_i or δ unifies with or has the same kind as one of the ψ_i.

The type of the formula abstraction is the universal quantification over that formula. It is important to note that this is not the same as an abstraction over a proof variable p with the type φ. A term $\lambda p : \varphi.\pi$ would have the type $\varphi \rightarrow \psi$, where ψ is the type of π. When typing the application of a formula abstraction, the replacement of X with φ requires us to use the kind information. The only place such type variables appear is in case statements.

Finally, we have typing rules for proof terms with recursive types. The two typing rules correspond to unfolding and folding the proof term. We take an equi-recursive approach to the recursive types. In other words, $\mu X.\varphi$ is equivalent to $\varphi[X/\mu X.\varphi]$.

From the standpoint of an automated theorem prover, it is our type system that does most of the work of finding the correct steps to apply from a tactic. Most of this work is in choosing the correct case when applying a case statement to a type δ. Without any restrictions, δ may match several cases, requiring the type system to search though an exponential number of possible proofs. It is this search problem that makes implementing theorem prover tactics difficult. We regard the search problem as an implementation issue separate from the issue of formally representing tactics that we deal with in this paper. For the sake of this paper, we assume that when matching a type against possible cases in a case statement, we only explore the first match found, which removes the need for search at all.

We provide several rules for creating and manipulating proofs. The rules allow one to build proofs constructively. They manipulate a structure of the form $\mathcal{L};\mathcal{T}$, where

- $\mathcal{L}$ is the library of theorems, $T_1 = \pi_1, \ldots, T_n = \pi_n$
- $\mathcal{T}$ is a list of annotated *proof tasks* of the form $A \vdash \pi : \varphi$, where A is a list of assumptions, π is a proof term, and φ is a formula.

The proof rules can easily be extended to handle theorem scoping as in [2].

In Table 3, we present the rules for basic proof manipulation. The rules are very similar to the ones in [1]. One difference is that the **(ident)** and **(assume)** rules allow one to introduce assumptions with formula types and not just equations. We also add the **(inst)** rule, which allows us to instantiate variables over which a proof term is abstracted. Before, this was handled by the **(cite)** rule, but new rules give us the ability to have term abstractions in proof tasks, so we need to instantiate explicitly.

We also have **(norm$_t$)** and **(norm$_p$)** rules for performing β-reduction on applications of λ-abstractions over terms and proofs, respectively. It is important to note that the **(norm$_t$)** rule does not replace x in a proof in a case of a case

Table 3. Proof Rules for Basic Theorem Manipulation

$$\textbf{(assume)}\ \frac{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \tau : \psi}{\mathcal{L}\ ;\ \mathcal{T},\quad A, p : \varphi \vdash \tau : \psi}$$

$$\textbf{(ident)}\ \frac{\mathcal{L}\ ;\ \mathcal{T}}{\mathcal{L}\ ;\ \mathcal{T},\quad p : \varphi \vdash p : \varphi}$$

$$\textbf{(mp)}\ \frac{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \pi : \varphi\ \rightarrow\ \psi \quad A \vdash \tau : \varphi}{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \pi\ \tau : \psi}$$

$$\textbf{(discharge)}\ \frac{\mathcal{L}\ ;\ \mathcal{T},\quad A, p : e \vdash \tau : \psi}{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \lambda p.\tau : e\ \rightarrow\ \psi}$$

$$\textbf{(publish)}\ \frac{\mathcal{L}\qquad\qquad\qquad\qquad ;\ \mathcal{T},\quad \vdash \pi : \varphi}{\mathcal{L}, T = \lambda\overline{x}.\pi : \forall\overline{x}.\varphi\ ;\ \mathcal{T}}$$

$$\textbf{(cite)}\ \frac{\mathcal{L}_1, T = \pi : \varphi, \mathcal{L}_2\ ;\ \mathcal{T}}{\mathcal{L}_1, T = \pi : \varphi, \mathcal{L}_2\ ;\ \mathcal{T},\quad \vdash \pi : \varphi}$$

$$\textbf{(inst)}\ \frac{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \lambda x.\pi : \forall x.\varphi}{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \pi\ t : \varphi[x/t]}$$

$$\textbf{(norm}_t\textbf{)}\ \frac{\mathcal{L}\ ;\ \mathcal{T}\quad A \vdash (\lambda x.\pi)\ t}{\mathcal{L}\ ;\ \mathcal{T}\quad A \vdash \pi[x/t] : \varphi}$$

$$\textbf{(norm}_p\textbf{)}\ \frac{\mathcal{L}\ ;\ \mathcal{T}\quad A \vdash (\lambda p.\pi)\ \tau}{\mathcal{L}\ ;\ \mathcal{T}\quad A \vdash \pi[p/\tau] : \varphi}$$

$$\textbf{(forget)}\ \frac{\mathcal{L}_1, T = \pi : \varphi, \mathcal{L}_2\ ;\ \mathcal{T}}{\mathcal{L}_1, \mathcal{L}_2[T/\pi]\qquad ;\ \mathcal{T}[T/\pi]}$$

statement where we perform unification if x occurs in the type for that case. In other words, for the proof term

$$\mathsf{case}\ X\ \mathsf{of}\ \overline{=\varphi \Rightarrow \pi}, \overline{\psi \Rightarrow \tau} : \{X :\ \overline{=\varphi}\ +\ \overline{\psi}\}$$

we do not replace x in τ_i if it occurs in ψ_i. We do, however, replace x in any of the π_i and φ_i in which they occur. This behavior is not unlike the case statement in Standard-ML. The **(forget)** rule allows us to remove a theorem from the library. With the possibility of recursive proof terms, the **(forget)** rule must perform its replacement of T with π and normalization repeatedly until T no longer appears.

In Table 4, we introduce the proof rules to create, use, and manipulate theorems and tactics. The **(case)** rule combines existing proof tasks into a case statement. The types variable X can be unified with one of the types $\varphi_1, \ldots, \varphi_n$ or matched exactly with one of types of the assumptions $p_1, \ldots, p_m$. These types must be equations. The **(decase$^=$)** and **(decase)** allow us to determine which case the type δ matches and replace the case statement with the proof term for that specific case.

The rules **(fold)** and **(unfold)** are standard rules one would expect for dealing with recursive types. The **(publish^r)** rule allows us to publish recursive

Table 4. Proof Rules for Tactics

$$\textbf{(case)}\quad \frac{\mathcal{L}\ ;\ \mathcal{T},\quad A,\overline{p:e} \vdash \pi_1 : \varphi_1 \quad \ldots \quad A,\overline{p:e} \vdash \pi_n : \varphi_n}{\mathcal{L}\ ;\ \mathcal{T},\quad \vdash \mathsf{case}\ X\ \mathsf{of}\ \overline{=e \Rightarrow p}, \overline{\varphi \Rightarrow \pi} : \{X :\ \overline{=e}\ +\ \overline{\varphi}\}}$$

$$\textbf{(decase}^{=}\textbf{)}\quad \frac{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \mathsf{case}\ \delta\ \mathsf{of}\ \overline{=\varphi \Rightarrow \pi}, \overline{\psi \Rightarrow \tau} : \delta}{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \pi_i : \delta}\ \varphi_i = \delta$$

$$\textbf{(decase)}\quad \frac{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \mathsf{case}\ \delta\ \mathsf{of}\ \overline{=\varphi \Rightarrow \pi}, \overline{\psi \Rightarrow \tau} : \delta}{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \tau_i[\overline{x}/\overline{t}] : \delta}\ \psi_i[\overline{x}/\overline{t}] = \delta$$

$$\textbf{(fold)}\quad \frac{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \pi[p/\lambda p.\pi] : \mu X.\varphi}{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \lambda p.\pi : \mu X.\varphi}$$

$$\textbf{(unfold)}\quad \frac{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \lambda p.\pi : \mu X.\varphi}{\mathcal{L}\ ;\ \mathcal{T},\quad A \vdash \pi[p/\lambda p.\pi] : \mu X.\varphi}$$

$$\textbf{(publish}^{r}\textbf{)}\quad \frac{\mathcal{L}\qquad\qquad\qquad\qquad ;\ \mathcal{T},\quad p : \mu X.\psi \vdash \pi : \varphi}{\mathcal{L}, p = \lambda\overline{x}.\lambda p.\pi : \forall\overline{x}.\mu X\varphi\ ;\ \mathcal{T}}\ \mu X.\psi = \forall\overline{x}.\mu X\varphi$$

$$\textbf{(forget}_1\textbf{)}\quad \frac{\mathcal{L}_1, T = \pi : \varphi, \mathcal{L}_2\ ;\ \mathcal{T},\quad A \vdash T\ \tau : \psi}{\mathcal{L}_1, T = \pi : \varphi, \mathcal{L}_2\ ;\ \mathcal{T},\quad A \vdash \pi\ \tau : \psi}$$

$$\textbf{(norm}_f\textbf{)}\quad \frac{\mathcal{L}\ ;\ \mathcal{T}\quad A \vdash (\lambda X.\pi)\ \psi}{\mathcal{L}\ ;\ \mathcal{T}\quad A \vdash \pi[X/\psi] : \varphi}$$

proof terms. In other words, these are tactics that use themselves in the proof. Recursion of this nature is very important for tactics; we want to be able to repeat proof steps several times, such as in our example in Section 2. The rule takes a proof task with a single assumption of a recursive type and moves it to the library. The name assigned to the theorem is the same as the proof variable in the assumption. It is also necessary that the type of the proof variable and the type of the proof term added to the library are equivalent.

We add the **(forget$_1$)** rule, which functions much like **(forget)**, except we replace a theorem name with the proof of that theorem in only a single application in a single proof task and we do not remove the theorem from the library. This rule allows us to make explicit one step in the application of a tactic. Finally, the **(norm$_f$)** rule performs β-reduction on applications of λ-abstractions over formulas.

The steps in creating a tactic with several cases that recursively call the tactic would be as follows:

1. Use the **(assume)** and **(ident)** rules to add a proof variable with the type of the tactic to be created.
2. Create the proof terms for the cases of the tactic, using the assumption added in step 1 for the recursive calls.
3. Use the **(case)** rule to combine the proof terms created in step 2 into a single case statement.
4. Use the **(publish^r)** rule to publish the new tactic.

4 A Constructive Example

We can provide a tactic for our example in Section 2. First, we give a general description of the proof steps in our tactic. For a given x and a, if we want to prove $x \wedge a = a$, we use a recursive tactic that is quantified over an equation Y. If Y is of the form $x = x$, then we use ref to prove the equation true. If Y is of the form $x \wedge a = a$, then it suffices to apply trans to proofs of $x \wedge a = a \wedge a$ and $a \wedge a = a$. The latter follows directly from $\mathsf{idemp}_\wedge$. For the former, we use $\mathsf{cong}_\wedge$ on a proof of $x = a$, which we obtain by recursively calling the tactic.

Let

$$\varphi_R \quad = \quad \mu X.\forall x.\forall a.\forall Y. \quad X \;\rightarrow\; \{Y : x = x \;+\; x \wedge a = a\}$$

First, we use **(ident)** to create a proof task

$$R : \varphi_R \vdash R : \varphi_R \tag{13}$$

Next, let us create the cases of our tactic. We first create what will be the "base case" for our recursion. We use **(cite)**, **(inst)**, and **(assume)** to get the proof task

$$R : \varphi_R \vdash \mathsf{ref}\ x : x = x \tag{14}$$

For the recursive case, we use **(inst)** on (13) and the fact that we use equi-recursive types to get

$$\begin{aligned} R : \varphi_R \vdash R\ x\ a\ (x = a : {*}\wedge{*} = {*}) \\ : \varphi_R \;\rightarrow\; \{(x = a : {*}\wedge{*} = {*}) :\; x = x \;+\; x \wedge a = a\} \end{aligned} \tag{15}$$

We have made the kind of $x = a$ explicit in order to make sure it matches the $x \wedge a = a$ case in our sum formula type in φ_R. Next, we use **(mp)** on (15) and (13) to get

$$R : \varphi_R \vdash R\ x\ a\ (x = a : {*}\wedge{*} = {*})\ R : (x = a : {*}\wedge{*} = {*}) \tag{16}$$

For the rest of the example, we do not show the kind of $x = a$ for readability. To use congruence of $\wedge$, we use **(cite)**, **(inst)**, and **(assume)** to add the task

$$R : \varphi_R \vdash \mathsf{cong}_\wedge\ x\ a\ a : x = a \;\rightarrow\; x \wedge a = a \wedge a \tag{17}$$

We combine (17) and (16) using **(mp)** to get

$$R : \varphi_R \vdash \mathsf{cong}_\wedge\ x\ a\ a\ (R\ x\ a\ (x = a)\ R) : x \wedge a = a \wedge a \tag{18}$$

For the proof of $a \wedge a = a$, we use **(cite)**, **(inst)**, and **(assume)** to add the proof task

$$R : \varphi_R \vdash \mathsf{idemp}_\wedge\ a : a \wedge a = a \tag{19}$$

We introduce transitivity with **(cite)**, **(inst)**, and **(assume)**

$$R : \varphi_R \vdash \mathsf{trans}\ (x{\wedge}a)\ (a{\wedge}a)\ a : \begin{aligned}[t] & x{\wedge}a = a{\wedge}a \\ \rightarrow\ & a{\wedge}a = a \\ \rightarrow\ & x{\wedge}a = a \end{aligned} \tag{20}$$

Two applications of **(mp)** with (20),(18), and (19) give the completed recursive case for our tactic:

$$R : \varphi_R \vdash \mathsf{trans}\ \begin{aligned}[t] & (x{\wedge}a)\ (a{\wedge}a)\ a \\ & (\mathsf{cong}_{\wedge}\ x\ a\ a\ (R\ x\ a\ (x = a)\ R)) \\ & (\mathsf{idemp}_{\wedge}\ a) \end{aligned} \quad : x{\wedge}a = a \tag{21}$$

Now we use the **(case)** rule to combine (14) and (21) for our tactic:

$$\begin{aligned}[t] R : \varphi_R \vdash\ & \mathsf{case}\ Y\ \mathsf{of} & : \{Y :\ x = x\ +\ x{\wedge}a = a\} \\ & (x = x) \Rightarrow \mathsf{ref}\ x & \\ & (x{\wedge}a = a) \Rightarrow \mathsf{trans}\ \begin{aligned}[t] & (x{\wedge}a)\ (a{\wedge}a)\ a \\ & (\mathsf{cong}_{\wedge}\ x\ a\ a \\ & \quad (R\ x\ a\ (x = a)\ R)) \\ & (\mathsf{idemp}_{\wedge}\ a) \end{aligned} & \end{aligned}$$

Finally, we use the **(publish^r)** rule to publish the tactic as

$$\begin{aligned}[t] R = \lambda x.\lambda a.\lambda Y.\lambda R.\ & \mathsf{case}\ Y\ \mathsf{of} \\ & (x = x) \Rightarrow \mathsf{ref}\ x \\ & (x{\wedge}a = a) \Rightarrow \mathsf{trans}\ \begin{aligned}[t] & (x{\wedge}a)\ (a{\wedge}a)\ a \\ & (\mathsf{cong}_{\wedge}\ x\ a\ a\ (R\ x\ a\ (x = a)\ R)) \\ & (\mathsf{idemp}_{\wedge}\ a) \end{aligned} \end{aligned}$$

The type of this tactic is

$$\forall x.\forall a.\forall Y.\quad \varphi_R\ \rightarrow\ \{Y :\ x = x\ +\ x{\wedge}a = a\}$$

Notice that $\forall x.\forall a.\forall Y.\quad \varphi_R\ \rightarrow\ \{Y : x = x\ +\ x{\wedge}a = a\}$ is equal to φ_R, which is necessary for applying the rule.

We now have a tactic that given an x of the form $a{\wedge}\ldots{\wedge}a$ will provide a proof of $a{\wedge}\ldots{\wedge}a = a$. If applied to a term that is not of this form, the tactic will not have a type.

We can now apply the tactic to create a new proof. We use the **(cite)** and **(inst)** rules just as we do on theorems to create the proof task

$$\vdash R\ (b{\wedge}b{\wedge}b)\ b\ (b{\wedge}b{\wedge}b{\wedge}b = b) : \varphi_R\ \rightarrow\ b{\wedge}b{\wedge}b{\wedge}b = b$$

We then use **(cite)** and **(mp)** to get the conclusion we desire.

$$\vdash R\ (b{\wedge}b{\wedge}b)\ b\ (b{\wedge}b{\wedge}b{\wedge}b = b)\ R : b{\wedge}b{\wedge}b{\wedge}b = b \tag{22}$$

We may want to make one step of the application of the tactic R explicit. First, we use the **(forget$_1$)** rule on (22) to replace the name of the tactic with its body and then use the normalize rules to perform β-reduction to get

$$\begin{array}{ll}\vdash \mathsf{case}\ (b{\wedge}b{\wedge}b{\wedge}b = b)\ \mathsf{of} & : b{\wedge}b{\wedge}b{\wedge}b = b \quad (23)\\ \quad (x = x) \Rightarrow \mathsf{ref}\ x & \\ \quad (x{\wedge}a = a) \Rightarrow \mathsf{trans}\ \ (x{\wedge}a)\ (a{\wedge}a)\ a & \\ \qquad\qquad\qquad\qquad (\mathsf{cong}_{\wedge}\ x\ a\ a\ (R\ x\ a\ (x = a)\ R)) & \\ \qquad\qquad\qquad\qquad (\mathsf{idemp}_{\wedge}\ a) & \end{array}$$

We can then use **(decase)** to replace the case statement with the specific case that is matched, where $b{\wedge}b{\wedge}b{\wedge}b = b$ unifies with $x{\wedge}a = a$ under the substitution $[x/b{\wedge}b{\wedge}b, a/b]$.

$$\begin{array}{ll}\vdash \mathsf{trans}\ \ (b{\wedge}b{\wedge}b{\wedge}b)\ (b{\wedge}b)\ b & : b{\wedge}b{\wedge}b{\wedge}b = b \quad (24)\\ \qquad\quad (\mathsf{cong}_{\wedge}\ (b{\wedge}b{\wedge}b)\ b\ b & \\ \qquad\quad\ \ (R\ (b{\wedge}b{\wedge}b)\ b\ (b{\wedge}b{\wedge}b = b)\ R)) & \\ \qquad\quad (\mathsf{idemp}_{\wedge}\ b) & \end{array}$$

Now one of the steps of the proof is explicit while the others are implicitly captured in the application of the tactic R.

5 Conclusion

We have presented a proof-theoretic approach in which tactics are treated at the same level as theorems and proofs. The proof rules allow us to create, manipulate, and apply tactics in a way that is completely formal and independent of system-level decisions regarding proof search. Many important tactics can be represented in the relatively simple system we have demonstrated, particularly in algebras such as our Boolean example.

Representing tactics at this level has several advantages for automated theorem provers, from both a user perspective and a developer perspective. For users, powerful tactics can be created without needing to learn a separate tactics language. However, the power of the language used to implement the theorem prover can be harnessed to make proof search as complete and efficient as desired. Additionally, when combined with the work in [2], tactics can be put into a local scope and abstractions can be manipulated just as we can with theorems, a powerful ability lacking from current theorem provers.

In the future, we plan to have a full implementation of tactics added to the Java implementation mentioned in [2] for Kleene algebra with tests [14]. We can then investigate the ability of the system to discover repeated citations in proofs that can be abstracted out at lemmas. Given the structure of tactics and theorems, detecting similar subproofs is a form of common subexpression elimination, a process we call *proof refactorization.* With this system, one could easily use tactics with a strong underlying formalism guiding their manipulation.

Acknowledgements

We would like to thank Dexter Kozen and Sigmund Cherem for valuable ideas and comments. This paper is dedicated in loving memory to the author's grandfather, Orlen F. Rice, who passed away during its writing.

References

1. Kozen, D., Ramanarayanan, G.: A proof-theoretic approach to knowledge acquisition. Technical Report 2005-1985, Computer Science Department, Cornell University (2005)
2. Aboul-Hosn, K., Damhøj Andersen, T.: A proof-theoretic approach to hierarchical math library organization. In Kohlhase, M., ed.: MKM. Volume 3863 of Lecture Notes in Computer Science., Springer (2005) 1–16
3. Giunchiglia, F., Traverso, P.: Program tactics and logic tactics. Annals of Mathematics and Artificial Intelligence **17**(3-4) (1996) 235–259
4. Syme, D.: Three tactic theorem proving. In: TPHOLs '99: Proceedings of the 12th International Conference on Theorem Proving in Higher Order Logics, London, UK, Springer-Verlag (1999) 203–220
5. Kreitz, C.: The Nuprl Proof Development System, Version 5: Reference Manual and User's Guide. Department of Computer Science, Cornell University. (2002)
6. The Coq Development Team: The Coq Proof Assistant Reference Manual – Version V7.3. (2002) `http://coq.inria.fr`.
7. Wenzel, M., Berghofer, S.: The Isabelle System Manual. (2003)
8. Felty, A.: Implementing tactics and tacticals in a higher-order logic programming language. Journal of Automated Reasoning **11**(1) (1993) 43–81
9. Appel, A.W., Felty, A.P.: Dependent types ensure partial correctness of theorem provers. J. Funct. Program. **14**(1) (2004) 3–19
10. Delahaye, D.: A tactic language for the system Coq. In Parigot, M., Voronkov, A., eds.: LPAR. Volume 1955 of Lecture Notes in Computer Science., Springer (2000) 85–95
11. Sørensen, M.H., Urzyczyn, P.: Lectures on the Curry–Howard isomorphism. Available as DIKU Rapport 98/14 (1998)
12. Morris, J.H.: Lambda calculus models of programming languages. Technical report, Massachuseets Instititue of Technology, Laboratory for Computer Science (1968)
13. Pierce, B.C.: Types and Programming Languages. MIT Press (2002)
14. Kozen, D.: Kleene algebra with tests. Transactions on Programming Languages and Systems **19**(3) (1997) 427–443

A Formal Correspondence Between OMDoc with Alternative Proofs and the $\overline{\lambda}\mu\tilde{\mu}$-Calculus

Serge Autexier[1] and Claudio Sacerdoti-Coen[2]

[1] DFKI GmbH and Saarland University, Saarbrücken, Germany
autexier@dfki.de
[2] Department of Computer Science, University of Bologna, Italy
sacerdot@cs.unibo.it

Abstract. We consider an extension of OMDoc proofs with alternative sub-proofs and proofs at different level of detail, and an affine non-deterministic fragment of the $\overline{\lambda}\mu\tilde{\mu}$-calculus seen as a proof format. We provide explanations in pseudo-natural language of proofs in both formats, and a formal correspondence between the two by means of two mutually inverse encodings of one format in the other one.

1 Introduction

Proofs play a major role in mathematics and their representation is a key issue in mathematical knowledge management. Proofs of all kinds need to be stored and retrieved in repositories of formalized mathematics and communicated across system and logic boundaries, for instance to be assembled to larger proofs in the context of computer assisted mathematical theorem proving or to be explained on an adaptive level of granularity in tutor systems for teaching mathematics.

In [1] the first author and his colleagues presented a data structure for the representation of proof attempts (proof data structure or *PDS*). The two main features of a PDS are the possibility of representing proofs at different levels of granularity and that of representing alternative, possibly incomplete, subproofs. A PDS is quite complex, being a directed graph with nodes representing proof goals, arcs justifying a proof goal with some subgoals via a calculus rule, high-level proof method or a proof sketch and *hierarchical arcs* that represent transitions between granularity levels. There can be more than one justification for each node, which are at different levels of granularity if connected by hierarchical arcs and alternative subproofs otherwise. A PDS maintains simultaneously all proofs at different levels of granularity as well as all alternative proofs of some subgoals, which are useful features during interactive or automatic proof construction, and also for proof explanation, for instance in a tutorial setting. Selection of a specific level of granularity is supported by *views* on a PDS. In this paper we consider *views* to also include the selection of exactly one alternative subproof for each goal. To store and communicate the proofs and proof plans contained in a PDS, a PDS can be serialized to a proof format, that can be OMDoc [3]. However, OMDoc is currently unable to represent alternative subproofs.

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 67–81, 2006.

Both PDSs and OMDoc documents claim to be adequate representation formalisms for mathematical proofs. However, they lack a semantics specification and are so generic that any structured document can be embedded into them. To make more precise the semantics of OMDoc and other proof formats, the second author has started the investigation of the $\overline{\lambda}\mu\tilde{\mu}$ calculus as a proof format in [4]. The calculus, that is Curry-Howard isomorphic to classical sequent calculus, has very remarkable properties per se and also as a proof format. In particular, as explained in [4], its intuitionistic (and deterministic) fragment can be equipped with a straightforward translation to pseudo-natural language, and proofs in the classical fragment can be easily translated to the intuitionistic fragment. What was so far unclear is whether the non-deterministic classical fragment of the calculus (or sub-fragments of it) is necessary to fully exploit the calculus as a proof format. In this paper we answer positively the question by proposing an extension of OMDoc with alternative proofs (inspired by PDSs) and an encoding of the extension in a fragment of the $\overline{\lambda}\mu\tilde{\mu}$-calculus. The encoding provides naturally a semantics for a view: a view on a proof is obtained by reducing the corresponding proof term according to the non-deterministic rules of the calculus. The non determinism dynamically selects just one of the alternative proofs for each choice. The encoding is particularly informative for two reasons. First of all it provides a clear semantics for OMDoc (and, indirectly, for the corresponding PDSs). Secondly it tries to respect the *rendering semantics* associated to OMDoc and to the $\overline{\lambda}\mu\tilde{\mu}$-calculus. A rendering semantics is the function that translates the term to its pseudo-natural language rendering.

2 $\overline{\lambda}\mu\tilde{\mu}$-Calculus

Table 1 shows the syntax of the $\overline{\lambda}\mu\tilde{\mu}$-calculus, proposed by Curien and Herbelin in [2]. Its rendering semantics can be found in [4]. In the rest of the paper we assume the reader to be familiar with the latter paper, while knowledge of the first one is not necessary.

The *intuitionistic* fragment of the calculus is obtained by restricting the set of continuation variables (ranged over by Greek letters) to a singleton (whose only element is conventionally the $\star$ symbol). This way every time a continuation is bound by the μ binder the previous bound continuation goes out of scope. The

Table 1. Syntax of the $\overline{\lambda}\mu\tilde{\mu}$-calculus & $\overline{\lambda}\mu\tilde{\mu}$-calculus reduction rules

Terms	*Environments*	*Reduction rules* :	
$v ::= x$	$E ::= \alpha$	$\langle \mu\alpha : T.c \Vert E\rangle$	$\rhd\ c\{E/\alpha\}$
$\mid\ \lambda x : T.v$	$\mid\ v \circ E$	$\langle v \Vert \tilde{\mu}x : T.c\rangle$	$\rhd\ c\{v/x\}$
$\mid\ \mu\alpha : T.c$	$\mid\ \tilde{\mu}x : T.c$	$\langle \lambda x : T.v_1 \Vert v_2 \circ E\rangle$	$\rhd\ \langle v_2 \Vert \tilde{\mu}x : T.v_1 \circ E\rangle$
Commands		η-like rules:	
$c ::= \langle v \Vert E\rangle$		μ-expansion:	$v \Rightarrow \mu\alpha : T.\langle v \Vert \alpha\rangle$
		$\tilde{\mu}$-expansion:	$E \Rightarrow \tilde{\mu}x : T.\langle x \Vert E\rangle$

rendering semantics in [4] provides an easy intuition: every time we state that we are going to prove something we are obliged to conclude it; in other words, subproofs must be well-nested. The intuitionistic fragment is Curry-Howard isomorphic to Gentzen LJ sequent calculus.

A term that is not in the intuitionistic fragment is said to be in the *classical* fragment of the calculus. In this fragment it is possible to bound a continuation, but later on give control to another continuation bound less recently in the past. The rendering semantics in [4] provides an easy intuition, but the result is not at all natural: we can state that we are going to prove something, but later on we can escape to an outer proof and conclude it instead. In doing this we can exploit the additional hypotheses we collected in the inner (and unfinished) proof. Since subproofs are not well-nested in this fragment, the pseudo-natural language obtained is not natural nor easy to understand. For this reason in [4] we apply the rendering semantics only to the intuitionistic fragment and provide a translation from the classical fragment to the intuitionistic fragment augmented with classical axioms that state excluded middle at each type. The intuitionistic fragment is Curry-Howard isomorphic to Gentzen LK sequent calculus.

Table 1 shows the reduction rules of the calculus, according to [2]. The first two rules may form a critical pair. Consistently solving the critical pair by giving priority to one of the two rules leads to a call-by-value (respectively call-by-name) strategy, that this way are shown to be perfectly dual. The classical fragment of the calculus is not deterministic, since for a critical pair there may be no common reduct to form a diamond. However, the intuitionistic fragment is deterministic and it is a closed subset with respect to reduction. The $\overline{\lambda}\mu\tilde{\mu}$-calculus typing rules and its principal meta-theoretical properties can be found in [2].

3 Representing Alternatives in Proofs in the $\overline{\lambda}\mu\tilde{\mu}$-Calculus

We are now interested in representing alternatives proofs and views in the $\overline{\lambda}\mu\tilde{\mu}$-calculus. Moreover, we want to avoid the addition to the calculus of new constructs and we also want to exploit a fragment as close as possible to the intuitionistic one. The latter requirement is necessary to preserve the good behavior of our rendering semantics that gives natural results only on that fragment.

The main idea underlying our encoding is that of seeing a view over a proof term just as reduced forms of the proof term (according to the reduction rules of the calculus). For each pair of alternative proofs in the calculus we have two possible set of views: one that picks the first view and one that picks the opposite one. Thus a proof term with an alternative must reduce non deterministically in two possible ways. This suggests that we must encode a pair of alternative proofs as a critical pair.

The $\overline{\lambda}\mu\tilde{\mu}$-calculus typing rules and the previous requirement suggest the following minimal encoding of a pair of alternative terms t_1 and t_2 that prove T:

$$\boxed{alt^T_r(t_1, t_2) := \mu\star : T.\langle \mu\_ : T.\langle t_1 || \star\rangle || \tilde{\mu}\_ : T.\langle t_2 || \star\rangle\rangle}$$

By duality we can also provide a continuation that embeds two alternative continuations E_1 and E_2 of type T. However, we will not need it in this paper and we consider as future work the study of fragments of the calculus that include it.

In the definition of $alt_r^T(t_1, t_2)$ we have used the notation $\mu_{\_} : T$ ($\tilde{\mu}_{\_} : T$) to remark that the bound term (continuation) will not occur in its scope. We also say that this occurrence of the binder is *affine*.

As requested, the term $alt_r^T(t_1, t_2)$ is subject to the following non deterministic reduction rules: $alt_r^T(t_1, t_2) \rhd \mu\star : T.\langle t_1||\star\rangle$ and $alt_r^T(t_1, t_2) \rhd \mu\star : T.\langle t_2||\star\rangle$. Notice that both right hand sides are μ-expanded forms respectively of t_1 and t_2 and thus they are extensionally equivalent to t_1 and t_2. Notice also that if t_1 and t_2 are both terms in the intuitionistic fragment then $alt_r^T(t_1, t_2)$ reduces only to terms in the intuitionistic fragment. Unfortunately, when the term is plugged into a command, it is also subject to the following reduction rule:

$$\langle alt_r^T(t_1, t_2)||E\rangle \rhd \langle \mu_{\_} : T.\langle t_1||E\rangle||\tilde{\mu}_{\_} : T.\langle t_2||E\rangle\rangle$$

The right hand side of the reduction rule is basically equivalent to the left hand side and it will enjoy all its interesting properties. However, the environment E is duplicated. According to our rendering semantics, the left hand side represents a large proof with two alternative subproofs. The right hand side represents two alternative large proofs that have duplicated parts. Thus we will be interested in preventing this form of reduction. Notice that we can easily syntactically detect the redexes we do not want to reduce. They are the redexes of the form:

$$\langle \mu\star : T.\langle \mu_{\_} : T'.c_1||\tilde{\mu}_{\_} : T''.c_2\rangle||E\rangle$$

Finally, the reader can check the following typing derivation for our encoding according to the typing rules given in [2]:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash t_1 : T|\star : T;\Delta \qquad \dfrac{}{\Gamma|\star : T \vdash \star : T;\Delta}}{\langle t_1||\star\rangle : \Gamma \vdash \star : T;\Delta}}{\Gamma \vdash \mu_{\_} : T.\langle t_1||\star\rangle : T|\star : T;\Delta} \qquad \dfrac{\dfrac{\Gamma \vdash t_2 : T|\star : T;\Delta \qquad \dfrac{}{\Gamma|\star : T \vdash \star : T;\Delta}}{\langle t_2||\star\rangle : \Gamma \vdash \star : T;\Delta}}{\Gamma|\tilde{\mu}_{\_} : T.\langle t_2||\star\rangle : T \vdash \star : T;\Delta}}{\langle \mu_{\_} : T.\langle t_1||\star\rangle||\tilde{\mu}_{\_} : T.\langle t_2||\star\rangle\rangle : \Gamma \vdash \star : T;\Delta}}{\Gamma \vdash \mu\star : T.\langle \mu_{\_} : T.\langle t_1||\star\rangle||\tilde{\mu}_{\_} : T.\langle t_2||\star\rangle\rangle : T|\Delta}$$

The typing derivation shows a few peculiarities we are now going to analyze. First of all, when introducing affine binders we have not added the variables bound by an affine binder to the premises of the introduction rule. This is consistent with the original typing rules since the missing premise cannot play any role in the derivation because it is not referenced in the term.

Thanks to the previous observation, we notice that the continuation context Δ plays a passive role in the derivation, being simply propagated from the root to the leaves of the tree. As a consequence the derivation holds also when Δ is empty. In the latter case the continuation context has always exactly one declaration, and the two premises of the tree become

$$\Gamma \vdash t_1 : T|\star : T \quad \text{and} \quad \Gamma \vdash t_2 : T|\star : T$$

If t_1 and t_2 are terms in the intuitionistic fragment, then the continuation $\star$ declared in the context cannot occur in any of them. Thus it can be dropped from the typing judgment.

To summarize, if t_1 and t_2 are terms in the intuitionistic fragment, then the following "morally intuitionistic" derived rule applies for $alt_r^T(t_1, t_2)$:

$$\frac{\Gamma \vdash t_1 : T|\emptyset \qquad \Gamma \vdash t_2 : T|\emptyset}{\Gamma \vdash alt_r^T(t_1, t_2) : T|\emptyset}$$

By structural induction on the typing derivation, the same derivation holds for t_1 and t_2 in the intuitionistic fragment extended with $alt_r^-(-,-)$.

Following a similar line of reasoning, we will also admit the environment

$$\boxed{alt_l^T(t_1, t_2) := \tilde{\mu}x : T.\langle \mu\_ : T.\langle t_1 || \star\rangle || \tilde{\mu}\_ : T.\langle t_2 || \star\rangle\rangle}$$

and more generally commands of the form $\boxed{c ::= \langle v || E\rangle \mid \langle \mu\_ : T.c || \tilde{\mu}\_ : T.c\rangle}$

Excursus: the affine fragment of the $\overline{\lambda}\mu\tilde{\mu}$-calculus is the fragment where each $\tilde{\mu}$-binder is either affine or binds the continuation variable $\star$. The affine fragment is a superset of the one we adopt for encoding alternative proofs.

It is interesting to ask whether the whole affine fragment is a natural candidate for being a proof fragment. For the fragment we use this is a consequence of being essentially intuitionistic. Thus we can ask if the whole affine fragment is essentially intuitionistic or if it is inherently classical and does not admit a natural explanation of its proofs in pseudo-natural language.

According to the syntax of the $\overline{\lambda}\mu\tilde{\mu}$-calculus, an affine binder can occur only in two positions: as the first subterm of a command, possibly prefixed by lambda-abstractions ($\langle \lambda x_1 : T_1 . \ldots \lambda x_n : T_n.\mu\_ : T.c || E\rangle$) — called a *spine* position — and as the first subterm of a "cons" environment, possibly prefixed by lambda-abstractions ($\lambda x_1 : T_1 . \ldots \lambda x_n : T_n.\mu\_ : T.c \circ E$) — called an *argument* position. Since we are supposed to analyze an extension of an essentially intuitionistic fragment, we suppose that continuation variables range over the singleton $\{\star\}$. We show how to prove without assuming any axiom and in two different ways the classical statement $(A \Rightarrow C) \Rightarrow ((A \Rightarrow B) \Rightarrow C) \Rightarrow C$. As far as we know, both proofs do not admit a reasonable translation to natural language.

The first proof exploits an affine binder in argument position:

$$\lambda H_1 : A \Rightarrow C.\lambda H_2 : (A \Rightarrow B) \Rightarrow C.\mu\star : C.\langle H_2 || (\lambda x : A.\mu\_ : B.\langle H_1 || x \circ \star\rangle) \circ \star\rangle$$

The classical core of the proof is represented by the term $\lambda x : A.\mu\_ : B.\langle H_1 || x \circ \star\rangle$. The term has type $A \Rightarrow B$, but it does not conclude B under the hypothesis A. As soon as A is known by hypothesis, the hypothesis is used in conjunction with H_1 to jump to the outer proof and conclude C. In the intuitionistic fragment this would be prevented by the μ binder that starts the proof of B binding $\star$ and hiding the previous declaration of $\star$. In the affine fragment, however, the μ binder can bind no variable, without hiding $\star$. As far as we know, there is no structural way of providing a natural explanation of the proof term above in natural language.

The second proof, that exploits an affine binder in spine position, $\tilde{\mu}$-reduces to the first proof:

$$\begin{array}{l}\lambda H_1 : A \Rightarrow C.\lambda H_2 : (A \Rightarrow B) \Rightarrow C. \\ \quad \mu\star : C.\langle(\lambda x : A.\mu_{\_} : B.\langle H_1)||x \circ \star\rangle||\tilde{\mu}y : A \Rightarrow B.\langle H_2||y \circ \star\rangle\rangle\end{array}$$

We conclude that the affine fragment is too large to be directly useful as a proof format. The restriction obtained by allowing affine binders only if not prefixed by lambda-abstractions is essentially intuitionistic, but does not seem to add much expressive power with respect to the intuitionistic fragment extended with the affine binders that occur in $alt_l^-(-,-)$ only.

Natural language rendering of alternative proofs. We equip the affine fragment considered to encode alternative proofs with the following *fully compositional* rendering semantics (according to [4]):

$$\begin{array}{rl}[\![\langle\mu_{\_} : T.c_1||\tilde{\mu}_{\_} : T.c_2\rangle]\!] = & \text{we provide two alternative proofs} \\ & \text{first proof:}\boxed{\hookrightarrow} \\ & \quad [\![c_1]\!] \\ & \text{alternative proof:}\boxed{\hookrightarrow} \\ & \quad [\![c_2]\!]\end{array}$$

The semantics is fully compositional since $[\![t_i]\!]$ occurs in output before $[\![t_j]\!]$ every time t_i occurs before t_j in the term. This is an important property since it allows one to translate a proof back and forth without any need to rearrange the subterms; in other words the translator can be implemented as a stream processor and a human being can easily understand a proof term running the translation in his head without need to take notes.

Views for alternative proofs, in the sense of [1], are produced from a proof (or a PDS) by picking just one alternative proof in each set of alternative choices. In the $\overline{\lambda}\mu\tilde{\mu}$-calculus we are considering, a view is obtained by reducing one of the two competing redexes of each critical pair in the proof term. This definition of a view is more informative than the corresponding one over PDSs and provides an effective guideline for extensions to more refined form of views.

4 Abstract Syntax for OMDoc Proofs

An extended OMDoc proof consists of hypothetical proof steps and proof steps that derive a new fact. These two proof steps correspond to the formal OMDoc proofs. In order to deal with alternative proof attempts, we add a third language construct marking the start of alternative proofs. The full grammar rules for proofs is given in Table 2. In Appendix A we present an adequate extension of the OMDoc document type definition to accommodate alternative proofs.

More specifically, hyp *L*:*F*;*PROOF* denotes the start of a hypothetical proof, where scope of the hypothesis *F* with label *L* reaches until the end of the rest

Table 2. Abstract Syntax for Formal OMDoc Proofs

$$\begin{aligned} \mathit{PROOF} &:= \mathsf{hyp}\ L{:}F; \mathit{PROOF} \\ &\mid \mathsf{alt}(\mathit{PROOF}_1 \mid \mathit{PROOF}_2) \\ &\mid \mathsf{derive}\ L : F\ \mathit{JUST}; \mathit{PROOF} \\ &\mid \mathsf{derive}\ \_ : F\ \mathit{JUST} \\ \mathit{ARG} &:= L \mid (F\ \mathit{PROOF}) \end{aligned}$$

$$\begin{aligned} \mathit{JUST} &:= \mathsf{method}\ M(\mathit{ARG}_1 \ldots \mathit{ARG}_n)\ \mathit{OPTEXP} \\ &\mid \mathsf{plan}(\mathit{ARG}_1 \ldots \mathit{ARG}_n)\ \boxed{(F\ \mathit{PROOF})} \\ &\mid \mathsf{sketch}(\mathit{ARG}_1 \ldots \mathit{ARG}_n) \\ \mathit{OPTEXP} &:= \mathsf{nil} \mid \boxed{(F\ \mathit{PROOF})} \end{aligned}$$

(where L are the labels of formulas, F the formulas, and M the references to methods.)

of the proof *PROOF*. Alternative subproofs are represented by $\mathsf{alt}(\mathit{PROOF}_1 \mid \mathit{PROOF}_2)$. The derive proof steps are the most complex and of the general form

$$\mathsf{derive}\ L : F\ \mathit{JUST}$$

Such a step states that we can derive the fact F by using the justification *JUST*. We allow three kinds of justifications *JUST* for derive-steps: It can either be a $\mathsf{sketch}(\mathit{ARG}_1 \ldots \mathit{ARG}_n)$ in the sense proof sketches in [5] (called “gap steps” in [3]) or it is a reference to a method $M(\mathit{ARG}_1 \ldots \mathit{ARG}_n)$ or it is the description of a subproof $\mathsf{plan}(\mathit{ARG}_1 \ldots \mathit{ARG}_n)$. In all cases the ARG_i are either premises or facts and associated proofs. Premises are referenced by their label L and a fact and its proof are represented by $(F\ \mathit{PROOF})$. If there is at least one fact and subproof, then this is a top-down proof step; otherwise it is a pure bottom-up proof step. A method can be calculus rule, but also a rule at some lower level of granularity which is associated with a proof at some higher-level of granularity. This accommodates the simultaneous representation of proofs for F at different levels of granularity that is necessary to encode the PDS [1]. In case the derive-step is the last step of a proof we require the labels L_i of the derived formulas to be undefined, which is indicated by _ in the grammar. Note that derive-steps are the only means to terminate a proof.

Further differences of the extended OMDoc proofs and the standard [3] are: (1) we consider only hypothesis and derivation steps that have *exactly one* formal formula (FMP) and (2) we do not allow for local declarations and definitions. The latter could easily be added and translated to $\overline{\lambda}\mu\tilde{\mu}$, but we omitted them for sake of simplicity. The first restriction is due to the fuzzy semantics of OMDoc proofs with several conclusions, that does not admit in the current state an explanation in $\overline{\lambda}\mu\tilde{\mu}$. Fixing the semantics of OMDoc will require also syntactic changes; we plan to do that in a future work, providing a correspondence with an already known extension of $\overline{\lambda}\mu\tilde{\mu}$ with primitive multiplicative conjunction.

The rendering semantics for extended OMDoc proofs is given in Table 3 where $[\![P]\!]$ denotes the rendering of the proof P. Due to the lack of space, we have omitted the explicit introduction of newlines. The rendering rules are straightforward: the difference between a derive-step being the final and thus concluding step of a proof or not is acknowledged by using the past tense *“. . . we proved. . . ”* instead of the present tense *“. . . we prove. . . ”* (for method application and proof descriptions) and *“. . . we show. . . ”* (for proof sketches).

Table 3. Rendering Semantics for Abstract OMDoc

$$\llbracket \mathsf{hyp}\ L{:}F; PROOF \rrbracket := \text{“}\textit{Assume } F\ (L) \llbracket PROOF \rrbracket\text{”}$$

$$\begin{aligned}
&\llbracket \mathsf{derive}\ L : F\ \mathsf{method}\ M(ARG_1, \dots, ARG_k)\ OPTEXP; PROOF' \rrbracket \\
&\qquad := \text{“}\textit{By } M\ \llbracket (ARG_1, \dots, ARG_k) \rrbracket^{al}\ \llbracket OPTEXP \rrbracket^{o}\ \textit{we prove } F\ (L); \llbracket PROOF' \rrbracket\text{”} \\
&\llbracket \mathsf{derive}\ L : F\ \mathsf{plan}(ARG_1, \dots, ARG_k)\ (F'\ PROOF); PROOF' \rrbracket \\
&\qquad := \text{“}\textit{We prove } F\ (L)\ \llbracket (ARG_1, \dots, ARG_k) \rrbracket^{al}\ \llbracket (F'\ PROOF) \rrbracket^{o}; \llbracket PROOF' \rrbracket\text{”} \\
&\llbracket \mathsf{derive}\ L : F\ \mathsf{sketch}(ARG_1, \dots, ARG_k); PROOF' \rrbracket \\
&\qquad := \text{“}\textit{We show } F\ (L\ \llbracket (ARG_1, \dots, ARG_k) \rrbracket^{al}; \llbracket PROOF' \rrbracket\text{”} \\
&\llbracket \mathsf{derive}\ \_{:}F\ \mathsf{method}\ M(ARG_1, \dots, ARG_k)\ OPTEXP \rrbracket \\
&\qquad := \text{“}\textit{By } M\ \llbracket (ARG_1, \dots, ARG_k) \rrbracket^{al}\ \llbracket OPTEXP \rrbracket^{o}\ \textit{we proved } F; \boxed{\hookleftarrow}\text{”} \\
&\llbracket \mathsf{derive}\ \_{:}F\ \mathsf{plan}(ARG_1, \dots, ARG_k)\ \boxed{(F'\ PROOF)} \rrbracket \\
&\qquad := \text{“}\textit{We proved } F\ \llbracket (ARG_1, \dots, ARG_k) \rrbracket^{al}\ \llbracket (F'\ PROOF) \rrbracket^{o}; \boxed{\hookleftarrow}\text{”} \\
&\llbracket \mathsf{derive}\ \_{:}F\ \mathsf{sketch}(ARG_1, \dots, ARG_k) \rrbracket \\
&\qquad := \text{“}\textit{We can obtain } F\ \llbracket (ARG_1, \dots, ARG_k) \rrbracket^{al}; \boxed{\hookleftarrow}\text{”} \\
&\llbracket \mathsf{alt}(PROOF_1 \mid PROOF_2) \rrbracket := \text{“}\textit{Either } \boxed{\hookrightarrow} \llbracket PROOF_1 \rrbracket \\
&\qquad\qquad\qquad\qquad\qquad\quad \textit{or } \boxed{\hookrightarrow} \llbracket PROOF_2 \rrbracket\text{”}
\end{aligned}$$

$$\begin{aligned}
&\llbracket () \rrbracket^{al} := \text{“”} && \llbracket (ARG_1, \dots, ARG_k) \rrbracket^{al} := \text{“}\textit{from } \llbracket ARG_1 \rrbracket^{a}, \dots,\ \textit{and } \llbracket ARG_k \rrbracket^{a}\text{”} \\
&\llbracket \mathsf{nil} \rrbracket^{o} := \text{“”} && \llbracket (F\ PROOF) \rrbracket^{a} := \text{“}F\ (\textit{proved by } \llbracket PROOF \rrbracket)\text{”} \\
&\llbracket L \rrbracket^{a} := \text{“}L\text{”} && \llbracket \boxed{(F\ PROOF)} \rrbracket^{o} := \text{“}(\textit{In detail: } \llbracket PROOF \rrbracket)\text{”}
\end{aligned}$$

5 Encoding OMDoc in $\overline{\lambda}\mu\tilde{\mu}$ and the Other Way Around

Tables 5 and 6 show forward and backward translations between OMDoc proofs in the full fragment and $\overline{\lambda}\mu\tilde{\mu}$-terms in the affine fragment of Table 4. The latter table also provides a rendering semantics for the fragment that is slightly more refined than the one given for the whole calculus.

Notice that the grammar of the fragment can be simplified, for instance by identifying the $Term$ and $ComplexArg$ productions, that are kept distinct to the benefit of the presentation of the rendering semantics.

Indeed, the reader can check that the only intuitionistic terms of the calculus that are not in the fragment are x in spine position and $\lambda x_1 : T_1. \dots \lambda x_n : T_n.x$ both in spine and argument position. In both cases a simple η-like μ-expansion can give an equivalent term in the fragment: x is expanded to $\mu\star : T.\langle x || \star \rangle$ and $\lambda x_1 : T_1. \dots \lambda x_n : T_n.x$ to $\lambda x_1 : T_1. \dots \lambda x_n : T_n.\mu\star : T.\langle x || \star \rangle$.

The new term “?” that can only occur as the first argument of a command is a linear placeholder for a missing term of the expected type.

By induction over OMDoc proof (respectively $\overline{\lambda}\mu\tilde{\mu}$-term) structure, it is possible to prove that the two translations behave as almost inverse functions. In particular $\llbracket - \rrbracket^{t}_{F}$ (in the first translation) is almost inverse of $\llbracket - \rrbracket$ (in the second one); $\llbracket - \rrbracket^{j}_{1}$ and $\llbracket - \rrbracket^{j}_{2}$ considered together are inverse of $\llbracket - \rrbracket^{m}$; the two $\llbracket - \rrbracket^{a}$ functions are inverse one of the other. The functions behave as inverse on every proof/term but

$$\mathsf{derive\ L : F}\ JUST\ PROOF_1; \mathsf{HYP\ L' : F'}; PROOF_2$$

Table 4. A $\overline{\lambda}\mu\tilde{\mu}$-fragment with its rendering semantics

Commands		*Environments*	
$c ::= \langle v' \Vert E\rangle$	$[\![v']\!][\![E]\!]$	$E ::= \star$	$\boxed{\hookleftarrow}$ done
$\mid\ \langle\mu_{\_} : T.c_1$		$\mid\ \tilde{\mu}x : T.c$	we proved T (x)
$\Vert\tilde{\mu}_{\_} : T.c_2\rangle$	we provide two alternative proofs		$[\![c]\!]$
	first proof: $\boxed{\hookrightarrow}$	$\mid\ a \circ E$	and $[\![a]\!]$
	$[\![c_1]\!]$		$[\![E]\!]$
	alternative proof: $\boxed{\hookrightarrow}$		
	$[\![c_2]\!]$		
ComplexArguments		*Arguments*	
$a' ::= \mu\star : T.c$	a proof of T $\boxed{\hookrightarrow}$	$a ::= x$	by x
	$[\![c]\!]$	$\mid\ a'$	$[\![a']\!]$
$\mid\ \lambda x : T.a'$	under hypothesis T (x)		
	$[\![a']\!]$		
SpineTerms		*Terms*	
$v' ::= x$	by x	$v ::= \lambda x : T.v$	suppose T (x)
$\mid\ ?$	by a conjecture		$[\![v]\!]$
$\mid\ v$	*if* $v = \mu\star : T.\langle v' \Vert \tilde{\mu}x : T.\langle x \Vert \star\rangle\rangle$ *then*	$\mid\ \mu\star : T.c$	we need to prove T
	by x (that proves T as follows $\boxed{\hookrightarrow}$		$\boxed{\hookrightarrow}[\![c]\!]$
	$[\![\langle v' \Vert \star\rangle]\!]$		
	else		
	by some proof (in detail $\boxed{\hookrightarrow}$		
	$[\![\langle v' \Vert \star\rangle]\!]$		

Table 5. OMDoc to $\overline{\lambda}\mu\tilde{\mu}$

$$[\![\mathsf{hyp}\ L{:}F; PROOF]\!]^t_{F\to F'} = \lambda L : F.[\![PROOF]\!]^t_{F'}$$
$$[\![\mathsf{derive}\ \_{:}F\ JUST]\!]^t_F = \mu\star : F.\langle[\![JUST]\!]^j_1 || [\![JUST]\!]^j_2(\star)\rangle$$
$$[\![\mathsf{derive}\ L{:}F\ JUST; PROOF]\!]^t_{F'} = \mu\star : F'.\langle[\![JUST]\!]^j_1 || [\![JUST]\!]^j_2(\tilde{\mu}L : F.\underline{\langle[\![PROOF]\!]^t_{F'} || \star\rangle}))\rangle$$
$$[\![\mathsf{alt}(PROOF_1 \mid PROOF_2)]\!]^t_F = alt^F_r([\![PROOF_1]\!]^t_F, [\![PROOF_2]\!]^t_F)$$
$$[\![\mathsf{sketch}(ARG_1 \ldots ARG_n)]\!]^j_1 = ?$$
$$[\![\mathsf{sketch}(ARG_1 \ldots ARG_n)]\!]^j_2(E) = [\![ARG_1]\!]^a \circ \cdots \circ [\![ARG_n]\!]^a \circ E$$
$$[\![\mathsf{plan}(ARG_1 \ldots ARG_n)\ \boxed{(F\ PROOF)}\ ]\!]^j_1 = [\![PROOF]\!]^t_F$$
$$[\![\mathsf{plan}(ARG_1 \ldots ARG_n)\ \boxed{(F\ PROOF)}\ ]\!]^j_2(E) = [\![ARG_1]\!]^a \circ \cdots \circ [\![ARG_n]\!]^a \circ E$$
$$[\![\mathsf{method}\ M(ARG_1 \ldots ARG_n)\ \boxed{(F\ PROOF)}\ ]\!]^j_1 = \mu\star : F.\langle[\![PROOF]\!]^t_F || \tilde{\mu}M : F.\langle M || \star\rangle\rangle$$
$$[\![\mathsf{method}\ M(ARG_1 \ldots ARG_n)\ \boxed{(F\ PROOF)}\ ]\!]^j_2(E) = [\![ARG_1]\!]^a \circ \cdots \circ [\![ARG_n]\!]^a \circ E$$
$$[\![\mathsf{method}\ M(ARG_1 \ldots ARG_n)\ \mathsf{nil}]\!]^j_1 = M$$
$$[\![\mathsf{method}\ M(ARG_1 \ldots ARG_n)\ \mathsf{nil}]\!]^j_2(E) = [\![ARG_1]\!]^a \circ \cdots \circ [\![ARG_n]\!]^a \circ E$$
$$[\![L]\!]^a = L$$
$$[\![(F\ PROOF)]\!]^a = [\![PROOF]\!]^t_F$$

After the translation, underlined μ-redexes of the form $\underline{\langle\mu\star : T.c || \star\rangle}$ (also comprising the case $\mu\star : T.c \equiv alt^T_r(v_1, v_2)$) must be μ-reduced to c. Underlining of the remaining commands must be removed.

Table 6. $\overline{\lambda}\mu\tilde{\mu}$ to OMDoc

$$\llbracket \lambda x : T.v \rrbracket = \mathsf{hyp}\ x :\ T; \llbracket v \rrbracket$$
$$\llbracket \mu\star : T.\langle v || v_1 \circ \cdots \circ v_n \circ \star\rangle \rrbracket = \mathsf{derive}\ \_ : T\ \llbracket v \rrbracket^m(\llbracket v_1 \rrbracket^a, \ldots, \llbracket v_n \rrbracket^a)$$
$$\llbracket \mu\star : T.\langle v || v_1 \circ \cdots \circ v_n \circ \tilde{\mu}x : T'.c\rangle \rrbracket = \mathsf{derive}\ x{:}T'\ \llbracket v \rrbracket^m(\llbracket v_1 \rrbracket^a, \ldots, \llbracket v_n \rrbracket^a); \llbracket \mu\star : T.c \rrbracket$$
$$\dagger\quad \llbracket x \rrbracket = \text{ruled out}$$
$$\llbracket alt_r^T(v_1, v_2) \rrbracket = \mathsf{alt}(\llbracket v_1 \rrbracket \mid \llbracket v_2 \rrbracket)$$
$$\llbracket x \rrbracket^m(ARG_1, \ldots, ARG_n) = \mathsf{method}\ x(ARG_1 \ldots ARG_n)$$
$$\llbracket ? \rrbracket^m(ARG_1, \ldots, ARG_n) = \mathsf{sketch}(ARG_1, \ldots, ARG_n)$$
$$\circ\quad \llbracket \mu\star : T.\langle v || \tilde{\mu}x : T.\langle x || \star\rangle\rangle \rrbracket^m(ARG_1, \ldots, ARG_n) = \mathsf{method}\ x(ARG_1 \ldots ARG_n)\ \boxed{\llbracket v \rrbracket}$$
$$\llbracket v \rrbracket^m(ARG_1, \ldots, ARG_n) = \mathsf{plan}(ARG_1 \ldots ARG_n)\ \boxed{\llbracket v \rrbracket}\quad \text{for } v \notin \{x, ?, \mu\star : T.\langle v || \tilde{\mu}x : T.\langle x || \star\rangle\rangle\}$$
$$\llbracket x \rrbracket^a = x$$
$$\llbracket \mu\star : T.c \rrbracket^a = (T\ \llbracket \mu\star : T.c \rrbracket)$$
$$\llbracket alt_r^T(v_1, v_2) \rrbracket^a = (T\ \llbracket alt_r^T(v_1, v_2) \rrbracket$$
$$\llbracket \lambda x_1 : T_1. \ldots \lambda x_n : T_n.\mu\star : T.c \rrbracket^a = (T_1 \Rightarrow \cdots \Rightarrow T_n \Rightarrow T\ \llbracket \lambda x_1 : T_1. \ldots \lambda x_n : T_n.\mu\star : T.c \rrbracket)$$
$$\llbracket \lambda x_1 : T_1. \ldots \lambda x_n : T_n.alt_r^T(v_1, v_2) \rrbracket^a = (T_1 \Rightarrow \cdots \Rightarrow T_n \Rightarrow T\ \llbracket \lambda x_1 : T_1. \ldots \lambda x_n : T_n.alt_r^T(v_1, v_2) \rrbracket$$
$$\dagger\quad \llbracket \lambda x_1 : T_1. \ldots \lambda x_n : T_n.x \rrbracket^a = \text{ruled out}$$

The rule marked with ○ is necessary to make this translation inverse of the translation from OMDoc to $\overline{\lambda}\mu\tilde{\mu}$ (Table 5). The (error) rules marked with † are never applied when translating terms generated from OMDoc.

that, translated to the $\overline{\lambda}\mu\tilde{\mu}$-calculus and back, becomes the richer term

$$\mathsf{derive\ L : F}\ \mathit{JUST\ PROOF}_1; \mathsf{derive}\_{:}\mathsf{F}''\ \mathsf{plan()}\ (\mathsf{F}''\ \mathsf{HYP\ L' : F'}; \mathit{PROOF}_2)$$

that states explicitly what the hypothetical proof proves. The reader can check the rendering semantics associated to the two OMDoc proofs.

We illustrate the semantics provided to our abstract OMDoc proofs with the following abstract and partial proof of the irrationality of $\sqrt{12}$:

1. *The proof is by contradiction*
2. *We assume* $rat(\sqrt{12})$*;*
3. *We show there are* n, m*, such that* $int(n) \wedge int(m) \wedge \neg commondiv(n, m) \wedge \sqrt{12} = \frac{n}{m}$*;*
4. *By Lemma* $\sqrt{z} = \frac{x}{y} \Rightarrow z \times y^2 = x^2$ *we know* $12 \times m^2 = n^2$*;*
5. *We show* $commondiv(n, m)$*;*
6. *We have a contradiction.*

In this proof, the proof steps (3.) and (5.) are only descriptions of more complicated proofs which are made explicit in the encoding of this proof given in Fig. 1. Note further that the expansion of proof step (5.) contains alternative proofs for the shown statement[1] .

More specifically, we illustrate the semantics by showing (1) the rendering of that proof using the rendering semantics from Table 3, (2) the $\overline{\lambda}\mu\tilde{\mu}$ resulting

[1] Note that the second alternative of using the prime divisor 2 basically comes back to use the prime divisor 3. So it is a proof with detour, but it is a proof.

```
derive _ : $\neg rat(\sqrt{12})$
 method ProofByContradiction  $((rat(\sqrt{12}) \Rightarrow \bot$
      hyp  $L_0 : rat(\sqrt{12});$
      derive  $L_1 : int(n) \wedge int(m) \wedge \neg commondiv(n,m) \wedge \sqrt{12} = \frac{n}{m}$
   plan($L_0$)  $(rat(\sqrt{12}) \Rightarrow int(n) \wedge int(m) \wedge \neg commondiv(n,m) \wedge \sqrt{12} = \frac{n}{m}$
                hyp  $L_{10} : rat(\sqrt{12});$ derive  $L_{11} : \exists y{:}int, z{:}int. \sqrt{12} = \frac{y}{z} \wedge \neg commondiv(y,z)$
                                             method ApplyLemma(Rat-Criterion, $L_{10}$) ;
                derive _ : $int(n) \wedge int(m) \wedge \neg commondiv(n,m) \wedge \sqrt{12} = \frac{n}{m}$
                       method decomposition($L_{11}$))
      derive  $L_2 : 12 \times m^2 = n^2$ method  ApplyLemma($\sqrt{z} = \frac{x}{y} \Rightarrow z \times y^2 = x^2$,$L_1$) ;
      derive  $L_3 : commondiv(n,m)$
   plan($L_2$)  $(12 \times m^2 = n^2 \Rightarrow commondiv(n,m)$
               hyp  $L_{30} : 12 \times m^2 = n^2$
               alt  ( derive  $L_{31} : div(n,3) \wedge div(m,3)$
                        method  and-I $(div(n,3)\ \ldots\ )\ (div(m,3)\ \ldots\ );\ \ldots$
                    | derive  $L_{34} : div(n,2) \wedge div(m,2)$
                        method  and-I $(div(n,2)\ \ldots\ )\ (div(m,2)\ \ldots\ );\ \ldots);$
      derive _ : $\bot$ method Contradiction($L_1, L_3$)))
```

Fig. 1. Part of a Proof of the irrationality of $\sqrt{12}$

from the translation of that proof and (3) the rendering of that $\overline{\lambda}\mu\tilde{\mu}$-term using the $\overline{\lambda}\mu\tilde{\mu}$-rendering semantics from Table 4.

The rendering of the OMDoc proof from Fig. 1 is the following:

Proof: *By ProofByContradiction from* $rat(\sqrt{12}) \Rightarrow \bot$
(proved by: Assume $rat(\sqrt{12})$ *(*L_0*)*
We prove $int(n) \wedge int(m) \wedge \neg commondiv(n,m) \wedge \sqrt{12} = \frac{n}{m}$ *(*L_1*) from* L_0
(In details: Assume $rat(\sqrt{12})$ *(*L_{10}*) By ApplyLemma from Rat-Criterion and* L_{10} *we prove* $\exists y{:}int, z{:}int. \sqrt{12} = \frac{y}{z} \wedge \neg commondiv(y,z)$ *(*L_{11}*)*
By decomposition from L_{11} *we proved* $int(n) \wedge int(m) \wedge \sqrt{12} = \frac{n}{m} \wedge \neg commondiv(n,m)$*);*
By ApplyLemma from $\sqrt{z} = \frac{x}{y} \Rightarrow z \times y^2 = x^2$ *and* L_1 *we prove* $12 \times m^2 = n^2$ *(*L_2*);*
We prove $commondiv(n,m)$ *(*L_3*) from* L_2
(In details:
Assume $12 \times m^2 = n^2$ *(*L_{30}*)*
Either by and-I from $div(n,3)$ *(proved by* $[\![\ldots]\!]$*) and* $div(m,3)$ *(proved by* $[\![\ldots]\!]$*) we proved* $div(n,3) \wedge div(m,3)$ *(*L_{31}*); ...*
Or by and-I from $div(n,2)$ *(proved by* $[\![\ldots]\!]$*) and* $div(m,2)$ *(proved by* $[\![\ldots]\!]$*) we proved* $div(n,2) \wedge div(m,2)$ *(*L_{34}*); ...)*
By Contradiction from L_1 *and* L_3 *we proved* $\bot$*)*
we proved $\neg rat(\sqrt{12})$

The $\overline{\lambda}\mu\tilde{\mu}$-term obtained by translation using the rules Table 5 after μ-reduction of the underlined μ-redexes of the form $\langle \mu\star : T.\langle \mu_{\_} : T'.c_1 || \tilde{\mu}_{\_} : T''.c_2\rangle || E\rangle$ is shown in Fig. 2.

The rendering of the $\overline{\lambda}\mu\tilde{\mu}$-proof from Fig. 2 according to Table 4 yields:

$$
\begin{array}{l}
\mu\star : \neg rat\sqrt{12}. \\
\quad \langle \mathsf{ProofByContradiction} \\
\quad\quad \| \ \lambda L_0 : rat(\sqrt{12}). \\
\quad\quad\quad \mu\star : \bot . \langle \ \mu\star : rat(\sqrt{12}) \Rightarrow int(n) \wedge int(m) \wedge \neg commondiv(n,m) \wedge \sqrt{12} = \frac{n}{m}. \\
\quad\quad\quad\quad\quad\quad \lambda L_{10} : rat(\sqrt{12}). \\
\quad\quad\quad\quad\quad\quad\quad \langle \mathsf{ApplyLemma} \ \| \\
\quad\quad\quad\quad\quad\quad\quad\quad \mathsf{Rat\text{-}Criterion} \circ L_{10} \circ \tilde{\mu} L_{11} : \exists y{:}int, z{:}int_{\blacksquare} \sqrt{12} = \frac{y}{z} \wedge \neg commondiv(y,z). \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \langle \mathsf{decomposition} \ \| \ L_{11} \circ \star \rangle \rangle \\
\quad\quad\quad\quad\quad \| \ L_0 \circ \tilde{\mu} L_1 : int(n) \wedge int(m) \wedge \neg commondiv(n,m) \wedge \sqrt{12} = \frac{n}{m}. \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \langle \mathsf{ApplyLemma} \ \| \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad [\![\sqrt{z} = \frac{x}{y} \Rightarrow z \times y^2 = x^2]\!] \circ L_1 \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \circ \tilde{\mu} L_2 : 12 \times m^2 = n^2. \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \langle \ \lambda L_{30} : 12 \times m^2 = n^2. \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad alt_r(\mu\star : commondiv(n,m). \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \langle \mathsf{and\text{-}I} \ \| \ [\![(div(n,3) \ldots)]\!] \circ [\![(div(m,3) \ldots)]\!] \circ \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \tilde{\mu} L_{31} : div(n,3) \wedge div(m,3). \langle [\![\ldots]\!], \| \star \rangle \rangle, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mu\star : commondiv(n,m). \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \langle \mathsf{and\text{-}I} \ \| \ [\![(div(n,2) \ldots)]\!] \circ [\![(div(m,2) \ldots)]\!] \circ \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \tilde{\mu} L_{34} : div(n,2) \wedge div(m,2). \langle [\![\ldots]\!], \| \star \rangle \rangle) \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \| \ L_2 \circ \tilde{\mu} L_3 : commondiv(n,m). \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \langle \mathsf{Contradiction} \ \| \ L_1 \circ L_3 \circ \star \rangle \rangle \circ \star \rangle \rangle \rangle
\end{array}
$$

Fig. 2. $\overline{\lambda}\mu\tilde{\mu}$-Proof obtained by translation and after reduction of μ-redexes

Proof: *we need to prove* $\neg rat(\sqrt{12})$

by ProofByContradiction *and the hypothesis* $rat(\sqrt{12})$ *(*L_0*) a proof of* $\bot$

by some proof

(in detail: we need to prove $rat(\sqrt{12}) \Rightarrow int(n) \wedge int(m) \wedge \neg commondiv(n,m) \wedge \sqrt{12} = \frac{n}{m}$*: Suppose* $rat(\sqrt{12})$ *(*L_{10}*); By* ApplyLemma *and* Rat-Criterion *and* L_0 *we proved* $\exists y{:}int, z{:}int_{\blacksquare} \sqrt{12} = \frac{y}{z} \wedge \neg commondiv(y,z)$ *(*L_{11}*). By* decomposition *and* L_{11}*. Done)*

and L_0 *we proved* $int(n) \wedge int(m) \wedge \neg commondiv(n,m) \wedge \sqrt{12} = \frac{n}{m}$*;*

By ApplyLemma *and* $\sqrt{z} = \frac{x}{y} \Rightarrow z \times y^2 = x^2$ *and* L_1 *we proved* $12 \times m^2 = n^2$ *(*L_2*);*

By some proof

(in detail: Suppose $12 \times m^2 = n^2$ *(*L_{30}*); we provide two alternative proofs:*

First proof: we need to prove $commondiv(n,m)$*: By* and-I *and* $[\![(div(n,3) \ldots)]\!]$ $[\![(div(m,3) \ldots)]\!]$ *we proved* $div(n,3) \wedge div(m,3)$*;* $[\![\ldots]\!]$ *Done.*

Second proof: we need to prove $commondiv(n,m)$*: By* and-I *and* $[\![(div(n,2) \ldots)]\!]$ $[\![(div(m,2) \ldots)]\!]$ *we proved* $div(n,2) \wedge div(m,2)$*;* $[\![\ldots]\!]$ *Done.)*

and L_2 *we have proved* $commondiv(n,m)$ *(*L_3*);*

By Contradiction *and* L_1 *and* L_3 *done.*

Done.

The informative content of the natural language explanation obtained from OMDoc and that obtained from the corresponding $\overline{\lambda}\mu\tilde{\mu}$-calculus encoding are clearly almost equivalent. The two main differences are omission of repetitions of the thesis: (1) in the two alternative proofs the local thesis is restated in the $\overline{\lambda}\mu\tilde{\mu}$-calculus, but not in OMDoc; (2) at the end of the first expanded proof OMDoc states again what has been proved while the $\overline{\lambda}\mu\tilde{\mu}$-calculus does not. It would certainly be possible as a future work to change one or both renderings to obtain two syntactically closer texts. However, the interest would be limited, since we are already convinced that the informative content is equivalent and since neither of the two is really more readable or natural than the other.

6 Conclusion

In this paper we have continued the investigation of the $\overline{\lambda}\mu\tilde{\mu}$-calculus as a proof format, including the representation of alternative proofs and proofs at different level of details. Alternative proofs can be easily accommodated in a non deterministic fragment of the calculus that remains essentially intuitionistic, admitting proof explanation in a pseudo-natural language.

We have also demonstrated how it is possible to establish a tight correspondence between OMDoc and the $\overline{\lambda}\mu\tilde{\mu}$-calculus, that imposes a clear understanding of OMDoc proofs (and, indirectly, for the corresponding PDSs) in terms of proofs in a given logic. It is now possible to speak, for instance, of cut elimination for OMDoc, considering the operation inherited by the formal correspondence.

The pseudo natural language generation considered in this paper is to be understood as a textual representation of the proof that allows to understand it in all its details. It is not meant to be a *nice* or *natural* description of the proof – that we leave to experts in linguistics. However, it is important to provide it to further constrain the translation between the two proof formats: only a translation that essentially preserves the two independently given rendering semantics is acceptable, pruning out irrelevant embeddings between mathematically unrelated formats that can represent anything (like Lisp S-expressions or plain XML).

Finally, a few difficulties we have faced in establishing the correspondence could be understood as flaws in the OMDoc recommendation and could guide the future development of the language. For instance, the fact that hypothetical proof steps that follow derive steps when translated to $\overline{\lambda}\mu\tilde{\mu}$-calculus and read back are enriched with the statement the hypothetical step is proving. Concretely, the abstract syntax for OMDoc proofs we have considered already represents an extension of OMDoc that clarifies the role of proofs at different levels of detail and that adds alternative proofs.

As a future work we plan to continue the study of the correspondence between the two languages and their natural language renderings, in order to improve OMDoc and to unveil other remarkable properties of bigger and bigger fragments of the $\overline{\lambda}\mu\tilde{\mu}$-calculus used as a proof format.

References

1. S. Autexier, C. Benzmüller, D. Dietrich, A. Meier, C. Wirth. “A Generic Modular Data Structure for Proof Attempts Alternating on Ideas and Granularity”. In Fourth International Conference on Mathematical Knowledge Management (MKM2005), Lecture Notes in Artificial Intelligence (LNAI), Vol. 3863, 2006.
2. P. Curien, H. Herbelin. “The duality of computation”. In Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP’00), ACM, SIGPLAN Notices 35(9), ISBN:1-58113-2-2-6, 233–243, 2000.
3. M. Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.2).*
4. C. Sacerdoti Coen. “Explanation in Natural language of $\overline{\lambda}\mu\tilde{\mu}$-terms”. In Fourth International Conference on Mathematical Knowledge Management (MKM2005), Lecture Notes in Artificial Intelligence (LNAI), Vol. 3863, 2006.
5. F. Wiedijk. “Formal Proof Sketches”. In S. Berardi, M. Coppo and F. Damiani eds., Types for Proofs and Programs: Third International Workshop, TYPES 2003, LNCS 3085, 378–393, 2004.

A DTD Extension for OMDoc with Alternative Proofs

```
<!--
   An XML DTD for Open Mathematical documents (OMDoc 1.2) Module PF
      SYSTEM http://www.mathweb.org/omdoc/dtd/omdocpf.mod
      PUBLIC -//OMDoc//ELEMENTS OMDoc PF V1.2//EN
   See the documentation and examples at http://www.mathweb.org/omdoc
   (c) 1999-2003 Michael Kohlhase, released under the GNU Public
       License (GPL)

   -- Added element <alt> for alternative proofs
      (Serge Autexier & Claudio Sacerdoti-Coen (Mai 2006))
-->

<!-- qnames for omdoc statements -->
<!ENTITY % omdocpf.metacomment.qname "%omdoc.pfx;metacomment">
<!ENTITY % omdocpf.derive.qname "%omdoc.pfx;derive">
<!ENTITY % omdocpf.hypothesis.qname "%omdoc.pfx;hypothesis">
<!ENTITY % omdocpf.method.qname "%omdoc.pfx;method">
<!ENTITY % omdocpf.premise.qname "%omdoc.pfx;premise">
<!ENTITY % omdocpf.alternatives.qname "%omdoc.pfx;alt">

<!ELEMENT %omdocpf.proof.qname;
          (%omdocdoc.meta.content;
           (%ss;|%omdocmtxt.omtext.qname;
                |%omdocst.symbol.qname;
                |%omdocst.definition.qname;
                |%omdocpf.derive.qname;
                |%omdocpf.hypothesis.qname;
            )*
           (%omdocpf.alternatives.qname;)?
           )>
```

```
<!ATTLIST %omdocpf.proof.qname;
          %omdoc.common.attribs;
          %omdoc.toplevel.attribs;
          %fori.attrib;>

<!ELEMENT %omdocpf.proofobject.qname;
          (%omdocdoc.meta.content;(%omdocmobj.class;))>
<!ATTLIST %omdocpf.proofobject.qname;
          %omdoc.common.attribs;
          %omdoc.toplevel.attribs;
          %fori.attrib;>

<!ELEMENT %omdocpf.derive.qname;
          (%omdocmtxt.MCF.content;,(%ss;|%omdocpf.method.qname;)?)>
<!ATTLIST %omdocpf.derive.qname;
          %omdoc.common.attribs;
          type CDATA #IMPLIED
          %id.attrib;>

<!ELEMENT %omdocpf.hypothesis.qname; (%omdocmtxt.MCFS.content;)>
<!ATTLIST %omdocpf.hypothesis.qname;
          %omdoc.common.attribs;
          %id.attrib;
          inductive (yes|no) #IMPLIED>

<!ELEMENT %omdocpf.alternatives.qname;
          (%omdocpf.proof.qname;
           (%omdocpf.proof.qname;)+
          )>

<!ATTLIST %omdocpf.alternatives.qname;
          %omdoc.common.attribs;
          %id.attrib;
          >

<!ELEMENT %omdocpf.method.qname;
                  (%omdocmobj.class;|%omdocpf.premise.qname;
                  |%omdocpf.proof.qname;|%omdocpf.proofobject.qname;)*>
<!ATTLIST %omdocpf.method.qname; %omdoc.common.attribs; %xrefi.attrib;>
<!-- 'xref' is a pointer to the element defining the method -->

<!ELEMENT %omdocpf.premise.qname; EMPTY>
<!ATTLIST %omdocpf.premise.qname; %omdoc.common.attribs;
          %xref.attrib; rank CDATA "0">
<!-- The rank of a premise specifies its importance in the
     inference rule. Rank 0 (the default) is a real premise,
     whereas positive rank signifies sideconditions of
     varying degree. -->
```

Proof Transformation by CERES*

Matthias Baaz[1], Stefan Hetzl[2], Alexander Leitsch[2],
Clemens Richter[2], and Hendrik Spohr[2]

[1] Institute of Discrete Mathematics and Geometry (E104),
Vienna University of Technology, Wiedner Hauptstraße 8-10,
1040 Vienna, Austria
baaz@logic.at

[2] Institute of Computer Languages (E185),
Vienna University of Technology, Favoritenstraße 9,
1040 Vienna, Austria
{hetzl, leitsch, richter, spohr}@logic.at

Abstract. Cut-elimination is the most prominent form of proof transformation in logic. The elimination of cuts in formal proofs corresponds to the removal of intermediate statements (lemmas) in mathematical proofs. The cut-elimination method CERES (cut-elimination by resolution) works by constructing a set of clauses from a proof with cuts. Any resolution refutation of this set then serves as a skeleton of an **LK**-proof with only atomic cuts.

In this paper we present an extension of CERES to a calculus **LKDe** which is stronger than the Gentzen calculus **LK** (it contains rules for introduction of definitions and equality rules). This extension makes it much easier to formalize mathematical proofs and increases the performance of the cut-elimination method. The system CERES already proved efficient in handling very large proofs.

1 Introduction

Proof analysis is a central mathematical activity which has proved crucial to the development of mathematics. Many mathematical concepts such as the notion of group or the notion of probability were introduced by analyzing existing arguments. In some sense the analysis and synthesis of proofs form the very core of mathematical progress[13,14].

Cut-elimination introduced by Gentzen [9] is the most prominent form of proof transformation in logic and plays an important role in automating the analysis of mathematical proofs. The removal of cuts corresponds to the elimination of intermediate statements (lemmas), resulting in a proof which is analytic in the sense, that all statements in the proof are subformulas of the result. Therefore, the proof of a combinatorial statement is converted into a purely combinatorial proof. Cut-elimination is therefore an essential tool for the analysis of proofs, especially to make implicit parameters explicit. In particular, cut free derivations allow for:

* Supported by the Austrian Science Fund (project no. P17995-N12).

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 82–93, 2006.

- the extraction of Herbrand disjunctions, which can be used to establish bounds on existential quantifiers (e.g. Luckhardt's analysis of the Theorem of Roth [11]),
- the construction of interpolants, which allow the replacement of implicit definitions by explicit ones according to Beth's Theorem,
- the calculation of generalized variants of the end formula.

In a formal sense Girard's analysis of van der Waerden's theorem [10] is the application of cut-elimination to the proof of Fürstenberg/Weiss with the "perspective" of obtaining van der Waerden's proof. Indeed an application of a complex proof transformation like cut-elimination by humans requires a goal oriented strategy. In contrast, such a transformation can be done purely automatically, which also might result in unexpected and interesting results [3]. Note that cut-elimination is *non-unique*, i.e. there is no single cut-free proof which represents *the* analytic version of a proof with lemmas. Indeed, it is non-uniqueness which makes computational experiments with cut-elimination interesting. The experiments can be considered as a source for a base of proofs in formal format which provide different mathematical and computational information.

CERES [6] is a cut-elimination method that is based on resolution. The method roughly works as follows: The structure of the proof containing cuts is mapped to a clause term which evaluates to an unsatisfiable set of clauses $\mathcal{C}$ (the *characteristic clause set*). A resolution refutation of $\mathcal{C}$, which is obtained using a first-order theorem prover, serves as a skeleton for the new proof which contains only atomic cuts. In a final step also these atomic cuts can be eliminated, provided the (atomic) axioms are valid sequents; but this step is of minor mathematical interest only. In the system CERES[1] this method of cut-elimination has been implemented. The system is capable of dealing with formal proofs in **LK**, among them also very large ones.

The extension of CERES to a calculus containing definition-introduction and equality rules moves the system closer to real mathematical proofs. By its high efficiency (the core of the method is first-order theorem proving by resolution and paramodulation) CERES might become a strong tool in *automated proof mining* and contribute to an experimental culture of *computer-aided proof analysis* in mathematics.

2 Extensions of Gentzen's LK

Gentzen's **LK** is the original calculus for which cut-elimination was defined. The original version of CERES in based on **LK** and several variants of it (we just refer to [6] and [7]). In formalizing mathematical proofs it turns out that **LK** (and also natural deduction) are not sufficiently close to real mathematical inference. First of all, the calculus **LK** lacks an efficient handling of equality

[1] Available at `http://www.logic.at/ceres/`

(in fact equality axioms have to be added to the end-sequent). Due to the importance of equality this defect was apparent to proof theorists; e.g. Takeuti [15] gave an extension of **LK** to a calculus **LKe**, adding atomic equality axioms to the standard axioms of the form $A \vdash A$. The advantage of **LKe** over **LK** is that no new axioms have to be added to the end-sequent; on the other hand, in presence of the equality axioms, full cut-elimination is no longer possible, but merely reduction to *atomic cut.* But still **LKe** uses the same rules as **LK** as equality is axiomatized. On the other hand, in formalizing mathematical proofs, using equality as a *rule* is much more natural and concise. For this reason we choose the most natural equality rule, which is strongly related to paramodulation in automated theorem proving. Our approach differs from this in [17], where a unary equality rule is used (which does not directly correspond to paramodulation). The *equality rules* are:

$$\frac{\Gamma_1 \vdash \Delta_1, s = t \quad A[s]_\Lambda, \Gamma_2 \vdash \Delta_2}{A[t]_\Lambda, \Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2} =: l1 \qquad \frac{\Gamma_1 \vdash \Delta_1, t = s \quad A[s]_\Lambda, \Gamma_2 \vdash \Delta_2}{A[t]_\Lambda, \Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2} =: l2$$

for inference on the left and

$$\frac{\Gamma_1 \vdash \Delta_1, s = t \quad \Gamma_2 \vdash \Delta_2, A[s]_\Lambda}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, A[t]_\Lambda} =: r1 \qquad \frac{\Gamma_1 \vdash \Delta_1, t = s \quad \Gamma_2 \vdash \Delta_2, A[s]_\Lambda}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, A[t]_\Lambda} =: r2$$

on the right, where Λ denotes a set of positions of subterms where replacement of s by t has to be performed. We call $s = t$ the *active equation* of the rules.

In CERES it is crucial that all nonlogical rules (which also work on atomic sequents) correspond to clausal inference rules in automated deduction. While cut and contraction correspond to resolution (and factoring, dependent on the version of resolution), the equality rules $=: l1, =: l2, =: r1, =: r2$ correspond to paramodulation, which is the most efficient equality rule in automated deduction [12]. Indeed, when we compute the most general unifiers and apply them to the paramodulation rule, then it becomes one of the rules $=: l1, =: l2$, $=: r1, =: r2$.

Perhaps the most significant tool in structuring mathematical proofs is the introduction of new concepts (formally definition-introduction). Though the use of definition introductions can be simulated by cuts, this simulation is rather unnatural and has a negative effect on the CERES-algorithm as will be explained in section 3 (definition introduction is a unary rule, while cut is a binary one).

The *definition rules* directly correspond to the *extension principle* (see [8]) in predicate logic. It simply consists in introducing new predicate- and function symbols as abbreviations for formulas and terms. Let A be a first-order formula with the free variables $x_1, \ldots, x_k$ (denoted by $A(x_1, \ldots, x_k)$) and P be a *new* k-ary predicate symbol (corresponding to the formula A). Then the rules are:

$$\frac{A(t_1, \ldots, t_k), \Gamma \vdash \Delta}{P(t_1, \ldots, t_k), \Gamma \vdash \Delta}\ \mathrm{def}_P\colon l \qquad \frac{\Gamma \vdash \Delta, A(t_1, \ldots, t_k)}{\Gamma \vdash \Delta, P(t_1, \ldots, t_k)}\ \mathrm{def}_P\colon r$$

for arbitrary sequences of terms $t_1, \ldots, t_k$. Definition introduction is a simple and very powerful tool in mathematical practice. Note that the introduction of important concepts and notations like groups, integrals etc. can be formally described by introduction of new symbols. There are also definition introduction rules for new function symbols which are of similar type.

The *axiom system* for **LKDe** may be an arbitrary set of atomic sequents containing the sequents $A \vdash A$ (for atomic formulas A) which is closed under substitution. The only axioms which have to be added for equality are $\vdash s = s$ where s is an arbitrary term. So every axiom system has to contain the axioms $A \vdash A$ and $\vdash s = s$.

The calculus **LKDe** is **LK** extended by the equality rules and by the (infinite set of) definition-introduction rules. Clearly these extensions do not increase the logical expressivity of the calculus, but they make him much more compact and natural. To illustrate the rules defined above we give a simple example. The aim is to prove the (obvious) theorem that a number divides the square of a number b if it divides b itself. In the formalization below a and b are constant symbols and the predicate symbol D stands for "divides" and is defined by

$$D(x, y) \leftrightarrow \exists z.x * z = y.$$

The active equations are written in boldface.

$$\dfrac{\vdash (\mathbf{a} * \mathbf{z_0}) * \mathbf{b} = \mathbf{a} * (\mathbf{z_0} * \mathbf{b}) \qquad \dfrac{a * z_0 = b \vdash \mathbf{a} * \mathbf{z_0} = \mathbf{b} \quad \vdash b * b = b * b}{a * z_0 = b \vdash (a * z_0) * b = b * b}\; =: r2}{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{a * z_0 = b \vdash a * (z_0 * b) = b * b}{a * z_0 = b \vdash \exists z.a * z = b * b}\;\exists r}{\exists z.a * z = b \vdash \exists z.a * z = b * b}\;\exists\colon l}{\exists z.a * z = b \vdash D(a, b * b)}\;\mathrm{def}_D\colon r}{D(a, b) \vdash D(a, b * b)}\;\mathrm{def}_D\colon l}{\vdash D(a, b) \to D(a, b * b)}\;\to\colon r}\; =: r1$$

The axioms of the proof are: (1) an instance of the associativity law, (2) the equational axiom $\vdash b * b = b * b$ and the tautological standard axiom $a * z_0 = b \vdash a * z_0 = b$.

3 CERES on LKDe

3.1 Definitions and Results

Though CERES has been defined for **LK** originally, the method is very flexible and can be applied to virtually any sequent calculus for classical logic. Indeed, the extensions defined above, can easily built in without affecting the clarity and efficiency of the method. The central idea of CERES consists in analyzing the proof first, extracting a so-called characteristic clause set from the proof, and then using a resolution refutation of this set to obtain a proof with only atomic cuts. We consider the proofs in **LKDe** as directed trees with nodes which are labelled by sequents, where the root is labelled by the end-sequent. According to the inference rules, we distinguish binary and unary nodes. In an inference

$$\frac{\nu_1\colon S_1 \quad \nu_2\colon S_2}{\nu\colon S}\ x$$

where ν is labelled by S, ν_1 by S_1 and ν_2 by S_2, we call ν_1, ν_2 *predecessors* of ν. Similarly ν' is predecessor of ν in a unary rule if ν' labels the premiss and ν the consequent. Then the *predecessor relation* is defined as the reflexive and transitive closure of the relation above. Every node is predecessor of the root, and the axioms have only themselves as predecessors. For a formal definition of the concepts we refer to [6] and [7]. A similar relation holds between *formula occurrences* in sequents. Instead of a formal definition we give an example.

Consider the rule:

$$\frac{\forall x.P(x) \vdash P(a) \quad \forall x.P(x) \vdash P(b)}{\forall x.P(x) \vdash P(a) \land P(b)}\ \land\colon r$$

The occurrences of $P(a)$ and $P(b)$ in the premiss are *ancestors* of the occurrence of $P(a) \land P(b)$ in the consequent. $P(a)$ and $P(b)$ are called *auxiliary formulas* of the inference, and $P(a) \land P(b)$ the *main formula.* $\forall x.P(x)$ in the premisses are ancestors of $\forall x.P(x)$ in the consequent. Again the *ancestor relation* is defined by reflexive transitive closure.

Let Ω be the set of all occurrences of cut-formulas in sequents of an **LKDe**-proof φ. The cut- formulas are not ancestors of the formulas in the end-sequent, but they might have ancestors in the axioms (if the cuts are not generated by weakening only). The construction of the characteristic clause set is based on the ancestors of the cuts in the axioms. Note that *clauses* are just defined as atomic sequents. We define a set of clauses $\mathcal{C}_\nu$ for every node ν in φ inductively:

- If ν is an occurrence of an axiom sequent $S(\nu)$, and S' is the subsequent of $S(\nu)$ containing only the ancestors of Ω then $\mathcal{C}_\nu = \{S'\}$.
- Let ν' be the predecessor of ν in a unary inference then $\mathcal{C}_\nu = \mathcal{C}_{\nu'}$.
- Let ν_1, ν_2 be the predecessors of ν in a binary inference. We distinguish two cases
  (a) The auxiliary formulas of ν_1, ν_2 are ancestors of Ω. Then

$$\mathcal{C}_\nu = \mathcal{C}_{\nu_1} \cup \mathcal{C}_{\nu_2}.$$

  (b) The auxiliary formulas of ν_1, ν_2 are not ancestors of Ω. Then

$$\mathcal{C}_\nu = \mathcal{C}_{\nu_1} \times \mathcal{C}_{\nu_2}.$$

  where $\mathcal{C} \times \mathcal{D} = \{C \circ D \mid C \in \mathcal{C}, D \in \mathcal{D}\}$ and $C \circ D$ is the merge of the clauses C and D.

The *characteristic clause set* CL(φ) of φ is defined as $\mathcal{C}_{\nu_0}$, where ν_0 is the root. The definition of CL(φ) is the same as the one used for **LK** since both they contain only unary and binary rules. Note that unary rules have no effect on the characteristic clause set.

Theorem 1. *Let φ be a proof in* **LKDe**. *Then the clause set* CL(φ) *is equationally unsatisfiable.*

Remark 1. A clause set $\mathcal{C}$ is equationally unsatisfiable if $\mathcal{C}$ does not have a model where $=$ is interpreted as equality over a domain.

Proof. The proof is essentially the same as in [6]. Let ν be a node in φ and $S'(\nu)$ the subsequent of $S(\nu)$ which consists of the ancestors of Ω (i.e. of a cut). It is shown by induction that $S'(\nu)$ is **LKDe**-derivable from $\mathcal{C}_\nu$. If ν_0 is the root then, clearly, $S'(\nu_0) = \vdash$ and the empty sequent $\vdash$ is **LKDe**-derivable from the axiom set $\mathcal{C}_{\nu_0}$, which is just CL(φ). As all inferences in **LKDe** are sound over equational interpretations (where new symbols introduced by definition introduction have to be interpreted according to the defining equivalence), CL(φ) is equationally unsatisfiable. Note that, without the rules $=: l$ and $=: r$, the set CL(φ) is just unsatisfiable. Clearly the rules $=: l$ and $=: r$ are sound only over equational interpretations. $\diamond$

Note that, for proving Theorem 1, we just need the soundness of **LKDe** not its completeness.

The next steps in CERES are

(1) the computation of the proof projections $\varphi[C]$ w.r.t. clauses $C \in$ CL(φ),
(2) the refutation of the set CL(φ), resulting in an RP-tree γ, i.e. in a deduction tree defined by the inferences of resolution and paramodulation, and
(3) "inserting" the projections $\varphi[C]$ into the leaves of γ.

Step (1) is done like in CERES for **LK**, i.e. we skip in φ all inferences where the auxiliary resp. main formulas are ancestors of a cut. Instead of the end-sequent S we get $S \circ C$ for a $C \in$ CL(φ). The construction does not differ from this in [6] as the form of the rules do not matter.
Step (2) consists in ordinary theorem proving by resolution and paramodulation (which is equationally complete). For refuting CL(φ) any first-order prover like Vampire [2], SPASS [3] or Otter [4] can be used. By the completeness of the methods we find a refutation tree γ as CL(φ) is unsatisfiable by Theorem 1.
Step (3) makes use of the fact that, after computation of the simultaneous most general unifier of the inferences in γ, the resulting tree γ' is actually a *derivation in* **LKDe**! Indeed, after computation of the simultaneous unifier, paramodulation becomes $=: l$ and $=: r$ and resolution becomes cut in **LKDe**. Note that the definition rules, like the logical rules, do not appear in γ'. Now for every leaf

[2] `http://www.vampire.fm/`

[3] `http://spass.mpi-sb.mpg.de/`

[4] `http://www-unix.mcs.anl.gov/AR/otter/`

ν in γ', which is labelled by a clause C' (an instance of a clause $C \in \mathrm{CL}(\varphi)$) we insert the proof projection $\varphi[C']$. The result is a proof with only atomic cuts.

The proof projection is only sound if the proof φ is skolemized, i.e. there are no strong quantifiers (i.e. quantifiers with eigenvariable conditions) in the end-sequent. If φ is not skolemized a priori it can be transformed into a skolemized proof φ' in polynomial (at most quadratic) time; for details see [5].

3.2 An Example

The example below is taken from [16]; it was formalized in **LK** and analyzed by a former version of CERES in the paper [3]. Here we use the extensions by equality rules and definition-introduction to give a simpler formalization and analysis of the proof. The end-sequent formalizes the statement: on a tape with infinitely many cells which are all labelled by 0 or by 1 there are two cells labelled by the same number. $f(x) = 0$ expresses that the cell nr. x is labelled by 0. Indexing of cells is done by number terms defined over $0, 1$ and $+$. The proof φ below uses two lemmas: (1) there are infinitely many cells labelled by 0 and (2) there are infinitely many cells labelled by 1. These lemmas are eliminated by CERES and a more direct argument is obtained in the resulting proof φ'. Ancestors of the cuts in φ are indicated in boldface.

Let φ be the proof

$$\dfrac{\dfrac{\overset{(\tau)}{A \vdash \mathbf{I_0}, \mathbf{I_1}} \quad \overset{(\epsilon_0)}{\mathbf{I_0} \vdash \exists p \exists q(p \neq q \wedge f(p) = f(q))}}{A \vdash \exists p \exists q(p \neq q \wedge f(p) = f(q)), \mathbf{I_1}}\; cut \qquad \overset{(\epsilon_1)}{\mathbf{I_1} \vdash \exists p \exists q(p \neq q \wedge f(p) = f(q))}}{A \vdash \exists p \exists q(p \neq q \wedge f(p) = f(q))}\; cut$$

where $\tau =$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overset{(\tau')}{f(n_0+n_1) = 0 \vee f(n_0+n_1) = 1 \vdash \mathbf{f(n_0+n_1) = 0}, \mathbf{f(n_1+n_0) = 1}}}{\forall x(f(x) = 0 \vee f(x) = 1) \vdash \mathbf{f(n_0+n_1) = 0}, \mathbf{f(n_1+n_0) = 1}}\; \forall{:}\,l}{A \vdash \mathbf{f(n_0+n_1) = 0}, \mathbf{f(n_1+n_0) = 1}}\; \mathrm{def}_A{:}\,l}{A \vdash \mathbf{f(n_0+n_1) = 0}, \exists \mathbf{k}.\mathbf{f(n_1+k) = 1}}\; \exists r}{A \vdash \exists \mathbf{k}.\mathbf{f(n_0+k) = 0}, \exists \mathbf{k}.\mathbf{f(n_1+k) = 1}}\; \exists r}{A \vdash \exists \mathbf{k}.\mathbf{f(n_0+k) = 0}, \forall \mathbf{n} \exists \mathbf{k}.\mathbf{f(n+k) = 1}}\; \forall{:}\,r}{A \vdash \forall \mathbf{n} \exists \mathbf{k}.\mathbf{f(n+k) = 0}, \forall \mathbf{n} \exists \mathbf{k}.\mathbf{f(n+k) = 1}}\; \forall{:}\,r}{A \vdash \mathbf{I_0}, \forall \mathbf{n} \exists \mathbf{k}.\mathbf{f(n+k) = 1}}\; \mathrm{def}_{I_0}{:}\,r}{A \vdash \mathbf{I_0}, \mathbf{I_1}}\; \mathrm{def}_{I_1}{:}\,r$$

For $\tau' =$

$$\dfrac{f(n_0+n_1) = 0 \vdash \mathbf{f(n_0+n_1) = 0} \qquad \dfrac{\overset{(\text{Axiom})}{\vdash n_1 + n_0 = n_0 + n_1} \quad f(n_1+n_0) = 1 \vdash \mathbf{f(n_1+n_0) = 1}}{f(n_0+n_1) = 1 \vdash \mathbf{f(n_1+n_0) = 1}}\; =:l1}{f(n_0+n_1) = 0 \vee f(n_0+n_1) = 1 \vdash \mathbf{f(n_0+n_1) = 0}, \mathbf{f(n_1+n_0) = 1}}\; \vee{:}\,l$$

And for $i = 1, 2$ we define the proofs $\epsilon_i =$

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\psi \quad \eta_i}{\mathbf{f}(\mathbf{s}) = \mathbf{i}, \mathbf{f}(\mathbf{t}) = \mathbf{i} \vdash s \neq t \wedge f(s) = f(t)}\ \wedge{:}\,r}{\mathbf{f}(\mathbf{s}) = \mathbf{i}, \mathbf{f}(\mathbf{t}) = \mathbf{i} \vdash \exists q(s \neq q \wedge f(s) = f(q))}\ \exists{:}\,r}{\mathbf{f}(\mathbf{s}) = \mathbf{i}, \mathbf{f}(\mathbf{t}) = \mathbf{i} \vdash \exists p \exists q(p \neq q \wedge f(p) = f(q))}\ \exists{:}\,r}{\mathbf{f}(\mathbf{n_0} + \mathbf{k_0}) = \mathbf{i}, \exists \mathbf{k}.\mathbf{f}(((\mathbf{n_0} + \mathbf{k_0}) + \mathbf{1}) + \mathbf{k}) = \mathbf{i} \vdash \exists p \exists q(p \neq q \wedge f(p) = f(q))}\ \exists{:}\,l}{\mathbf{f}(\mathbf{n_0} + \mathbf{k_0}) = \mathbf{i}, \forall \mathbf{n} \exists \mathbf{k}.\mathbf{f}(\mathbf{n} + \mathbf{k}) = \mathbf{i} \vdash \exists p \exists q(p \neq q \wedge f(p) = f(q))}\ \forall{:}\,l}{\exists \mathbf{k}.\mathbf{f}(\mathbf{n_0} + \mathbf{k}) = \mathbf{i}, \forall \mathbf{n} \exists \mathbf{k}.\mathbf{f}(\mathbf{n} + \mathbf{k}) = \mathbf{i} \vdash \exists p \exists q(p \neq q \wedge f(p) = f(q))}\ \exists{:}\,l}{\forall \mathbf{n} \exists \mathbf{k}.\mathbf{f}(\mathbf{n} + \mathbf{k}) = \mathbf{i}, \forall \mathbf{n} \exists \mathbf{k}.\mathbf{f}(\mathbf{n} + \mathbf{k}) = \mathbf{i} \vdash \exists p \exists q(p \neq q \wedge f(p) = f(q))}\ \forall{:}\,l}{\forall \mathbf{n} \exists \mathbf{k}.\mathbf{f}(\mathbf{n} + \mathbf{k}) = \mathbf{i} \vdash \exists p \exists q(p \neq q \wedge f(p) = f(q))}\ c{:}\,l}{\mathbf{I_i} \vdash \exists p \exists q(p \neq q \wedge f(p) = f(q))}\ \mathrm{def}_{I_i}{:}\,l$$

for $s = n_0 + k_0$, $t = ((n_0 + k_0) + 1) + k_1$, and the proofs

$$\cfrac{\cfrac{\cfrac{\text{(axiom)}}{\vdash (n_0 + k_0) + (1 + k_1) = ((n_0 + k_0) + 1) + k_1} \quad \cfrac{\text{(axiom)}}{n_0 + k_0 = (n_0 + k_0) + (1 + k_1) \vdash}}{n_0 + k_0 = ((n_0 + k_0) + 1) + k_1 \vdash}\ =:l1}{\vdash n_0 + k_0 \neq ((n_0 + k_0) + 1) + k_1}\ \neg{:}\,r$$

and $\eta_i =$

$$\cfrac{\mathbf{f}(\mathbf{s}) = \mathbf{i} \vdash f(s) = i \quad \cfrac{\mathbf{f}(\mathbf{t}) = \mathbf{i} \vdash f(t) = i \quad \cfrac{\text{(axiom)}}{\vdash i = i}}{\mathbf{f}(\mathbf{t}) = \mathbf{i} \vdash i = f(t)}\ =:r2}{\mathbf{f}(\mathbf{s}) = \mathbf{i}, \mathbf{f}(\mathbf{t}) = \mathbf{i} \vdash f(s) = f(t)}\ =:r2$$

The characteristic clause set is (after variable renaming)

$$\begin{aligned}\mathrm{CL}(\varphi) = \{ &\vdash f(x + y) = 0, f(y + x) = 1; \quad (C_1)\\ &f(x + y) = 0, f(((x + y) + 1) + z) = 0 \vdash; \quad (C_2)\\ &f(x + y) = 1, f(((x + y) + 1) + z) = 1 \vdash\} \quad (C_3).\end{aligned}$$

The axioms used for the proof are the standard axioms of type $A \vdash A$ and instances of $\vdash x = x$, of commutativity $\vdash x + y = y + x$, of associativity $\vdash (x + y) + z = x + (y + z)$, and of the axiom

$$x = x + (1 + y) \vdash,$$

expressing that $x + (1 + y) \neq x$ for all natural numbers x, y.

The comparison with the analysis of Urban's proof formulated in **LK** (without equality) [3] shows that this one is much more transparent. In fact the set of characteristic clauses contains only 3 clauses (instead of 5), which are also simpler. This also facilitates the refutation of the clause set and makes the output proof simpler and more transparent. On the other hand, the analysis below shows that the mathematical argument obtained by cut-elimination is the same as in [3].

The program Otter found the following refutation of $\mathrm{CL}(\varphi)$ (based on hyperresolution only – without equality inference):

The first hyperesolvent, based on the clash sequence $(C_2; C_1, C_1)$, is

$$\begin{aligned} C_4 = &\ \vdash f(y+x)=1,\ f(z+((x+y)+1))=1, \text{ with the intermediary clause}\\ D_1 = &\ f(((x+y)+1)+z)=0 \vdash f(y+x)=1. \end{aligned}$$

The next clash is sequence is $(C_3; C_4, C_4)$ which gives C_5 with intermediary clause D_2, where:

$$\begin{aligned} C_5 = &\ \vdash f(v'+u')=1,\ f(v+u)=1,\\ D_2 = &\ f(x+y)=1 \vdash f(v+u)=1. \end{aligned}$$

Factoring C_5 gives $C_6\colon \vdash f(v+u)=1$ (which roughly expresses that all fields are labelled by 1). The final clash sequence $(C_3; C_6, C_6)$ obviously results in the empty clause $\vdash$ with intermediary clause $D_3\colon f(((x+y)+1)+z)=1 \vdash$. The hyperresolution proof ψ_3 in form of a tree can be obtained from the following resolution trees, where C' and ψ' stand for renamed variants of C and of ψ, respectively:

$$\dfrac{\dfrac{C_1 \quad C_2}{D_1}\ \mathit{res} \qquad C_1'}{\psi_1\colon\ C_4}\ \mathit{res} \qquad \dfrac{\dfrac{\dfrac{C_3' \quad \psi_1}{D_2}\ \mathit{res} \qquad \psi_1'}{C_5}\ \mathit{res}}{\psi_2\colon\ C_6}\ \mathit{factor} \qquad \dfrac{\dfrac{\psi_2 \quad C_3''}{D_3}\ \mathit{res} \qquad \psi_2'}{\psi_3\colon\ \vdash}\ \mathit{res}$$

Instantiation of ψ_3 by the uniform most general unifier σ of all resolutions gives a deduction tree $\psi_3\sigma$ in **LKDe**; indeed, after application of σ, resolution becomes cut and factoring becomes contraction. The proof $\psi_3\sigma$ is the skeleton of an **LKDe**-proof of the end-sequent with only atomic cuts. Then the leaves of the tree $\psi_3\sigma$ have to be replaced by the proof projections. E.g., the clause C_1 is replaced by the proof $\varphi[C_1]$, where $s = n_0 + n_1$ and $t = n_1 + n_0$:

$$\dfrac{\dfrac{\dfrac{\dfrac{f(s)=0 \vdash f(s)=0 \qquad \dfrac{\overset{\text{(Axiom)}}{\vdash t=s} \quad f(t)=1 \vdash f(t)=1}{f(s)=1 \vdash f(t)=1}\ =\colon l1}{f(s)=0 \lor f(s)=1 \vdash f(s)=0, f(t)=1}\ \lor\colon l}{\forall x(f(x)=0 \lor f(x)=1) \vdash f(s)=0, f(t)=1}\ \forall\colon l}{A \vdash f(s)=0, f(t)=1}\ \mathrm{def}_A\colon l}{A \vdash \exists p \exists q(p \neq q \land f(p)=f(q)), f(s)=0, f(t)=1}\ w\colon r$$

Furthermore C_2 is replaced by the projection $\varphi[C_2]$ and C_3 by $\varphi[C_3]$, where (for $i = 0, 1$) $\varphi[C_{2+i}] =$

$$\cfrac{\cfrac{\cfrac{\cfrac{\psi \quad \eta_i}{f(s) = i, f(t) = i \vdash s \neq t \land f(s) = f(t)}\ \land\colon r}{f(s) = i, f(t) = i \vdash \exists q(s \neq q \land f(s) = f(q))}\ \exists\colon r}{f(s) = i, f(t) = i \vdash \exists p \exists q(p \neq q \land f(p) = f(q))}\ \exists\colon r}{f(s) = i, f(t) = i, A \vdash \exists p \exists q(p \neq q \land f(p) = f(q))}\ w\colon l$$

Note that ψ, η_0, η_1 are the same as in the definition of ϵ_0, ϵ_1 above.

By inserting the σ-instances of the projections into the resolution proof $\psi_3\sigma$ and performing some additional contractions, we eventually obtain the desired proof φ' of the end-sequent

$$A \vdash \exists p \exists q(p \neq q \land f(p) = f(q))$$

with only *atomic* cuts. φ' no longer uses the lemmas that infinitely many cells are labelled by 0 and by 1, respectively.

4 The System CERES

The cut-elimination system CERES is written in ANSI-C++. There are two main tasks. On one hand, to compute an unsatisfiable set of clauses characterizing the cut formulas. This is done by automatically extracting the so-called characteristic clause term from a proof φ and computing the resulting *characteristic clause set* $\mathrm{CL}(\varphi)$. On the other hand, to generate a resolution refutation of $\mathrm{CL}(\varphi)$ by an external theorem prover[5], and to compute the necessary projection schemes of φ w.r.t. the clauses in $\mathrm{CL}(\varphi)$ actually used for the refutation. The properly instantiated projection schemes are concatenated, using the refutation obtained by the theorem prover as a skeleton of a proof with only atomic cuts.

Concerning the extension of **LK** to **LKDe**, equality rules appearing within the input proof are propagated to the projection schemes as any other binary rules. During theorem proving equality is treated by paramodulation (which is closely related to the equality rules in **LKDe**); its application within the final clausal refutation is then transformed to the appropriate equality rules in **LKDe**. The definition introductions do not require any other special treatment within CERES than all other unary rules; in particular, they have no influence on the theorem proving part.

Since the restriction to skolemized proofs is crucial to the CERES-method, the system also performs skolemization (according to Andrew's method [2]) on the input proof.

The system CERES expects an **LKDe** proof of a sequent S and a set of axioms as input, and computes a proof of S containing at most non atomic-cuts.

[5] The current version of CERES uses the automated theorem prover Otter (see http://www-unix.mcs.anl.gov/AR/otter/), but any refutational theorem prover based on resolution and paramodulation may be used.

Input and output are formatted using the well known data representation language XML. This allows the use of arbitrary and well known utilities for editing, transformation and presentation and standardized programming libraries. To increase the performance and avoid redundancy, most parts of the proofs are internally represented as directed acyclic graphs. This representation turns out to be very handy, also for the internal unification algorithms.

The formal analysis of mathematical proofs (especially by a mathematician as a pre- and post-"processor") relies on a suitable format for the input and output of proofs, and on an appropriate aid in dealing with them. We developed an intermediary proof language connecting the language of mathematical proofs with **LKDe**. Furthermore we implemented a proof viewer and proof editor with a graphical user interface, allowing a convenient input and analysis of the output of `CERES`. Thereby the integration of definition- and equality-rules into the underlying calculus plays an essential role in overlooking, understanding and analyzing complex mathematical proofs by humans.

5 Future Work

We plan to develop the following extensions of CERES:

- As the cut-free proofs are often very large and difficult to interpret, we intend to provide the possibility to analyze certain characteristics of the cut-free proof (which are simpler than the proof itself). An important example are Herbrand sequents which may serve to extract bounds from proofs (see e.g. [11]). We plan to develop algorithms for extracting Herbrand sequents (also from proofs of nonprenex sequents as indicated in [4]) and for computing interpolants.
- A great challenge in the formal analysis of mathematical proofs lies in providing a suitable format for the input and output of proofs. We plan to improve our intermediary proof language and to move closer to the "natural" language of mathematical proofs.
- In the present version CERES eliminates all cuts at once. But — for the application to real mathematical proofs — only interesting cuts (i.e. lemmas) deserve to be eliminated, others should simply remain or be integrated as additional axioms.
- As CERES requires the skolemization of the end-sequent the original proof must be transformed to skolem form. We plan to develop an efficient *deskolemization*-algorithm, which transforms the theorem to be proved into its original form.

To demonstrate the abilities of `CERES` and the feasibility of formalizing and analyzing complex proofs of mathematical relevance, we currently investigate a well known proof of the infinity of primes using topology (which may be found in [1]). Our aim is to eliminate the topological concepts from the proof by means of CERES, breaking it down to a proof solely based on elementary number arithmetic.

References

1. M. Aigner, G. M. Ziegler. *Proofs from THE BOOK*. Springer 1998.
2. P. B. Andrews: Resolution in Type Theory, *Journal of Symbolic Logic*, 36, pp.414–432, 1971.
3. M. Baaz, S. Hetzl, A. Leitsch, C. Richter, H. Spohr: Cut-Elimination: Experiments with CERES, LPAR 2004, *Lecture Notes in Artificial Intelligence*, pp. 481-495, 2005.
4. M. Baaz, A. Leitsch: On skolemization and proof complexity, *Fundamenta Informaticae*, 20(4), pp. 353–379, 1994.
5. M. Baaz, A. Leitsch: Cut normal forms and proof complexity, *Annals of Pure and Applied Logic*, 97, pp. 127-177, 1999.
6. M. Baaz, A. Leitsch: Cut-Elimination and Redundancy-Elimination by Resolution, *Journal of Symbolic Computation*, 29, pp. 149-176, 2000.
7. M. Baaz, A. Leitsch: Towards a Clausal Analysis of Cut-Elimination, *Journal of Symbolic Computation*, 41, pp. 381–410, 2006.
8. E. Eder: Relative complexities of first-order calculi, Vieweg, 1992.
9. G. Gentzen: Untersuchungen über das logische Schließen, *Mathematische Zeitschrift*, 39, pp. 405–431, 1934–1935.
10. J.Y. Girard: Proof Theory and Logical Complexity, in *Studies in Proof Theory*, Bibliopolis, Napoli, 1987.
11. H. Luckhardt: Herbrand-Analysen zweier Beweise des Satzes von Roth: polynomiale Anzahlschranken. *The Journal of Symbolic Logic* 54, 234–263, 1989.
12. R. Nieuwenhuis, A. Rubio: Paramodulation-based Theorem Proving, in *Handbook of Automated Reasoning*, eds. J.A. Robinson, A. Voronkov, pp. 371–443, Elsevier, 2001.
13. G. Polya. *Mathematics and plausible reasoning, Volume I: Induction and Analogy in Mathematics.* Princeton University Press, Princeton, New Jersey, 1954.
14. G. Polya. *Mathematics and plausible reasoning, Volume II: Patterns of Plausible Inference.* Princeton University Press, Princeton, New Jersey, 1954.
15. G. Takeuti: Proof Theory, North-Holland, Amsterdam, 2nd edition, 1987.
16. C. Urban: *Classical Logic and Computation* Ph.D. Thesis, University of Cambridge Computer Laboratory, 2000.
17. A. Degtyarev and A. Voronkov: Equality Reasoning in Sequent-Based Calculi *Handbook of Automated Reasoning* vol. I, ed. by A. Robinson and A. Voronkov, chapter 10, pp. 611-706, Elsevier Science, 2001.

Synthesizing Proof Planning Methods and Ω-Ants Agents from Mathematical Knowledge

Serge Autexier[1,2] and Dominik Dietrich[2]

[1] German Research Center for Artificial Intelligence (DFKI GmbH),
Saarbrücken, Germany
autexier@dfki.de
[2] FR 6.2 Informatik, Saarland University, Saarbrücken, Germany
{autexier, dodi}@ags.uni-sb.de

Abstract. In this paper we investigate how to extract proof procedural information contained in declarative representations of mathematical knowledge, such as axioms, definitions, lemmas and theorems (collectively called *assertions*) and how to effectively include it into automated proof search techniques. In the context of the proof planner MULTI and the agent-based reasoning system Ω-ANTS, we present techniques to automatically synthesize proof planning methods and Ω-ANTS-agents from assertions such that they can be *actively* used by these systems. This in turn enables a user to effectively use these systems without having to know the peculiarities of coding methods and agents.

1 Introduction

The development of the proof assistant system ΩMEGA is one of the major attempts to build an all encompassing assistance tool for the working mathematician or for the formal work of a software engineer. It is a representative of systems in the paradigm of *proof planning* and combines interactive and automated proof construction for domains with rich and well-structured mathematical knowledge. The search for a proof is usually conducted at a high level of granularity defined by *tactics* and *methods*. Automation of proof search at this 'abstract' level is realized in two components: The multi-strategy proof planner MULTI [13] and the resource-guided agent-based reasoning system Ω-ANTS [4,15]. MULTI integrates a basic set of algorithms parameterized over strategic information. Among others, it includes a best-first proof planning algorithm that is parameterized over methods and control rules and which searches through the space of applicable methods by using the heuristic function defined by the provided control knowledge. The Ω-ANTS-system is based on encapsulations of calculus rules, tactics, external system calls and methods into pro-active agents which automatically check for their own applicability. Each of these agents is associated with a set of *argument agents* for formal arguments of the encapsulated procedure that compute possible instantiations for the associated formal argument subject to existing instantiations for other formal arguments.

The tactics and methods encode specific proof knowledge, such as, for instance, when and how to perform a proof by case analysis, to use a diagonalization

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 94–109, 2006.

argument, to call a computer algebra system or an automated theorem prover, but also axioms, lemmas or theorems. The latter are typically contained in the mathematical theories but needed to be reformulated manually as tactics, methods and agents in order to enable Multi and Ω-Ants to *actively* use them during proof search. Hence a human user has to know the peculiarities of coding tactics, methods and agents for these systems in order to be able to make effective use of their automated proof search capabilities. In this paper we aim at remedying this situation. Based on a notion of *inferences* integrating the so far separated notions of tactics and methods, (1) we propose a technique to synthesize inferences from axioms, lemmas and theorems, (2) we define proof planning directly at the level of arbitrary, i.e. both the synthesized *and* the user-defined inferences, and (3) provide a mechanism to automatically generate an optimal set of argument agents from inferences.

The paper is organized as follows: In Sec. 2 we set the context of this work by describing the tactics, methods and Ω-Ants-agents of the old Ωmega system. In Sec. 3 we introduce *inferences* as one major means for proof construction of the so-called task layer in the new Ωmega-system. In Sec. 4 we present a technique to synthesize *inferences* from axioms, definitions, lemmas, and theorems contained in mathematical theories. A procedure to determine the possible directions in which an inference can be applied is presented in Sec. 5 and the computation of optimal sets of Ω-Ants-agents is given in Sec. 6. We discuss related works in Sec. 7 before concluding the paper with a summary of the results in Sec. 8.

2 Tactics, Methods, Ω-Ants

We briefly describe the context of the work presented in this paper by introducing tactics, methods and Ω-Ants as they exist in the old Ωmega system. Throughout this section by Ωmega we mean the old Ωmega system [5].

Tactics. Ωmega provides the definition of tactics as a means for proof construction at an abstract level. These tactics are comparable to standard LCF-style tactics [12], that encode repeatedly occurring sequences of calculus steps combined by so-called *tacticals* such as *repeat*, *then*, *or*. Intuitively we can see a tactic as a program which performs a certain task. It is executed when the tactic is invoked and returns a verifiable proof object if it terminates. In Ωmega a tactic requires certain assumptions and open goals to be present in the current proof state in order to be applicable: these are, respectively, the premises $P_1, \ldots, P_k$ and conclusions $C_1, \ldots, C_n$ of the tactic. In practice, however, there are classes of tactics which essentially coincide, and only slightly differ in the premises and conclusions they require. The reason for this is that a tactic with input $P_1, \ldots, P_n, C_1, \ldots, C_k$ can often also be applied with input $P_1, \ldots, P_{n-1}, C_1, \ldots, C_k$ only, but then introduces P_n as additional subgoal. It is clear that all these cases could in principle be combined in one tactic, but this was to cumbersome to do in Ωmega's tactics specification formalism and hence was never done.

Methods. A method is a declarative specification of a tactic that describes how an application of the tactic modifies a given proof state. The underlying idea is to be able to transform a proof state without the need to execute the tactic. Given such a specification, it must contain information stating which premises and conclusions are to be added to the proof state and which are to be removed during method application. In ΩMEGA premises and conclusions can be annotated with flags $\oplus$ and $\ominus$: those annotated with $\oplus$ will be added when applying the method, those annotate with $\ominus$ shall be removed and those without annotations must be present before and after application of the method.

In general there is no guarantee that the specification really describes the tactic to which it belongs; hence methods can be unsound. To check the validity of a method proof step, the method must be expanded, that is the tactic attached to the method must be executed and return a verifiable proof object. Proof search using methods is called *proof planning* [6]. In contrast to other proof planning systems, such as λ-CLAM [14], ΩMEGA methods do not contain heuristic knowledge which could also be encoded in the specification.

Ω-ANTS. The Ω-ANTS-system was originally developed to support a user in an interactive theorem proving environment by distributively searching via agents for possible next proof steps [4,15] and was later extended to a fully automated system. Conceptually the system consists of two kinds of agents: *command agents* and *argument agents.* Each command agent is associated to a tactic, method or calculus rule, uniformly called rule, and orders suggestions for the associated rule according to some heuristics. These suggestions are generated by a society of argument agents which are assigned to a command agent. From the perspective of this paper we are only interested in finding suggestions for a particular rule and thus only describe argument agents in more detail.

The goal of a society of argument agents consists of generating suggestions of how a particular rule can be applied in the current proof situation. Such a rule consists of premises, conclusions and a set of additional parameters, called arguments. Starting from the situation in which no argument is instantiated, the agents instantiate these arguments stepwise, resulting in so-called *partial argument instantiations.* If sufficiently many arguments are instantiated, the rule can be applied. Intuitively an argument agent is responsible for one or more specific arguments – stored in the *goal set.* Given a set of already computed partial argument instantiations it checks whether it can add an instance for those arguments in its goal set. Usually an argument agent requires that certain formal arguments are already specified. These are stored in the so-called *dependency set.*

Discussion. Whereas in principle it is beneficial to have tools for interactive and automated proof search which can be combined to find a proof, for a ΩMEGA user this means that before starting a proof he has the burden of specifying tactics for interactive proof construction, corresponding methods for the proof planner MULTI and corresponding agents for proof construction with the Ω-ANTS-system. In particular he had to separately specify very similar tactics (and thus methods) as described above. Usually an essential subset of these tactics was a formulation of an axiom of the theory in which a proof was constructed. In this paper we show how the workload of a user can be drastically reduced:

First, we introduce the notion of an *inference* which unifies the previously separated notions of tactics and methods. In particular we develop a mechanism to synthesize inferences from axioms, lemmas and theorems. Furthermore we present mechanisms to generate from individual inferences all tactics within a tactic class and an optimal set of argument agents for each tactic.

3 Task Layer

The ΩMEGA-system is currently under re-development where, among others, the underlying natural deduction calculus is being replaced with the CoRE-calculus [1]. The task layer [7] is an instance of the new proof datastructure (PDS) [2] and is the uniform proof construction interface used by both the human user and the automated proof search procedures MULTI and Ω-ANTS. The nodes of the PDS are annotated with *tasks*, which are Gentzen-style multi-conclusion sequents augmented by means to define multiple foci of attention on subformulas that are maintained during the proof. Each task is reduced to a possibly empty set of subtasks by one of the following proof construction steps: (1) the introduction of a proof sketch [17][1], (2) deep structural rules for weakening and decomposition of subformulas, (3) the application of a lemma that can be postulated on the fly, (4) the substitution of meta-variables, and (5) the application of an inference.

Due to the presence of meta-variables, substitutions and alternative proof steps, the task layer extends the PDS from [2] by a mechanism to deal with substitutions of the same meta-variable in alternative subproofs (see [7] for details).

The logic used in ΩMEGA is a simply-typed higher-order logic with arbitrarily many base types, the usual β-reduction and η-expansion rules and we consider all terms to be always in $\beta\eta$ long normal form, which is unique up to renaming of bound variables (α-equal) (see [3] for details).

For the purposes of this paper we assume $T^{\mathcal{X}}_{\Sigma,\mathcal{V}}$ are the terms built over a signature Σ, typed variables $\mathcal{V}$ and typed meta-variables $\mathcal{X}$ ($\mathcal{V} \cap \mathcal{X} = \emptyset$) and $\mathit{wff}^{\mathcal{X}}_{\Sigma,\mathcal{V}}$ is the set of terms of boolean type, also denoted as *formulas*: as usual quantifiers and λ-binders can only bind variables and we require the formulas from $\mathit{wff}^{\mathcal{X}}_{\Sigma,\mathcal{V}}$ to contain no free variables.

Each subterm of a term t can be uniquely qualified by its position π in the term which we denote by $t_{|\pi}$. The type compliant replacement of a subterm $t_{|\pi}$ by a term s is denoted by $t_{|\pi\leftarrow s}$. The set of all positions is denoted by $\mathcal{POS}$ and $\mathcal{POS}(t)$ denotes the set of all valid positions of the term t.

A *substitution* is a type preserving and idempotent function $\sigma : \mathcal{V} \cup \mathcal{X} \rightarrow T^{\mathcal{X}}_{\Sigma,\mathcal{V}}$ that is the identity function but for finitely many elements from $\mathcal{V} \cup \mathcal{X}$. This allows for a finite representation of σ as $\{\sigma(x_1)/x_1, \ldots, \sigma(x_n)/x_n\}$ where $\sigma(y) = y$ if $\forall 1 \leq i \leq n, y \neq x_i$. The *domain of* σ is $dom(\sigma) := \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ and we denote by *meta-variable substitution* those substitutions σ where $dom(\sigma) \subset \mathcal{X}$. As usual we do not distinguish between a substitution and its homomorphic extension to terms.

[1] In the old ΩMEGA system this was realized by using so-called *Island*-methods.

Each subformula occurs either negatively or positively in a formula and we can assign a positive or negative polarity to the subformula subject to the polarity of the entire formula. Following [10], polarized propositional formulas can be classified into α-type and β-type formulas. We say that two subformulas F and G of some polarized formula are *α-related* (resp. *β-related*), if the smallest subformula containing both F and G is of type α (resp. β).

For the purpose of this paper we consider tasks as multi-conclusion sequents $F_1, \ldots, F_k \vdash G_1, \ldots, G_l$, where F_i and G_j are from $\mathit{wff}^{\mathcal{X}}_{\Sigma,\mathcal{V}}$. The notions of substitution and positions of formulas carry over to sequents in the obvious way, and we define the F_i to have negative polarity and the G_j to have positive polarity.

3.1 Inferences

Intuitively, an *inference* is a proof step with multiple premises and conclusions augmented by (1) a possibly empty set of hypotheses for each premise, (2) a set of *application conditions* that must be fulfilled upon inference application, (3) a set of *outline functions* that can compute the values of premises and conclusions from values of other premises and conclusions, and (4) an *expansion function* that refines the abstract inference step. Each premise and conclusion consists of a unique name and a formula scheme from $\mathit{wff}^{\mathcal{X}}_{\Sigma,\mathcal{V}}$. Note that we employ the term *inference* in its general meaning; Taken in that sense, an inference can be either valid or invalid in contrast to the formal logic notion of an *inference rule*.

Additional information needed in the application conditions or the outline functions, such as, for instance, the position of a subterm or the instance of some non-boolean meta-variable, can be specified by additional *parameters* to the inference. Since this can be arbitrary information, we refrain from a formal definition, but assume that we can check the admissibility of a substitution of the formal parameters. The parameters of an inference, the names of premises and conclusions and the meta-variables that occur in the associated formulas form the *variables of the inference*.

Definition 1 (Inference Variables & Inference Substitutions). *The pairwise disjoint sets $\mathcal{P}$ of parameter variables, $\mathcal{N}$ of names for premises and conclusions and $\mathcal{X}$ of meta-variables are the* inference variables*. Let further* $\boldsymbol{Val}$ *denote the possible values for parameters and T be a task. An* inference substitution wrt. *T is a triple $\sigma := \langle \sigma_{\mathcal{P}}, \sigma_{\mathcal{N}}, \sigma_{\mathcal{X}} \rangle$, where (1) $\sigma_{\mathcal{P}} : \mathcal{P} \to \boldsymbol{Val} \cup \{\bot\}$ is an admissible* parameter substitution*, (2) $\sigma_{\mathcal{N}} : \mathcal{N} \to \mathcal{POS}(T) \cup \mathit{wff}^{\mathcal{X}}_{\Sigma,\mathcal{V}} \cup \{\bot\}$ is a* name substitution*, and (3) $\sigma_{\mathcal{X}}$ is a meta-variable substitution.*

The domain of the parameter substitutions $dom(\sigma_{\mathcal{P}})$ is the largest subset of $\mathcal{P}$ on which the value of $\sigma_{\mathcal{P}}$ is not $\bot$. We define $dom(\sigma_{\mathcal{N}})$ analogously and define $dom(\sigma) := dom(\sigma_{\mathcal{P}}) \cup dom(\sigma_{\mathcal{N}}) \cup dom(\sigma_{\mathcal{X}})$. Furthermore, we define $\sigma^{\#}$ to denote the function which returns $\sigma^{\#}(i) := \sigma_{\mathcal{X}}(T_{|\sigma_{\mathcal{N}}(i)})$ if $i \in \mathcal{N}$ and $\sigma_{\mathcal{N}}(i) \in \mathcal{POS}(T)$, and otherwise behaves like σ.

We say that an inference substitution $\langle \sigma_{\mathcal{P}}, \sigma_{\mathcal{N}}, \sigma_{\mathcal{X}} \rangle$ is *more general* than the inference substitution $\langle \tau_{\mathcal{P}}, \tau_{\mathcal{N}}, \tau_{\mathcal{X}} \rangle$, iff (1) there exists a substitution ρ such that

$$\frac{\begin{array}{ccc}[\mathsf{Hyp}(p_1)] & & [\mathsf{Hyp}(p_k)]\\ \vdots & & \vdots \\ p_1 : \mathsf{FS}(p_1) & \ldots & p_k : \mathsf{FS}(p_k)\end{array}}{c_1 : \mathsf{FS}(c_1) \;\ldots\; c_m : \mathsf{FS}(c_m)}\, \mathit{Name}(\omega_1,\ldots,\omega_n)$$

$$\begin{array}{ll}\textbf{Appl. Cond.:} & P(\boldsymbol{i^0_k})\\ \textbf{Outline:} & \langle i^1, f^1(\boldsymbol{i^1_{j_1}})\rangle \ldots \langle i^l, f^l(\boldsymbol{i^l_{j_l}})\rangle\end{array}$$

Fig. 1. Graphical representation of an Inference

$\tau_{\mathcal{X}} = \rho \circ \sigma_{\mathcal{X}}$, (2) for all $\omega \in dom(\sigma_{\mathcal{P}})$ it holds $\tau_{\mathcal{P}}(\omega) = \sigma_{\mathcal{P}}(\omega)$, and (3) for all $I \in dom(\sigma_{\mathcal{N}})$ it holds $\sigma_{\mathcal{N}}(I) = \tau_{\mathcal{N}}(I)$ and $\tau^{\#}_{\mathcal{N}}(I) = \rho(\sigma^{\#}_{\mathcal{N}}(I))$.

Application conditions are predicates on the values of inference variables and outline functions compute values for specific inference variables out of values of other inference variables.

Definition 2 (Application Conditions & Outline Functions). *Assume the inference variables $i_0, \ldots, i_n \subset \mathcal{P} \cup \mathcal{N} \cup \mathcal{X}$, P a predicate and f a function on $\mathcal{POS} \cup$ **Val**, and σ an inference substitution wrt. some task T. An expression $P(i_1, \ldots, i_n)$ is an* application condition *which holds for σ iff $\{i_1, \ldots, i_n\} \subset dom(\sigma)$ and $P(\sigma^{\#}(i_1), \ldots, \sigma^{\#}(i_n))$ holds. An* outline function *is an expression $\langle i_0, f(i_1, \ldots, i_n)\rangle$ if the values computed by f are admissible for i_0. The outline function is applicable, iff $\{i_1, \ldots, i_n\} \subset dom(\sigma)$ and $i_0 \notin dom(\sigma)$. The extended inference substitution computed by the outline function $\langle i_0, f(i_1, \ldots, i_n)\rangle$ applicable for σ is $\sigma \oplus \langle i_0, f(i_1, \ldots, i_n)\rangle$ defined by*

$$\sigma \oplus \langle i_0, f(i_1, \ldots, i_n)\rangle(x) := \begin{cases} f(\sigma^{\#}(i_1), \ldots, \sigma^{\#}(i_n)) & \textit{if } x = i_0 \\ \sigma(x) & \textit{otherwise} \end{cases}$$

Note that the operator $\oplus$ associates to the left and is only defined if the outline function is applicable. We now formally define *inferences* and we use $\mathcal{P}(N)$ to denote the power set of the set N.

Definition 3 (Inference). *Let $P = \{p_1, \ldots, p_k\}$, $C = \{c_1, \ldots, c_m\}$ be disjoint sets of names, $\mathsf{Hyp} : P \to \mathcal{P}(\mathit{wff}^{\mathcal{X}}_{\Sigma,\mathcal{V}})$ a function which associates a set of hypotheses to each premise, $\mathsf{FS} : P \cup C \to \mathit{wff}^{\mathcal{X}}_{\Sigma,\mathcal{V}}$ the function that associates a formula scheme to each premise and conclusion, $\Omega = \{\omega_1, \ldots, \omega_l\}$ be a set of parameters, $P(\boldsymbol{i^0_k})$ an application condition and $OF = \{\langle i^1, f^1(\boldsymbol{i^1_{j_1}})\rangle, \ldots, \langle i^l, f^l(\boldsymbol{i^l_{j_l}})\rangle\}$ outline functions, and Exp an expansion function. Then $\langle P, C, \mathsf{Hyp}, \mathsf{FS}, \Omega, P(\boldsymbol{i^0_k}), OF, \mathsf{Exp}\rangle$ is an* inference.

The graphical representation of an inference is given in Fig. 1. Note that there can be more than one outline function for the same inference variable. We say that $P \cup C$ are the *formal arguments* of **I** and the *variables of* **I** are the elements of $P \cup C \cup \Omega$ and the meta-variables occurring in the formula and hypotheses associated with the formal arguments.

An instance of an inference is given in Fig. 2. The inference *subst-m* has two premises p_1,p_2 with formula schemes F and $U = V$ respectively, one conclusion

$$\frac{p_1 : F \quad p_2 : U = V}{c : G}\,\mathit{subst\text{-}m}(\pi)$$

Appl. Cond.: $(F|_\pi = U \wedge G|_{\pi \leftarrow V} = F) \vee (G|_\pi = U \wedge F|_{\pi \leftarrow V} = G)$
Outline: $\langle c, \texttt{compute-subst-m}(p_1, p_2, \pi)\rangle$
$\langle p_1, \texttt{compute-subst-m}(p_2, c, \pi)\rangle$
$\langle \pi, \texttt{compute-pos}\ (p_1, p_2)\rangle$
$\langle \pi, \texttt{compute-pos}\ (p_2, c)\rangle$

Fig. 2. Inference *subst-m*

c with formula scheme G, and one parameter π. It represents the deduction step that if we are given a formula F in which at position π the term U occurs and we are given that $U = V$, then we can deduce the formula G which equals F except that U is replaced by V. The outline functions can be used to compute the conclusion formula c, given p_1, p_2, and π or to compute the formula p_1, given c, p_2, and π, or to compute the position π at which the replacement can be performed. Note that there are two outline functions for computing π.

Definition 4 (Admissible and Fully Specified Inference Substitutions). *Assume an inference* $\mathbf{I} := \langle P, C, \mathit{Hyp}, \mathit{FS}, \Omega, P(\boldsymbol{i^0_k}), \mathit{OF}, \mathit{Exp}\rangle$*, an inference substitution* σ *wrt. some task* T*. We say that* σ *is* admissible for $\mathbf{I}$ and T *iff (1) for all* $p \in P$ *and* $c \in C$ *such that* $\sigma(p), \sigma(c) \in \mathcal{POS}(T)$ $\sigma(p)$ *and* $\sigma(c)$ *are* α*-related positions in* T*, (2) for all* $c, c' \in C$ *such that* $\sigma(c), \sigma(c') \in \mathcal{POS}(T)$ $\sigma(c)$ *and* $\sigma(c')$ *are* β*-related positions in* T *(3) for all* $i \in (P \cup C) \cap \mathit{dom}(\sigma)$ *it holds (3.1) if* $i \in P$ *and* $\sigma(i) \in \mathcal{POS}(T)$ *then* $\sigma(i)$ *is a negative position in* T*; (3.2) if* $i \in C$ *and* $\sigma(i) \in \mathcal{POS}(T)$ *then* $\sigma(i)$ *must be a positive position in* T*; (3.3) if* $\sigma(i) \in \mathit{wff}^{\mathcal{X}}_{\Sigma,\mathcal{V}}$*, then* $\sigma(i) = \sigma(\mathit{FS}(i))$ *must hold.*

We say that σ *is* fully specified for $\mathbf{I}$ and T *iff it is admissible for* $\mathbf{I}$ *and* T *and* $P \cup C \subseteq \mathit{dom}(\sigma)$*. Otherwise we say* σ *is partial.*

4 Synthesizing Inferences from Mathematical Knowledge

In the old Ωmega-system the knowledge contained in the mathematical theories was not automatically available to the proof planner or the Ω-Ants-system. In order to make them available for these systems, they had to be specified manually as methods or tactics and agents. In this section we present a mechanism that allows the computation of a set of inferences for arbitrary formulas. The intuition is as follows: Given the following axiom which is a part of the definition of addition on natural numbers

$$\forall x, y.\ y > 0 \Rightarrow x + y = s(x + p(y)) \tag{1}$$

where s and p are the successor the predecessor functions from natural numbers respectively. We want it to result in the inference rules

$$\frac{R\star\mathcal{C};\Gamma\vdash_R A\Leftrightarrow B}{R\star\mathcal{C};\Gamma\vdash_R A\Rightarrow B\star\mathcal{C};\Gamma\vdash_R B\Rightarrow A}\Leftrightarrow_E$$

$$\frac{R\star\mathcal{C};\Gamma\vdash_R s=t}{R\star\mathcal{C};\Gamma\vdash_R X(s)\Rightarrow X(t)\star\mathcal{C};\Gamma\vdash_R X(t)\Rightarrow X(s)}=_E$$
where X new wrt. $\mathcal{C};\Gamma\vdash_R s=t$

$$\frac{R\star\mathcal{C};\Gamma\vdash_R A\Rightarrow B}{R\star\mathcal{C};\Gamma,[]A\vdash_R B}\Rightarrow_E$$

$$\frac{R\star\mathcal{C};\Gamma\vdash_R A\wedge B}{R\star\mathcal{C};\Gamma\vdash_R A\star\mathcal{C};\Gamma\vdash_R B}\wedge_E$$

$$\frac{R\star\mathcal{C};\Gamma\vdash_R \forall xA}{R\star\mathcal{C};\Gamma\vdash_R A[X/x]}\forall_E$$
where X new wrt. $\mathcal{C};\Gamma\vdash_R \forall xA$

$$\frac{R\star\mathcal{C};\Gamma\vdash_R \exists xA}{R\star\mathcal{C},c\notin\Gamma,\exists xA;\Gamma\vdash_R A[c/x]}\exists_E$$

$$\frac{R\star\mathcal{C};\Gamma,[\Delta]A\Rightarrow B\vdash_R F}{R\star\mathcal{C};\Gamma,[\Delta,A]B\vdash_R F}\Rightarrow_I$$

$$\frac{R\star\mathcal{C};\Gamma,[\Delta]A\wedge B\vdash_R F}{R\star\mathcal{C};\Gamma,[\Delta]A,[\Delta]B\vdash_R F}\wedge_I$$

$$\frac{R\star\mathcal{C};\Gamma,[\Delta]A\vee B\vdash_R F}{R\star\mathcal{C};\Gamma,[\Delta]A\vdash_R F\star\mathcal{C};\Gamma,[\Delta]B\vdash_R F}\vee_I$$

$$\frac{R\star\mathcal{C};\Gamma,[\Delta]\forall xA\vdash_R F}{R\star\mathcal{C},c\notin[\Delta]\forall xA;\Gamma,[\Delta]A[c/x]\vdash_R F}\forall_I$$

$$\frac{R\star\mathcal{C};\Gamma,[\Delta]\exists xA\vdash_R F}{R\star\mathcal{C};\Gamma,[\Delta]A[X/x]\vdash_R F}\exists_I$$
where X new wrt. $\mathcal{C};\Gamma,[\Delta]\exists xA\vdash_R F$

$$\frac{R\star\mathcal{C};\Gamma,[\Delta,A\wedge B]G\vdash_R F}{R\star\mathcal{C};\Gamma,[\Delta,A,B]G\vdash_R F}\wedge_E^H$$

Fig. 3. Sets of Rules to compute conclusions & premises

$$\frac{P_1:Y>0 \qquad P_2:H(X+Y)}{C:H(s(X+p(Y)))} \qquad\qquad \frac{P_1:Y>0 \qquad P_2:H(s(X+p(Y)))}{C:H(X+Y)} \tag{2}$$

Application Condition: – **Application Condition:** –

where H, X, Y are meta-variables. Similarly, the following direction of the equivalence of the definition of the limit of a function

$$\begin{aligned}&\forall f,a,l.\forall\epsilon.\epsilon>0\Rightarrow\exists\delta.\delta>0\Rightarrow\forall x.(0<\mid x-a\mid\wedge\mid x-a\mid<\delta)\Rightarrow\mid f(x)-l\mid<\epsilon\\&\qquad\Rightarrow\lim\nolimits_a f=l\end{aligned} \tag{3}$$

should give us the inference

$$\frac{\begin{matrix}[\epsilon>0,D>0,0<\mid x-A\mid,\mid x-A\mid<D]\\ \vdots\\ P:\mid F(x)-L\mid<\epsilon\end{matrix}}{C:\lim\limits_{A}F=L} \tag{4}$$

Application Condition: $\mathsf{EV}(\epsilon,\{F,A,L\})\wedge\mathsf{EV}(x,\{F,A,L,D\})$
Parameters: ϵ, x

where F, A, L, D are meta-variables, ϵ and x are parameters to the rule and $\mathsf{EV}(x,\{F,A,L,D\})$ is the application condition requiring that the parameter x should not occur in the instances of $\{F,A,L,D\}$.

Our mechanism is inspired by the generalized natural deduction procedure proposed by Wack [16]. It determines the application directions of formulas by following the introduction and elimination rule structure of a natural deduction

(ND) calculus [11]. Given a formula, in a first phase the ND elimination rules are exhaustively applied to that formula and where we eliminate equivalences in the obvious way and handle equality via Leibniz' definition of equality. While applying the rules, we collect the *Eigenvariable* conditions which results in a set of inference descriptions with *Eigenvariable* conditions. In a second phase the premises of the inference descriptions are simplified by exhaustively applying ND introduction rules to the premises as well as to possible hypotheses obtained for the premises in that phase. The two sets of rules are given in Fig. 3: We use $R \star x$ to denote $R \cup \{x\}$ and the notation $\mathcal{C}; \Gamma \vdash_R C$ for inference descriptions stating that the conclusion C can be derived from the hypotheses in Γ if the conditions in $\mathcal{C}$ are respected. The premises in Γ are of the form $[\mathcal{H}]P$ stating that in order to show the premise P we can assume the hypotheses $\mathcal{H}$. The *Eigenvariable* conditions collected in $\mathcal{C}$ are of the form *"y new wrt. S"*, where y is the Eigenvariable and S is a list of constants and meta-variables in which y shall not occur (including the symbols in the meta-variable substitutions). Checking these conditions by using the predicate $\mathsf{EV}(y, S)$ s, an inference description $\mathsf{EV}(y_1, S_1), \dots, \mathsf{EV}(y_m, S_m); [\mathcal{H}_1]P_1, \dots, [\mathcal{H}_n]P_n \vdash_R C$ gives rise to the inference

$$\dfrac{\begin{matrix}[\mathcal{H}_1] & & [\mathcal{H}_n] \\ \vdots & & \vdots \\ P_1 & \dots & P_n\end{matrix}}{C}\,\textbf{Parameters}\ (y_1, \dots, y_n)$$

$$\textbf{Application Condition: } \mathsf{EV}(y_1, S_1) \wedge \dots \wedge \mathsf{EV}(y_m, S_m)$$

The elimination rules for decomposition of the conclusions in the first phase are on the left-hand side and the introduction rules for the premises and the single elimination rule for the hypotheses are on the right-hand side.

It is obvious to see that both sets of rules are terminating and confluent (up to the ordering of the premises and their hypotheses, and the renaming of introduced Eigenvariables and meta-variables). We illustrate the procedure by considering the axioms (1) and (3): For (1) the initial inference description is $.;. \vdash_R \forall x, y.\ y \neq 0 \Rightarrow x + y = s(x + p(y))$ and saturation using the elimination rules yields the sets $\{.; []Y > 0, []H(X + Y) \vdash_R H(s(X + p(Y))) .; []Y > 0, []H(s(X + p(Y))) \vdash_R H(X + Y)\}$. None of the rules from the second set of rules is applicable and hence we obtain the two inferences from (2).

The initial inference description for (3) is $.;. \vdash_R \forall f, a, l.(\forall \epsilon.\epsilon > 0 \Rightarrow \exists \delta.\delta > 0 \Rightarrow \forall x.(0 <\mid x - a \mid \wedge \mid x - a \mid< \delta) \Rightarrow\mid f(x) - l \mid< \epsilon) \Rightarrow \lim_a f = l$ and the elimination rules yield the singleton $\{.; []\forall \epsilon.\epsilon > 0 \Rightarrow \exists \delta.\delta > 0 \Rightarrow \forall x.(0 <\mid x - A \mid \wedge \mid x - A \mid< \delta) \Rightarrow\mid F(x) - L \mid< \epsilon \vdash_R \lim_A F = L\}$. The second set of rules yields the singleton set

$$\left\{\begin{array}{l}\mathsf{EV}(\epsilon, \{F, A, L\}), \mathsf{EV}(x, \{F, A, L, D\}); \\ {[\epsilon > 0, D > 0, 0 <\mid x - A \mid, \mid x - A \mid< D] \mid F(x) - L \mid< \epsilon \vdash_R \lim_A F = L}\end{array}\right\}.$$

from which we obtain the inference (4).

5 Partial Argument Instantiations

The problem of determining the different possibilities to apply a given inference on some task consists of finding all fully specified inference substitution. Typically, some of the parameters of the inference are instantiated as well as possibly some of the formal arguments. Hence, the process starts with a partial inference substitution and we have to find the values for the non-instantiated variables of the inference. During the process these variables are assigned either an position of the task or a formula. In order to make the stages of the process explicit, we define the notion of *partial argument instantiation*, which is an adaptation of the notion from [4,15] to the new inferences presented here.

Definition 5 (Partial Argument Instantiation). *Let* **I** *be an inference with formal arguments* $a_1, \dots, a_n$ *and* T *a task. Any inference substitution* $PAI_I^T := \langle \sigma_{\mathcal{P}}, \sigma_{\mathcal{N}}, \sigma_{\mathcal{X}} \rangle$ *admissible for* I *and* T *is a* partial argument instantiation (PAI)*. A PAI* PAI_I^T *is called* empty*, iff* $dom(PAI_I^T) = \emptyset$*; it is called* initial *iff for all* $1 \leq j \leq n$ *holds* $\sigma_{\mathcal{N}}(a_j) \notin wff^{\mathcal{X}}_{\Sigma,\mathcal{V}}$ *and for all* $p \in \mathcal{P}$ *holds* $\sigma_{\mathcal{P}}(p) = \bot$*; finally we say it is* complete *iff there is sequence of outline functions* $\langle V_l, f(\boldsymbol{I^k_{k_l}})$, $0 \leq k \leq m$ *such that* $PAI_I^T \oplus \langle v_1, f(\boldsymbol{i^1_{k_1}})\rangle \oplus \dots \oplus \langle v_m, f(\boldsymbol{i^m_{k_m}})\rangle$ *is a fully specified inference substitution wrt.* **I** *and* T*.*

Example 1. Let us reconsider the inference *subst-m* from Fig. 2, which we want to apply to the task T: $2 * 3 = 6 \vdash 2 * 3 < 7$. Then the substitution $pai_1 = \langle \emptyset, \{c \mapsto (10)\}, \emptyset \rangle$ is a PAI for the inference *subst-m*, where (10) denotes the position of $2 * 3 < 7$ in the task. As no outline function has been invoked so far pai_1 is initial. It is not complete as there are no outline functions to compute π given only c; thus p_1, p_2 can also not be computed.

The extension of a partial argument instantiation PAI_I^T consists of an assignment of values to variables of the inference that are not in the domain of PAI_I^T. We are only interested in those extensions where at least one not yet instantiated *formal argument* is assigned a new value.

Definition 6 (Partial Argument Instantiation Update). *Let* **I** *be an inference,* T *be a task,* $PAI_{\mathbf{I}}^T, {PAI'}_{\mathbf{I}}^T$ *be partial argument instantiations for* **I** *with respect to* T*. Then* ${PAI'}_{\mathbf{I}}^T$ is a partial argument update of $PAI_{\mathbf{I}}^T$ *iff (1)* $PAI_{\mathbf{I}}^T$ *is more general than* ${PAI'}_{\mathbf{I}}^T$ *and (2) there is at least one* formal argument *of* **I** *in* $dom({PAI'}_{\mathbf{I}}^T) \setminus dom(PAI_{\mathbf{I}}^T)$*.*

There are two possibilities to perform a partial argument instantiation update: (1) assigning a task position to a formal argument or (2) assigning a term to a formal argument. The first kind of update involves searching for possible positions in the task while respecting already introduced bindings. The second kind of updates involves no search in practice and is performed by using the outline functions which require that some arguments are already instantiated.

Thus we can divide the updating process into two phases: In the first phase we allow only updates of the first kind and in the second phase only updates of the second kind. The underlying idea is that we try to use as much derived knowledge as possible and then decide whether this knowledge suffices for the inference to be applied. The knowledge in an initial PAI suffices for the outline functions to compute all other values, if the initial PAI is complete. Each initial and complete PAI determines one way the inference can be applied.

Example 2. If we add an instantiation for the argument p_2 in pai_1 we obtain a new PAI $pai_2 = \langle\emptyset, \{p_2 \mapsto (00), c \mapsto (10)\}, \emptyset\rangle$, where (00) denotes the position of the formula $2 * 3 = 6$ in our task T. p_2 is a PAI-update of p_1. It is complete, as we can invoke the outline functions to first obtain π and then to obtain p_1. Note that this does not mean that we have to invoke the outline functions, but we can try to instantiate further arguments by formulas of our task.

Abstracting from the concrete values the formal parameters are instantiated by these PAIs, we obtain a general descriptions of the application directions of the inferences.

The determination of the application direction in turn can be used (1) for planning as we know what the possible effects of an inference application are, and (2) for generating agents as it is reasonable that the agents must be able to construct partial argument instantiations for all application directions.

In the remainder of this section we formalize the notion of application direction by defining *PAI-statuses* that represent the status of a PAI rather than the concrete values.

Definition 7 (*PAI*-Status, *PAI*-Status defined by a *PAI*). *Let* $\mathbf{I}$ *be an inference with formal arguments* $\mathbf{A}$ *and* T *a task. A* PAI-status $\mathcal{S}_\mathbf{I}$ *of* $\mathbf{I}$ *is a function* $\mathcal{S}_\mathbf{I} : \mathbf{A} \to \{TERM, POS, \perp\}$*. Its domain is* $\mathrm{dom}(\mathcal{S}_\mathbf{I}) := \{a \mid \mathcal{S}_\mathbf{I}(a) \neq \perp\}$*. A PAI* $PAI^T_\mathbf{I}$ *of* $\mathbf{I}$ *belongs to some PAI-status* $\mathcal{S}_\mathbf{I}$ *iff for all* $a \in \mathbf{A}$ *it holds (*$\perp$*) if* $PAI^T_\mathbf{I}(a) = \perp$ *then* $\mathcal{S}_\mathbf{I}(a) = \perp$*, (POS) if* $PAI^T_\mathbf{I}(a) \in \mathcal{POS}$ *then* $\mathcal{S}_\mathbf{I}(a) = POS$*, or (TERM) if* $PAI^T_\mathbf{I} \in wff^{\mathcal{X}}_{\Sigma,\mathcal{V}}$ *then* $\mathcal{S}_\mathbf{I}(a) = TERM$*.*

We say that two PAIs are *equivalent*, written $PAI^{(T)}_\mathbf{I} \sim PAI'^{(T')}_\mathbf{I}$, if they belong to the same PAI status.

Example 3. As an example consider the PAI pai_2. The status defined by this PAI is a function $f : \{p_1, p_2, c\} \to \{TERM, POS, \perp\}$ with $f(p_1) = \perp$, $f(p_2) = POS$, and $f(c) = POS$. Suppose we are given another task $T_2 : 2 + 2 = 4 \vdash A \subset B, 2+2 = 1+1+1+1$ with a PAI for *subst-m* $pai_3 = \langle\emptyset, \{p_1 \mapsto (00), c \mapsto (11)\}, \emptyset\rangle$. Then pai_3 and pai_2 have the same status.

The notions of updating a PAI, as well as initial and complete PAIs carry over to PAI-statuses in a straightforward manner:

Definition 8 (*PAI*-Status Update). *Let* $\mathcal{S}, \mathcal{S}'$ *be PAI-statuses of an inference* $\mathbf{I}$*.* $\mathcal{S}'$ *is called* PAI-status-update *iff (1)* $\forall x \in \mathrm{dom}(\mathcal{S})$. $\mathcal{S}'(x) = \mathcal{S}(x)$ *and (2) there occurs at least one* formal argument *of* $\mathbf{I}$ *in* $\mathrm{dom}(\mathcal{S}') \setminus \mathrm{dom}(\mathcal{S}')$*.*

Note that the PAI-status update relationship is a partial order on PAI-statuses, which we denote by $<_\mathcal{S}$*.*

Consequently, the status of pai_2 is a PAI-status update of the status of pai_1.

Definition 9 (Initial, Empty, Complete, and Full PAI-Statuses). *Let $\mathcal{S}$ be a PAI-status of an inference* $\mathbf{I}$ *with formal arguments* $\mathbf{A}$*. Let further $\mathcal{S}^{-1}$ denote the inverse image of $\mathcal{S}$ defined by $\mathcal{S}^{-1}(x) := \{a \in \mathbf{A} \mid \mathcal{S}(a) = x\}$. We say that $\mathcal{S}$ is (1)* initial *iff $\mathcal{S}^{-1}(TERM) = \emptyset$, (2) it is* empty *iff $\mathcal{S}^{-1}(POS) = \mathcal{S}^{-1}(TERM) = \emptyset$, (3) it is* complete *iff there exists a* complete *PAI $PAI_{\mathbf{I}}^{(T)} \in \mathcal{S}_{\mathbf{I}}$, and (4) it is* full *iff there exists a* full *PAI $PAI_{\mathbf{I}}^{(T)} \in \mathcal{S}_{\mathbf{I}}$.*

Notational Convention. Given an inference $\mathbf{I}$, the empty PAI-status of $\mathbf{I}$ is denoted by $\mathcal{S}^{\emptyset}$. We agree to denote an initial PAI-status $\mathcal{S}$ where $\mathcal{S}^{-1}(POS) = \{a_1, \ldots, a_n\}$ by $\langle a_1, \ldots, a_n \rangle$.

Definition 10 (Application Directions). *Let* $\mathbf{I}$ *be an inference. The initial and complete PAI-statuses of* $\mathbf{I}$ *are the* application directions $\mathcal{AD}_{\mathbf{I}}$ *of* $\mathbf{I}$.

Example 4. As an example consider again the inference *subst-m*: it has the 3 application directions $\langle p_1, p_2, c \rangle$, $\langle p_1, p_2 \rangle$, and $\langle p_2, c \rangle$.

To check whether or not an initial $\mathcal{S}$ is an application direction, i.e. is complete, we construct a so-called *PAI-status completion tree* for the $\mathcal{S}$ we want to test. Each node of the tree is labeled with a PAI-status and we add an edge from a PAI-status $\mathcal{S}'$ to some $\mathcal{S}''$, if there is an outline function $of := \langle v, f(i_1, \ldots, i_n) \rangle$ which is applicable on $\mathcal{S}$, i.e. $\{i_1, \ldots, i_n\} \subseteq dom(\mathcal{S})$ and $\mathcal{S}(v) = \bot$. We recursively construct that tree starting with the given $\mathcal{S}$, and finally check if there is at least one *full PAI-status* among the leaf-nodes of the tree. If so, then $\mathcal{S}$ is complete, i.e. is an application direction, and otherwise not. If $\mathcal{S}^{-1}(a) = TERM$, a is annotated with $\oplus$, otherwise if $a \in C$ with $\ominus$, if $a \in P$ it remains unannotated.

6 Generating Agents

Given a specification of an inference, we want to create a set of argument agents for the Ω-Ants-system which provide the user with suggestions of how the formal arguments of the inference can be instantiated. As the agents must cooperate to find instantiations, the benefit of a single agent cannot be assessed, rather we have to see the agents as a unit. These units are "fragile" in the sense that removing one agent can result in a non-operational unit that cannot produce useful suggestions or any suggestions at all. Hence we must choose a sufficiently large set of agents such that for each agent there is another agent which produces partial argument instantiations required by the agent. On the other hand each agent consumes runtime. If we created all possible agents the system performance would deteriorate, as it would take too much time to create a suggestion. Hence we have to construct a set of agents which forms an operational unit and is as small as possible.

Intuitively a unit of agents for an inference reads partial argument instantiations from a blackboard. If a single agent realizes that it can perform an update, it performs that update and writes the new partial argument instantiation onto the blackboard, such that it can be further updated by the other agents of the unit. We aim at a unit of agents that is able to find all application directions of an inference, provided instances of these application directions can be built with respect to the task to which the agents are applied.

Agent Creation Graph. The agent creation graph is composed of nodes labeled with initial PAI-statuses and edges between these nodes, representing PAI-status updates. We only encode updates in the graph which update exactly one formal argument.

Definition 11 (Agent Creation Graph). *Let* $\mathbf{I}$ *be an inference and* $N = \{\mathcal{S}^1_{\mathbf{I}}, \ldots, \mathcal{S}^n_{\mathbf{I}}\}$ *be the set of all initial* PAI*-statuses of* $\mathbf{I}$*. Let* $(\mathcal{S}^i_{\mathbf{I}}, \mathcal{S}^j_{\mathbf{I}}) \in E$ *iff* $\mathcal{S}^j_{\mathbf{I}}$ *is a PAI status update of* $\mathcal{S}^i_{\mathbf{I}}$ *and* $|dom(\mathcal{S}^i_{\mathbf{I}})| + 1 = |dom(\mathcal{S}^j_{\mathbf{I}})|$*. Then* $G = \langle N, E\rangle$ *is called* agent creation graph *where each edge is labeled with* $\oplus A$*, where* A *is the formal argument additionally instantiated in* $\mathcal{S}^j_{\mathbf{I}}$*.*

An argument agent $\mathfrak{A}$ consists of a dependency set $\mathcal{D}_{\mathfrak{A}}$ of formal arguments and a goal set $\mathcal{G}_{\mathfrak{A}}$ of formal arguments. Hence each edge in an agent creation graph corresponds to exactly one argument agent whose dependency set contains those formal arguments instantiated in the source PAI-status and whose goal set contain that formal argument which is additionally instantiated in the target PAI-status. Hence we can associate to a given set of argument agents $\mathfrak{K}$ that subgraph of the agent creation graph covered by these argument agents. This graph is called *actual agent graph.*

Definition 12 (Actual Agent Graph). *Let* $G = \langle N, E\rangle$ *be an agent creation graph and* $\mathfrak{K}$ *a set of agents. The smallest subgraph of* G *that contains for each* $\mathfrak{A} \in \mathfrak{K}$ *an edge* $(\mathcal{S}, \mathcal{S}')$ *such that* $dom(\mathcal{S}) = \mathcal{D}_{\mathfrak{A}}$ *and* $dom(\mathcal{S}') \setminus dom(\mathcal{S}) = \mathcal{G}_{\mathfrak{A}}$ *is the* actual agent graph of $\mathfrak{K}$*.*

In an agent creation graph, those PAI-statuses which are complete are those which must be reachable from the root node of the graph, since they are the states which can be transformed into a fully specified PAI-status, in which the associated inference is applicable. In order to have a minimal set of agents $\mathfrak{K}$ we require that in the actual agent graph of $\mathfrak{K}$ there is *exactly one* path from the root node to each complete PAI-status. From this we can easily derive the following algorithm to create agent units for a given inference.

Algorithm to Create an Agent Unit. Given an inference $\mathbf{I}$ and the complete initial PAI-statuses partially ordered wrt. $<_{\mathcal{S}}$ (Def. 8). The algorithm creates a unit of argument agents for $\mathbf{I}$ such that the given complete PAI-statuses are all reachable from the empty PAI-status.

Initially, the set of agents is empty and the actual agent graph contains only the empty PAI-status. The algorithm recurses over the given list of complete PAI-statuses and in each iteration selects the first element $\mathcal{S}$ of the list of PAI-statuses. Since they are ordered by $<_{\mathcal{S}}$, $\mathcal{S}$ is not yet reachable in the actual

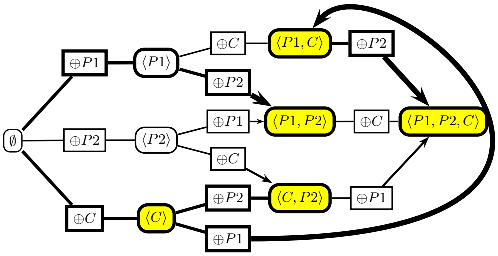


Fig. 4. Agent Creation Graph

agent graph. It then calculates the shortest paths[2] in the agent creation graph to $\mathcal{S}$ from all PAI-statuses that are already in the actual agent graph. It selects a path that has minimal distance to $\mathcal{S}$ and adds agents for each edge on that path to the set of agents. As the list of statuses is ordered and the minimal path is always added the algorithm produces an optimal set of agents.

Figure 4 shows an agent creation graph for an inference **I** which has two premises $P1$ and $P2$ and one conclusion C. Suppose that the complete PAI-statuses of **I** are $\langle C\rangle$, $\langle C, P2\rangle$, $\langle P1, P2\rangle$, $\langle P1, C\rangle$, and $\langle P1, P2, C\rangle$. The complete PAI-statuses are indicated in gray and the actual agent graph constructed by our algorithm is indicated by the bold edges and nodes. It illustrates that the agents optimally cooperate to find partial argument instantiations. A comparison in [7] of some automatically generated units of argument agents with manually specified argument agents shows that they are almost identical.

7 Related Work

Closest to the presented style of operationalizing mathematical knowledge are the *macetes* in the IMPS-system [9] and the *replacement rules* of the CoRe-calculus [1]. Macetes only work with pure universal formulas in prenex normal-form and require the formulas to be given in the right format; the conditions are not further decomposed and equations (and equivalences) are only applied from left to right. Our mechanism in turn is less flexible than the synthesis of replacement rules: our rules require that negations occur only on atoms inside the formulas in order to have only literals as premises while the CoRe mechanisms would not require this restricted format. Adding this feature to our rules would pose no difficulty, but we refrained from doing so, because in the cases exploiting that feature it would be difficult for a human user the recognize the original formula from the generated inferences. Consider as an example the formula $\neg(A \wedge (B \Rightarrow C))$ and the then generated inferences $.; []A \vdash_R B$ and $.; []A \vdash_R \neg C$. Furthermore, the result of the synthesis must meet the format of inferences and, hence, except for conjunctions, we cannot allow for rules that further decompose the hypotheses of premises.

[2] Using, for instance, Dijkstra's algorithm [8].

8 Conclusion

We have presented an approach to extract proof procedural information from declarative representations of mathematical knowledge. It is based on a new notion of an inference and techniques to determine all sensible application directions and argument agents of inferences. We adapted a mechanism from [16] to automatically synthesize inferences from assertions, which enables the proof-planner Multi and the agent-based reasoning system Ω-Ants to actively use knowledge contained in mathematical theories and to effectively assist the user even if he does not know the peculiarities of how to condition knowledge for these systems.

References

1. S. Autexier. The CoRe calculus. In R. Nieuwenhuis, editor, *Proceedings of CADE-20*, LNAI 3632, Tallinn, Estonia, july 2005. Springer.
2. S. Autexier, Chr. Benzmüller, D. Dietrich, A. Meier, and C.-P. Wirth. A generic modular data structure for proof attempts alternating on ideas and granularity. In M. Kohlhase, editor, *Proceedings of MKM'05*, LNAI 3863, IUB Bremen, Germany, january 2006. Springer.
3. H. Barendregt. *The λ-Calculus - Its Syntax and Semantics.* North Holland, 1984.
4. Chr. Benzmüller and V. Sorge. Ω-Ants – an open approach at combining interactive and automated theorem proving. In M. Kerber and M. Kohlhase, editors, *Proceedings of Calculemus-2000*, St. Andrews, UK, 6–7 August 2001. AK Peters.
5. Christoph Benzmüller, Armin Fiedler, Andreas Meier, Martin Pollet, and Jörg Siekmann. Omega. In *The seventeen provers of the world*, number 3600 in LNAI, pages 127–141, 2006.
6. Alan Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction (CADE-88)*, LNAI, pages 111–120. Springer, 1988.
7. D. Dietrich. The task-layer of the *Ω*mega system. Diploma thesis, FR 6.2 Informatik, Universität des Saarlandes, Saarbrücken, Germany, 2006.
8. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
9. W. Farmer, J. Guttman, and F. J. Thayer. IMPS: An interactive mathematical proof system. *Journal of Automated Reasoning*, 11, 1993.
10. M. Fitting. *First-Order Logic and Automated Theorem Proving / 2nd Edition.* Springer-Verlag New York Inc., 1996. ISBN 0-387-94593-8.
11. G. Gentzen. *The Collected Papers of Gerhard Gentzen (1934-1938).* Edited by Szabo, M. E., North Holland, Amsterdam, 1969.
12. M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF – A mechanised logic of computation.* Springer Verlag, 1979. LNCS 78.
13. A. Meier and E. Melis. Multi: A multi-strategy proof-planner. In R. Nieuwenhuis, editor, *Proceedings of CADE-20*, LNAI 3632, Tallinn, Estonia, july 2005. Springer.
14. J. D. C. Richardson, A. Smaill, and I. M. Green. System description: proof planning in higher-order logic with λ-clam. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction (CADE-98)*, volume 1421 of *LNAI*. Springer, 1998.

15. V. Sorge. *A Blackboard Architecture for the Integration of Reasoning Techniques into Proof Planning*. PhD thesis, FR 6.2 Informatik, Universität des Saarlandes, Saarbrücken, Germany, Nvermber 2001.
16. B. Wack. *Typage et déduction dans le calcul de réécriture*. Thèse de doctorat, Université Henri Poincaré (Nancy 1), octobre 2005.
17. F. Wiedijk. Formal proof sketches. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Types for Proofs and Programs: Third International Workshop, TYPES 2003*, LNCS 3085, pages 378–393, Torino, Italy, 2004. Springer.

Verifying and Invalidating Textbook Proofs Using Scunak

Chad E. Brown

Universität des Saarlandes, Saarbrücken, Germany
cebrown@ags.uni-sb.de

Abstract. Many textbook proofs are essentially human-readable representations of natural deduction proofs. Terms in dependent type theory provide formally checkable representations of natural deduction proofs. We show how the new mathematical assistant system Scunak can be used to verify a textbook proof by translating the LaTeX version into a proof term in a dependent type theory. We also show how Scunak can give interesting output upon failure.

1 Introduction

We use the new mathematical assistant system Scunak to analyze different versions of a textbook proof. We explain how Scunak can transform a LaTeX version of the proof into a formally checkable proof term. Furthermore, in some cases, Scunak can signal an error and reject the proof. We use this case study to illustrate and discuss various issues related to the formalization of mathematics. For instance, in a textbook proof, how do we recognize if an assumption is "correct"? Also, how do we determine when an eigenvariable or hypothesis has been discharged? As we shall see, Scunak offers potential answers to such questions.

2 What is Scunak?

Scunak is like Automath [16] or Twelf [13] in the sense that the user creates a dependently typed signature of constants (corresponding to basic concepts and axioms) and definitions (corresponding to defined concepts and theorems with proofs). Scunak is like Coq [3] in that one can interactively create proof objects using commands corresponding to natural deduction in addition to applying previously proven facts. The type theory of Scunak is different from (and in most respects more restricted than) the type theories of Twelf, Automath and especially Coq. In particular, Scunak excludes (all forms of) polymorphism, and restricts to a fixed number of basic type families and second-order types. Also, though one is free to declare constants and definitions for whatever mathematical foundations one wants, Scunak does include built in support for signatures which include certain set theoretic concepts. So, Scunak is unlike Automath or Twelf in that set theory signatures have direct support in the system.

Scunak is also similar to MIZAR [15] in many respects. Scunak currently uses a form of untyped set theory (Mac Lane set theory with Universes). The type

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 110–123, 2006.

theory allows the user to treat any class (relative to the set theory) like a type. MIZAR supports a different form of set theory (Tarski-Groethendieck), but this is not relevant here. MIZAR has a type structure, but prohibits empty types. This difference is best illustrated by the "class" $\{x|x \in A\}$ where A is a set. In MIZAR, the type `Element of` A represents this class, unless A is empty, in which case `Element of` A consists of the empty set [4]. In Scunak, the corresponding class type `class (in A)` is empty if `A` is empty. (We usually simply write `A` for `class (in A)` and let the parser determine it is the class type of `A`.) There are pros and cons of allowing empty types which we do not discuss here. Another important distinction between Scunak and MIZAR is that Scunak currently has a library of about 300 theorems whereas MIZAR has a library of about $40,000$ theorems [4]. It is important to point out that neither Scunak nor MIZAR is irrevocably tied to the particular set theory in which the mathematics is represented (as discussed in [15]).

Four goals were the most important in the design of Scunak:

1. **Naturality:** The mathematics should be represented in a natural way, similar (up to isomorphism) to what appears in mathematics texts.
2. **Formal Correctness:** The proofs should be formally checkable.
3. **Retrievability:** Retrieving theorems by content rather than by name must be possible.
4. **Automation:** Some reasonable degree of automated reasoning for proving theorems in the logic should be available.

We will illustrate the first three points in this paper by considering the formalization of a simple textbook proof from [2] (also considered in [1]). We will explain how Scunak can read the LaTeX representation of the proof, verify the proof, and construct a corresponding checkable proof term. During this process, Scunak must sometimes use facts in the signature in order to justify steps in the proof. We will also demonstrate how Scunak can read mutilated LaTeX versions of the proof and identify the error. We do not claim, however, Scunak is an "industrial strength" proofreader for mathematical proofs in LaTeX. The example is a vehicle for discussing various issues related to formalizing mathematics and for introducing Scunak as a tool for working with formalized mathematics.

3 A Simple Textbook Proof

We consider the first proof given in the introductory analysis book [2]. The proof (shown in Figure 1) is of the distributive law $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ (the first equation in part (d) of Theorem 1.1.4 in [2]). This proof was also considered in [1] (which was the source for the LaTeX version of the proof). The only difference between the LaTeX source of the proof in [1] and the LaTeX source discussed here is that the version in [1] used macros `\union` and `\inter` where the version checked by Scunak uses the standard macros `\cup` and `\cap`.

We begin by considering the structure of this proof. The first sentence states the intention to prove the distributive law. Scunak finds no pattern in this sentence with mathematical or proof content and so ignores the sentence. (Any text

In order to give a sample proof, we shall prove the first equation in (d). Let x be an element of $A \cap (B \cup C)$, then $x \in A$ and $x \in B \cup C$. This means that $x \in A$, and either $x \in B$ or $x \in C$. Hence we either have (i) $x \in A$ and $x \in B$, or we have (ii) $x \in A$ and $x \in C$. Therefore, either $x \in A \cap B$ or $x \in A \cap C$, so $x \in (A \cap B) \cup (A \cap C)$. This shows that $A \cap (B \cup C)$ is a subset of $(A \cap B) \cup (A \cap C)$.

Conversely, let y be an element of $(A \cap B) \cup (A \cap C)$. Then, either (iii) $y \in A \cap B$, or (iv) $y \in A \cap C$. It follows that $y \in A$, and either $y \in B$ or $y \in C$. Therefore, $y \in A$ and $y \in B \cup C$ so that $y \in A \cap (B \cup C)$. Hence $(A \cap B) \cup (A \cap C)$ is a subset of $A \cap (B \cup C)$.

In view of Definition 1.1.1, we conclude that the sets $A \cap (B \cup C)$ and $(A \cap B) \cup (A \cap C)$ are equal.

Fig. 1. Textbook Proof

which matches no rule in the proofreader's context-free grammar is ignored.) The second sentence introduces $x \in A \cap (B \cup C)$. Why? A reader of the proof (say, R) is aware of the goal of proving the sets $A \cap (B \cup C)$ and $(A \cap B) \cup (A \cap C)$ are equal. R should also know a common technique for proving such sets equal is to prove the two subset inclusions (relying on set extensionality, which is the content Definition 1.1.1 in [2]). Given this information, R can conclude that the author of the proof is introducing the eigenvariable x and hypothesis that $x \in A \cap (B \cup C)$ holds in order to prove $x \in (A \cap B) \cup (A \cap C)$ and thus conclude $A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$. Thus R expects to read a proof of the goal $x \in (A \cap B) \cup (A \cap C)$ and then read a proof of the other inclusion $(A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C)$. The mathematical content of the remainder of paragraph one, until the last sentence, outlines a series of facts which should follow from the previous facts and assumptions. The last sentence concludes $A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$. Note that sentence does not explicitly say, "we discharge x and the hypothesis that $x \in A \cap (B \cup C)$ in order to conclude...," though this is clearly the logical structure.

The second paragraph introduces a new eigenvariable y and assumption $y \in (A \cap B) \cup (A \cap C)$. R should be expecting a proof of $(A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C)$ and so such an assumption is warranted. Upon reading the assumption, R expects a proof that $y \in A \cap (B \cup C)$, which is exactly the content of the next three sentences. The last sentence implicitly discharges the y and corresponding assumption. The last sentence of the proof simply acknowledges that the proof is finished. We ignore the reference to "Definition 1.1.1." Instead of ignoring this reference, a superior system would recognize "Definition 1.1.1" corresponds to particular facts or rules (such as the rule named `setextsub` discussed later) and ensure that such a rule is used in the justification.

The explanation above gives an intuitive idea of the behavior of Scunak upon reading the proof in Figure 1. Before giving a more detailed explanation, along with examples of proofs with errors, we consider the Scunak type theory and set theory in which the content will be formally represented.

4 The Scunak Type Theory and Set Theory

The “logic” of Scunak is a modified version of the type theory LF as implemented in Twelf [13]. The meta-theory of LF is developed in [5]. The type theory of Scunak incorporates proof irrelevance for proof types. A more general framework for proof irrelevance is developed in [12] and [14]. A more thorough investigation of the Scunak type theory is planned for future work. Here we only sketch the type theory.

In the Scunak type theory, we restrict to three basic types:

- `obj` is the type of all (untyped) mathematical objects. Since we will be mainly interested in set theory, `obj` will be a synonym for `set`.
- `prop` is the type of mathematical propositions.
- `pf` P is a type whenever P is of type `prop` (in a context).

Such a restriction is not in the LF-tradition. The Scunak type theory is intended to model mathematical foundations instead of general logical and computational systems, so the restriction seems warranted.

One reason to insist on a single type of all mathematical objects is to avoid problems with polymorphism. However, sometimes one wants to consider a “typed” object. For this reason, we allow “class types.” `class` ϕ is a type whenever ϕ is a predicate (i.e., a function from `obj` to `prop`). Members of `class` ϕ are pairs $\langle x, p\rangle$ where x has type `obj` and p has type `pf` $(\phi p)^{\downarrow}$, where $(\phi p)^{\downarrow}$ is a normal form of (ϕp). Intuitively, `class` ϕ consists of objects x along with a proof that x satisfies ϕ. Since we do not want different proofs of the property to correspond to different elements of `class` ϕ, we include proof irrelevance in the type theory. In particular, if M and N are of type `pf` P, then M and N are the *same* in `pf` P.

All other types are function types constructed using the the Π dependent type constructor.

We now describe the syntax in more detail. We use $x, y, z, x^1, X, Y, \ldots$ to denote variables and $c, d, c^1, \ldots$ to denote constants. We define terms and types as follows:

$$\begin{array}{ll}\textbf{Terms } M, N, P, Q, R, \phi, \ldots := & x|c|(\lambda x.M)|(M\ N)|\langle M, N\rangle|\pi_1(M)|\pi_2(M)\\ \textbf{Types } A, B, C, \ldots := & \texttt{obj}|\texttt{prop}|(\texttt{pf}\ \ P)|(\texttt{class}\ \ \phi)|(\Pi x : A.B)\end{array}$$

As usual we use $(A \to B)$ to denote $(\Pi x : A.B)$ when x does not occur free in B. Also, we identify terms and types up to α-conversion. We assume all the usual notions of λ-calculus: substitution, β-reduction, η-reduction and the following pairing reductions:

$$\begin{array}{ll}(\pi_1) : \pi_1(\langle M, N\rangle) \to_{\pi_1} M & (\pi_2) : \pi_2(\langle M, N\rangle) \to_{\pi_2} N\\ (\pi) : \langle \pi_1(M), \pi_2(M)\rangle \to_{\pi} M & \end{array}$$

We say a term or type is normal if it contains no redexes. We write $W^{\downarrow}$ for the normal form of W, if a unique normal form of W exists. In practice we will only consider terms which are typable using simple types (in the Curry style [6]).

This guarantees strong normalization and the Church-Rosser property whenever necessary, so that we can assume $W^{\downarrow}$ exists uniquely.

In Scunak, terms and types are always given in $\beta\pi_1\pi_2$-normal form, so that the types of λ-abstractions and pairs can be inferred from the given intended type. We do not consider η-long (or π-long) forms. Instead, the rules for the typing judgments η- or π-expand on demand.

A signature Σ is a list of distinct constants associated with types. A type context Γ is a list of distinct variables associated with types. When Γ is $z^1 : A^1, \ldots, z^n : A^n$, we may write $\lambda\Gamma M$ for $\lambda z^1 \cdots \lambda z^n M$ and $(M\Gamma)$ for $(Mz^1 \cdots z^n)$.

We list the main typing judgments below, but omit the rules for space reasons.

- "$\vdash \Sigma$ sig" Intuitively, Σ is a valid signature. The idea is to ensure $\vdash_{\Sigma} A : \mathit{Type}$ before adding $c : A$ to Σ.
- "$\vdash_{\Sigma} \Gamma$ ctx" Intuitively, Γ is a valid context. The idea is to ensure $\Gamma \vdash_{\Sigma} A : \mathit{Type}$ holds before adding $x : A$ to Γ.
- "$\Gamma \vdash_{\Sigma} M \sim N \uparrow A$" Intuitively, M can be checked to be A-related to N.
- "$\Gamma \vdash_{\Sigma} M \sim N \downarrow A$" Intuitively, the type A can be extracted as a type in which M is A-related to N.
- "$\Gamma \vdash_{\Sigma} A : \mathit{Type}$" In words, A is a valid type.

We say M has type A in context Γ if $\Gamma \vdash M \sim M \uparrow A$ or $\Gamma \vdash M \sim M \downarrow A$ holds. (The difference between extracting and checking types is a form of "bi-directional" algorithmic typing.) We usually omit the dependence on Σ in the judgments.

The Scunak type theory is further restricted to second-order. We consider `obj`, `prop`, `pf` P and `class` ϕ to be first-order types. For all $x : B \in \Gamma$, we insist B is a second-order type, i.e., of the form $\Pi x^1 : C^1 \cdots \Pi x^n : C^n.C^{n+1}$ where each C^i is a first-order type. For all $c : A \in \Sigma$, A must be a third-order type of the form $\Pi x^1 : B^1 \cdots \Pi x^n : B^n.C$ where each B^i is a second-order type and C is a first-order type.

The set theory is implemented in Scunak in the same spirit as the formal systems in [17]. In fact, the fact that many signatures in [17] are second-order was a motivation for considering that a second-order logical framework might be sufficient for representing mathematics.

The particular set theory is a form of Mac Lane set theory with universes which we will denote by **MU**. The kernel for this set theory is contained in a signature $\Sigma_{\mathbf{MU}}$. This signature currently contains 482 constants and abbreviations which includes the basic constructions and theorems leading to sets of functions. Of these 482 signature elements, 29 are constants entered by the user, 339 are abbreviations entered by the user, and 114 are constants for folding and unfolding definitions. These 114 constants are automatically generated by Scunak when the abbreviations are given. If one assumes δ-reduction during type checking, the 114 generated constants are definable. (Due to proof irrelevance, one never needs to fold or unfold definitions returning a proof type. For this reason, constants for folding and unfolding such definitions are not generated.) We only present enough details of the signature for the formalization of the textbook proof to make sense.

Two constants in $\Sigma_{\mathbf{MU}}$ are `eq` and `in`, both of type `obj` → `obj` → `prop`. As concrete syntax, Scunak allows infix notation `(A==B)` for `(eq A B)` and `(x::A)` for `(in A x)`. Note that `(in A x)` means $x \in A$. The reason for this choice is so that the η-short form `(in A)` corresponds to the class determined by the set `A`. In concrete syntax, we can use a set `A` as a type, though this is technically the class type `class (in A)`. Likewise, we can indicate this class type as `(in A)`. As noted earlier, if A is empty, this class type is empty.

There are abbreviations in $\Sigma_{\mathbf{MU}}$ corresponding to conjunction, disjunction, subset, binary union, and binary intersection. We use infix notations `&`, `|`, `<=`, `\cup`, and `\cap` as concrete syntax for these notions. When convenient, we may also display `<=`, `\cup` and `\cap` as $\subseteq$, $\cup$ and $\cap$, respectively.

There are also abbreviations corresponding to facts about these concepts. Consider the following two abbreviations (we only give the types):

- `setextsub` : Π`A` : `obj`.Π`B` : `obj`.Πu : `pf (A<=B)`.Πv : `pf (B<=A). pf (A==B)`. This constant can be used to form a proof that A equals B given two sets A and B, a proof of $A \subseteq B$ and a proof of $B \subseteq A$.
- `subsetI1` : Π`A` : `obj`.Π`B` : `obj`.Πu : (Πx : `class (in A)`.`pf (in B` $\pi_1(x)$`))`. `pf (A<=B)`. We can conclude $A \subseteq B$ given two sets A and B and a function taking any x in the class type determined by A to a proof that the untyped part of x is an element of B.

In the concrete syntax when x is in a class type such as `class (in A)`, we can write `(`x`::B)` for `(in B` $\pi_1(x)$`)` since this can only be well-typed if we apply π_1 to take x from being a member of a class type to being of type `obj`. Likewise, we need never explicitly write $\pi_2(x)$ since the type constraints determine when x is being used as a proof type. Essentially, the π_1 and π_2 operators are reconstructed during parsing and the reconstructed term is type checked. Thus in any particular occurrence of a term M of class type, M may be used as a member of this class type, as an untyped object, or as a proof that the untyped object is in the class.

5 Verifying the Simple Textbook Proof

Following [8,7], one can formalize a textbook proof via a three stage "formalization path." First, translate from text to weak type theory (WTT). Second, translate this to a type theory with open terms (OTT). Third, instiate the metavariables in open terms.

We follow an approach similar to [8,7], though we do not work in stages. Instead, we extract certain commands from the text (LaTeX) and these commands are executed by Scunak to affect the current proof state. At any point, the proof state corresponds to an open term with a stack of remaining tasks. In the end, the proof is completed if there are no remaining tasks (hence no remaining free variables). All this is performed when a user invokes the Scunak command proofread with a filename containing the LaTeX source and a Scunak term corresponding to the formal version of the theorem which is to be proved.

Figure 2 shows the 14 Scunak commands extracted from the text in Figure 1 by the Scunak proofreader. These commands are in principle hidden from the user and are immediately executed upon being extracted from the text. The method of extracting these commands from the LaTeX source is simple. A context-free grammar describes certain linguistic patterns which correspond to commands. For example, the rule

$$\mathit{PROOF}1 :: \mathit{LET}\ \$\mathit{MATHVARIABLE}\$\ \texttt{be}\ \mathit{A}\ \mathit{ATTRIBUTION}$$

is used to generate the first command in Figure 2. In this rule, *LET* may concretely be `Let` or `let`, and *A* may concretely be `a` or `an`. Note that we do not, for example, consider the parts-of-speech of the components of the sentences in the proof. We consider it an open question whether the mathematical content of texts can be better extracted using a large collection of specific rules such as the one above or using more sophisticated natural language processing techniques. The MathLang approach described in [9] appears to be somewhere in between the two extremes. In the end, experience with a large number of texts will determine the best way to extract the content automatically. Nothing significant with respect to language processing should be concluded from the simple proof in Figure 1.

```
 1. let x:(in (A \cap (B \cup C))).
 2. hence ((x::A) & (x::(B \cup C))).
 3. hence ((x::A) & ((x::B) | (x::C))).
 4. clearly (((x::A) & (x::B)) | ((x::A) & (x::C))).
 5. hence ((x::(A \cap B)) | (x::(A \cap C))).
 6. hence (x::((A \cap B) \cup (A \cap C))).
 7. hence ((A \cap (B \cup C)) <= ((A \cap B) \cup (A \cap C))).
 8. let y:(in ((A \cap B) \cup (A \cap C))).
 9. clearly ((y::(A \cap B)) | (y::(A \cap C))).
10. hence ((y::A) & ((y::B) | (y::C))).
11. hence ((y::A) & (y::(B \cup C))).
12. hence (y::(A \cap (B \cup C))).
13. hence (((A \cap B) \cup (A \cap C)) <= (A \cap (B \cup C))).
14. clearly ((A \cap (B \cup C))==((A \cap B) \cup (A \cap C))
```

Fig. 2. Extracted Commands

The sequence of commands in Figure 2 can be fruitfully compared to the structured representation of the proof shown in Figure 3 of [1]. If one deletes the last line “4. Trivial” from Figure 3 of [1], then in both cases we have 14 lines corresponding to the same moves in the proof. However, [1] explicitly represents the scope of the assumptions $x \in A \cap (B \cup C)$ and $y \in (A \cap B) \cup (A \cap C)$. This information is not explicit in our representation. Note also that [1] is intended to be general with respect to formulas and terms, while we are using a concrete syntax for formulas and terms. The similarity between the two representations is not a coincidence. A close examination of the language in [1] (and its successors) inspired the parsing rules mentioned above.

One could use the parser to generate annotated text in the form of the language in [1], MathLang [9] or OMDoc [10]. However, this is a process independent of correctness of the proofs. In particular, the mutilated "proofs" described in Section 6 can be parsed and Scunak commands can be generated. Likewise, an annotated form of the mutilated "proofs" can be generated, even though the proofs may be technically incorrect. One can only judge correctness of proofs once one has a formal representation of the proposed proof and some formal notion of correctness. The Scunak type theory provides such a representation and such a notion of correctness. One could, of course, generate annotated text, then extract the Scunak commands for checking correctness from such annotated text. However, on such a simple proof there seems to be no motivation for using annotated text. One can simply regenerate the Scunak commands directly from the unannotated LaTeX source.

The Scunak proofreader keeps a set of alternative states of the (open) formal proof. Each command in Figure 2 is evaluated with respect to each alternative proof state giving a set of alternative proof states as a result. If there are no resulting proof states after the command is executed, then a "verification failure" is reported, along with a message indicating where the failure occurred (in the LaTeX source) and a general message indicating a reason for the failure.

An *open task* is $\langle X, \Gamma, G\rangle$ where X is a variable, Γ is a type context, and G is a type containing no free variables outside of the context Γ. Intuitively, we wish to instantiate X with a closed term M such that $(M\Gamma)^{\downarrow}$ has type G in context Γ. A *closed task* is $\langle \Gamma, G\rangle$ where Γ is a type context, and G is a type containing no free variables outside of the context Γ. As we shall see, a closed task can be used for bookkeeping with respect to discharging eigenvariables and hypotheses. A *task* is either an open task or a closed task. A *proof state* S is an open proof term P with free variables $X^1, \ldots, X^n$ and a list of *tasks*, where each free variable corresponds to an open task.

There are certain global variables which control the behavior of the Scunak proofreader. These include two subsets of the signature $\Sigma_{\mathbf{MU}}$. One subset Σ^u consists of signature elements corresponding to rules which can be used to try to justify steps in the proof. In analogy with first-order theorem proving, we say Σ^u is the *usable* set. Another subset Σ^e is a set of signature elements corresponding to rules which can be eagerly applied in the backwards direction. Only the elements in Σ^e can be used to apply a backwards step which introduces two new subgoals. The two sets Σ^u and Σ^e represent a model of the knowledge of the intended reader. Of course, to claim the proofreader is fully automatic, Scunak should choose the sets Σ^u and Σ^e. The proofreader is not fully automatic in this sense. We will consider different options for Σ^u, but we will explicitly restrict to the case where Σ^e contains only the rule `setextsub`. This rule is used to prove an equation between two sets by proving the two inclusions, and must be applied eagerly before Scunak begins to read the proof. The choice of Σ^e is arguably where the user is supplying the most explicit information about how the proof should proceed.

We initialize the Scunak proofreader with a claim corresponding to the goal $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ in the context Γ where A, B, and C are of type $\mathtt{obj}$ (equivalently, $\mathtt{set}$). Let G be the type $\mathtt{pf}\ \ ((A \cap (B \cup C) \mathtt{==} (A \cap B) \cup (A \cap C)))$ of proofs of this proposition. Scunak creates a variable X of type $\Pi A : \mathtt{obj}.\Pi B : \mathtt{obj}.\Pi C : \mathtt{obj}.G$ and an open task $\langle X, \Gamma, G\rangle$.

One initial proof state consists of the open proof X and the single task $\langle X, \Gamma, G\rangle$. Using $\mathtt{setextsub} \in \Sigma^e$, Scunak creates an open proof term

$$N := (\lambda A \lambda B \lambda C(\mathtt{setextsub} \cdots (YABC)(ZABC)))$$

with new variables Y and Z corresponding to proofs of the two inclusions. Scunak creates four more proof states, all of which correspond to the open proof term N, but differ in the list of tasks. One alternative is to have two tasks corresponding to instantiating Y by proving $(A \cap (B \cup C)) \subseteq ((A \cap B) \cup (A \cap C))$ and then instantiating Z by proving the other inclusion. A second alternative is to instantiate Z first by proving $((A \cap B) \cup (A \cap C)) \subseteq (A \cap (B \cup C)))$ and then instantiating Y. The third and fourth alternatives are the same as the first and second, except we include a third task: the closed task $\langle \Gamma, G\rangle$. As we shall see, this closed task must be included to check the final sentence of the proof of Figure 1. On the other hand, the proof in Figure 1 is valid if the final sentence is deleted, in which case the closed task should not be included. Hence we include alternatives with and without the closed task. In Figure 3, we show the five task stacks corresponding to the five initial proof states.

Let Γ be $A : \mathtt{obj}, B : \mathtt{obj}, C : \mathtt{obj}$.
Let G be $\mathtt{pf}\ \ ((A \cap (B \cup C)) \mathtt{==} ((A \cap B) \cup (A \cap C)))$.
Let $G1$ be $\mathtt{pf}\ \ ((A \cap (B \cup C)) \subseteq ((A \cap B) \cup (A \cap C)))$.
Let $G2$ be $\mathtt{pf}\ \ (((A \cap B) \cup (A \cap C)) \subseteq (A \cap (B \cup C)))$.

			$\langle Y, \Gamma, G1\rangle$	$\langle Z, \Gamma, G2\rangle$
	$\langle Y, \Gamma, G1\rangle$	$\langle Z, \Gamma, G2\rangle$	$\langle Z, \Gamma, G2\rangle$	$\langle Y, \Gamma, G1\rangle$
$\langle X, \Gamma, G\rangle$	$\langle Z, \Gamma, G2\rangle$	$\langle Y, \Gamma, G1\rangle$	$\langle \Gamma, G\rangle$	$\langle \Gamma, G\rangle$

Fig. 3. Task Stacks of Initial Proof States

Scunak evaluates the first command in Figure 2 with the five alternative proof states described above (see Figure 3). The command corresponds to the text "Let x be an element of $A \cap (B \cup C)$" in Figure 1. In order to evaluate this $\mathtt{let}$ command, the next task in the proof state should be an open task, and there must be some rule which introduces an eigenvariable of class type $A \cap (B \cup C)$ and which can be used to conclude the corresponding goal. For two alternatives, the current task is to instantiate Y by proving $(A \cap (B \cup C)) \subseteq ((A \cap B) \cup (A \cap C))$. Assuming the signature element $\mathtt{subsetI1}$ is in the usable set Σ^u, this element is found to have the appropriate type. For these alternatives, Scunak can imitate $\mathtt{subsetI1}$ for Y and create new alternatives. In each of the new alternatives, we create a version with a closed task corresponding to $(A \cap (B \cup C)) \subseteq ((A \cap B) \cup (A \cap C))$ and a version without such a closed task. For the other three initial alternatives, there are no corresponding rules in $\Sigma^u \subseteq \Sigma_{\mathbf{MU}}$, so the alternatives have no successors after the command is executed. In particular, if the current goal is to prove $((A \cap B) \cup (A \cap C)) \subseteq (A \cap (B \cup C))$, then introducing $x \in (A \cap (B \cup C))$ is clearly inappropriate. Hence we see that Scunak can determine when making an assumption (or introducing an eigenvariable) is a "correct" step. After the first command, it becomes clear that we are first proving the inclusion $(A \cap (B \cup C)) \subseteq ((A \cap B) \cup (A \cap C))$ by proving $x \in ((A \cap B) \cup (A \cap C))$.

Let Γ be $A : \texttt{obj}, B : \texttt{obj}, C : \texttt{obj}$.
Let Γ' be $A : \texttt{obj}, B : \texttt{obj}, C : \texttt{obj}, x : (A \cap (B \cup C))$.
Let G be $\texttt{pf}\ \ ((A \cap (B \cup C))\texttt{==}((A \cap B) \cup (A \cap C)))$.
Let $G1$ be $\texttt{pf}\ \ ((A \cap (B \cup C)) \subseteq ((A \cap B) \cup (A \cap C)))$.
Let $G2$ be $\texttt{pf}\ \ (((A \cap B) \cup (A \cap C)) \subseteq (A \cap (B \cup C)))$.
Let $G3$ be $\texttt{pf}\ \ (x \in ((A \cap B) \cup (A \cap C)))$.

$$\begin{array}{l|l|l|l}
 & & & \langle W, \Gamma', G3 \rangle \\
 & \langle W, \Gamma', G3 \rangle & \langle W, \Gamma', G3 \rangle & \langle \Gamma, G1 \rangle \\
\langle W, \Gamma', G3 \rangle & \langle \Gamma, G1 \rangle & \langle Z, \Gamma, G2 \rangle & \langle Z, \Gamma, G2 \rangle \\
\langle Z, \Gamma, G2 \rangle & \langle Z, \Gamma, G2 \rangle & \langle \Gamma, G \rangle & \langle \Gamma, G \rangle
\end{array}$$

Fig. 4. Task Stacks of Proof States after First Command

After the first command is evaluated, there are four alternatives. All four have a proof term corresponding to using `setextsub` and `subsetI1`, but differ in the inclusion or exclusion of closed tasks. In all four cases, the next task is an open task corresponding to proving $x \in ((A \cap B) \cup (A \cap C))$ in a context with $x : (A \cap (B \cup C))$ (i.e., x in the class type determined by $(A \cap (B \cup C))$). We display the task stacks for these alternative proof states in Figure 4, using W for the new variable.

The second command in Figure 2 is a `hence` command. A command `hence` P can be interpreted in two different ways, depending on whether the next task is open or closed. If the next task is closed and the closed task has type $\texttt{pf}\ \ P$, then a new proof state is created by removing this closed task from the list. (Commands 7, 13, and 14 in Figure 2 provide examples of this behavior.) On the other hand, if the next task is an open task $\langle X', \Gamma', G' \rangle$, then Scunak attempts to justify the fact by providing a closed term of type $\texttt{pf}\ \ P$. In order to justify this fact, some propositional reasoning is applied (removing some conjunctions and disjunctions) and then searching the usable set Σ^u for signature elements which can fill remaining gaps (without introducing new gaps). Suppose the gap is filled with a term Q. In the new proof state, X' is instantiated with $\lambda\Gamma(X''\Gamma Q)$ where X'' is a new variable and the task $\langle X', \Gamma', G' \rangle$ is replaced by the task $\langle X'', \Gamma'', G' \rangle$. where Γ'' is $\Gamma', w : \texttt{pf}\ \ P$. The idea is that we still have the goal of showing G', but now we have (a proof of) P in the context of this goal.

In our particular example, the fact that $x \in A$ and $x \in (B \cup C)$ holds follows by propositional reasoning (conjunction introduction) along with two rules `binintersectEL` and `binintersectER` which take $x \in (A \cap (B \cup C))$ to $x \in A$ and $x \in (B \cup C)$. Note that these rules must be in the usable set in order for the proofreader to justify this step.

Commands 3-6 in Figure 2 are all used to conclude a certain fact, as with the second command. (The only difference between `clearly` and `hence` is that `hence` indicates that particular attention should be paid to the last fact added to the context.)

After the sixth command is executed, there are several alternatives. For some of these alternatives, the next task is a closed task corresponding to $(A \cap (B \cup C))) \subseteq ((A \cap B) \cup (A \cap C))$. Command 7 in Figure 2 creates proof states by removing this closed task. Note that this has the effect of discharging the x and the hypothesis that x is in $(A \cap (B \cup C))$. The other alternatives disappear after the seventh command is executed.

Commands 8-12 correspond to proving the other inclusion and follow a similar pattern as commands 1-6. As with command 7, command 13 simply notes the end of this part of the proof (technically by deleting a closed task). Finally, command 14 notes the end of the proof by deleting the last closed task.

Since the closed tasks can be included or excluded, one can delete some of the sentences in the textbook proof (those corresponding to commands 7, 13, and 14) and Scunak can still verify the proof. Also, one sentence in the first paragraph (corresponding to command 4) can be deleted since this is only involves propositional reasoning. The shortened proof is shown in Figure 5. One could say that the proof in Figure 5 is at an appropriate level of granularity for Scunak. (This shortened version can be contrasted to the longer "patched" version in Figure 6 of [1].)

In order for Scunak to verify the proof, the usable set Σ^u must contain at least the rule for introducing subset and rules for introducing and eliminating binary unions and intersections. We ran the Scunak proofreader with three possible settings of Σ^u. First, with Σ^u_0 equal to eight elements corresponding to the rules we require about subset, binary union and binary intersection. Second, with Σ^u_1 containing all 79 rules which mention subset, binary union or binary intersection. Third, with Σ^u_2 containing all 435 rules in the **MU**-kernel. In the table below, we show the time (in seconds) taken to verify the two proofs using each usable set. From these results, it is clear that the usable set makes a significant difference.

Let x be an element of $A \cap (B \cup C)$, then $x \in A$ and $x \in B \cup C$. This means that $x \in A$, and either $x \in B$ or $x \in C$. Therefore, either $x \in A \cap B$ or $x \in A \cap C$, so $x \in (A \cap B) \cup (A \cap C)$.

Conversely, let y be an element of $(A \cap B) \cup (A \cap C)$. Then, either (iii) $y \in A \cap B$, or (iv) $y \in A \cap C$. It follows that $y \in A$, and either $y \in B$ or $y \in C$. Therefore, $y \in A$ and $y \in B \cup C$ so that $y \in A \cap (B \cup C)$.

Fig. 5. Shortened Textbook Proof

	Σ_0^u	Σ_1^u	Σ_2^u
Fig 1 Proof	2	62	457
Fig 5 Proof	3	20	251

Upon completing the verification, Scunak outputs the proof term. This proof term can be easily type-checked without any need for search.

6 Finding Bugs in Textbook Proofs

For the purpose of building a mathematical library, the most interesting proofs are correct proofs which give new facts for the formal library. On the other hand, when proofreading, one is usually searching for mistakes. We briefly consider eight erroneous "proofs" obtained by mutilating the proof in Figure 1. Instead of repeating the entire proof, we indicate the change introducing the error and describe the output of the Scunak proofreader. Computing a "reason" for the error can be problematic because the "reason" may be different in different alternative proof states. Scunak simply collects reasons and outputs the one which occurs the most often.

1. Change the first sentence so "then $x \in A$ and $x \in B \cup C$" reads "then $x \in B$ and $x \in B \cup C$." Scunak indicates that "$x \in B$ and $x \in B \cup C$" may not follow.
2. Change "Therefore, either $x \in A \cap B$ or $x \in A \cap C$," to be "Therefore, either $x \in A \cap B$ or $x \in A \cup C$," in fifth sentence. In this case, Scunak indicates that the statement *following* the mutilated statement cannot be verified. In particular, Scunak can verify the mutilated statement "$x \in A \cap B$ or $x \in A \cup C$," but cannot afterwards verify "so $x \in (A \cap B) \cup (A \cap C)$."
3. Change the conclusion of the first paragraph to be "This shows that $A \cap (B \cup C)$ is a subset of $(A \cup C) \cap (B \cup C)$." Scunak indicates that we have not shown this conclusion.
4. In the second paragraph, change "(iii) $y \in A \cap B$" to be "(iii) $x \in A \cap B$." Scunak signals that "Then, either (iii) $x \in A \cap B$, or (iv) $y \in A \cap C$" is not obvious. (Signalling that x is out of context would be preferable.)
5. Change the conclusion of the last sentence to read "$A \cap (B \cup C)$ and $(A \cap B) \cup (C \cap A)$ are equal" (commuting A and C). Scunak signals that the last sentence does not follow.
6. In the second sentence, change "Let x be an element of $A \cap (B \cup C)$" to be "Let x be an element of $A \cup (B \cap C)$" to corrupt the assumption about x. Scunak indicates that the type given for x does not seem to be correct.
7. Change the first sentence of the second paragraph to read "Conversely, let y be an element of $(A \cap C) \cup (B \cap C)$." Scunak indicates that the type given for y does not seem to be correct.
8. Remove the sentence "This means that $x \in A$, and either $x \in B$ or $x \in C$." The resulting proof is, in a sense, missing a step. Scunak indicates that the next sentence is not obvious.

We can also consider incomplete proofs. In such a case, the Scunak proofreader will not contain a complete proof upon termination, and will simply output a message indicating that the proof is not complete. Of course, an incomplete proof may also contain an error, in which case Scunak will signal the error before signalling that the proof is incomplete. One need not have a complete proof before invoking the proofreader.

7 Conclusion

Scunak can be used to proofread a simple mathematical proof written in LaTeX. For Scunak to proofread the proof the basic rules must be part of the usable subset of the full signature. More work on improving unification and indexing is required to handle the case when the usable set is large.

Because the proof of distributivity is so simple, it can act as a first touchstone for proposed approaches to checking proofs in texts. Any reasonable approach for verifying mathematical text should be able not only to verify this proof, but also find the errors in the mutilated versions of the proof. Then different approaches can be compared on a common, easy-to-understand problem.

The example also makes it clear when approaches are *not* intended to verify mathematical text. That is, any system which can read the proof in its mutilated forms without signalling an error is not intended to verify the proof content of mathematical text. Such a distinction is obviously important for understanding the intention of different systems.

Acknowledgements. Thanks to Magdalena Wolska and Alberto Gonźalez for helpful discussions about parsing natural language.

References

1. Serge Autexier and Armin Fiedler. Textbook proofs meet formal logic - the problem of underspecification and granularity. In Michael Kohlhase, editor, *Proceedings of MKM'05*, volume 3863 of *LNAI*, IUB Bremen, Germany, January 2006. Springer.
2. R.G. Bartle and D.R. Sherbert. *Introduction to Real Analysis.* John Wiley & Sons, New York, 1982.
3. Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.
4. Adam Grabowski and Christoph Schwarzweller. Translating mathematical vernacular into knowledge repositories. In Kohlhase [11], pages 49–64.
5. Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.
6. J. Roger Hindley. *Basic Simple Type Theory.* Cambridge University Press, 1997.
7. Gueorgui I. Jojgov. Translating a fragment of weak type theory into type theory with open terms. In Kohlhase [11], pages 389–403.
8. Gueorgui I. Jojgov and Rob Nederpelt. A path to faithful formalizations of mathematics. In Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec, editors, *MKM*, volume 3119 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2004.

9. Fairouz Kamareddine, Manuel Maarek, and J. B. Wells. Toward an object-oriented structure for mathematical text. In Kohlhase [11], pages 217–233.
10. Michael Kohlhase. Omdoc: Towards an internet standard for the administration, distribution, and teaching of mathematical knowledge. In John A. Campbell and Eugenio Roanes-Lozano, editors, *Artificial Intelligence and Symbolic Computation: International Conference AISC 2000*, volume 1930 of *Lecture Notes in Artificial Intelligence*, pages 32–52. Springer-Verlag, 2001.
11. Michael Kohlhase, editor. *Mathematical Knowledge Management, 4th International Conference, MKM 2005, Bremen, Germany, July 15-17, 2005, Revised Selected Papers*, volume 3863 of *Lecture Notes in Computer Science*. Springer, 2006.
12. Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In Joseph Halpern, editor, *Proceedings of the Sixteenth Annual IEEE Symp. on Logic in Computer Science, LICS 2001*, pages 221–. IEEE Computer Society Press, June 2001.
13. Frank Pfenning and Carsten Schürmann. System Description: Twelf–A Meta-Logical Framework for Deductive Systems. In Harald Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 202–206, Trento, Italy, 1999. Springer-Verlag.
14. Jason Reed. Proof irrelevance and strict definitions in a logical framework. Technical Report 02-153, School of Computer Science, Carnegie Mellon University, 2002.
15. Piotr Rudnicki and Andrzej Trybulec. On the integrity of a repository of formalized mathematics. In Andrea Asperti, Bruno Buchberger, and James H. Davenport, editors, *MKM*, volume 2594 of *Lecture Notes in Computer Science*, pages 162–174. Springer, 2003.
16. Freek Wiedijk. A new implementation of Automath. *J. Autom. Reasoning*, 29 (3-4):365–387, 2002.
17. Freek Wiedijk. Is ZF a hack? Comparing the complexity of some (formalist interpretations of) foundational systems for mathematics. *Journal of Applied Logic*, 4, 2006. to appear.

Capturing Abstract Matrices from Paper

Toshihiro Kanahori[1], Alan Sexton[2,⋆], Volker Sorge[2,⋆], and Masakazu Suzuki[3]

[1] Tsukuba University of Technology, Japan
kanahori@k.tsukuba-tech.ac.jp
[2] School of Computer Science, University of Birmingham, UK
{A.P.Sexton, V.Sorge}@cs.bham.ac.uk
http://www.cs.bham.ac.uk/~{aps|vxs}
[3] Faculty of Mathematics, Kyushu University, Japan
suzuki@math.kyushu-u.ac.jp
http://www.math.kyushu-u.ac.jp/~suzuki/

Abstract. Capturing and understanding mathematics from print form is an important task in translating written mathematical knowledge into electronic form. While the problem of syntactically recognising mathematical formulas from scanned images has received attention, very little work has been done on semantic validation and correction of recognised formulas. We present a first step towards such an integrated system by combining the *Infty* system with a semantic analyser for matrix expressions. We applied the combined system in experiments on the semantic analysis of matrix images scanned from textbooks. While the first results are encouraging, they also demonstrate many ambiguities one has to deal with when analysing matrix expressions in different contexts. We give a detailed overview of the problems we encountered that motivate further research into semantic validation of mathematical formula recognition.

1 Introduction

Since much of mathematics is still only available in printed, or at best in digital image, form, the automatic recognition and proper interpretation of mathematical texts would greatly enhance the extent of mathematical knowledge available electronically. Research in this area has mainly concentrated on the syntactic recognition of mathematical objects from images of texts or handwritten mathematics. This has led to systems capable of analysing scanned images of such texts to recognise the glyphs involved and identify their correct positional relationship to each other with quite high levels of accuracy [1,2,3,7,8,14]. The results of such an analysis can be used to print high quality versions of the scanned input or to provide access to printed mathematics for the visually impaired.

On the other hand, there has been little research on the problems of analysing specialist mathematical expressions (a) to provide feedback to inform the recognition process to obtain greater accuracy, (b) to provide higher level functions such as input into symbolic manipulation, mathematical assistant and theorem proving systems, and (c) to support mathematical knowledge management features such as domain specific search and storage systems.

[⋆] The authors' work was supported by EPSRC grant EP/D036925/1.

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 124–138, 2006.

As a first step in this direction we have concentrated on the syntactic and semantic capturing of one particular type of mathematical expression, namely on matrices as they commonly appear in textbooks. Such matrices often contain omissions, marked by a series of (usually) three dots called an *ellipsis*[1]. To this end we have connected the matrix recognition engine of the *Infty* system [13] with the algorithmic tools for the semantic analysis and computational treatment of underspecified matrix structures [10,11,12] implemented in the Computer Algebra system Maple [5].

The general idea is to capture matrix expressions from scanned documents and to find a proper semantics for them. If no semantics can be computed immediately, failure information can feed back into a more refined syntactic analysis of the expression. Once a semantics is computed, the matrix can be further utilised for computational purposes as well as for formal mathematical knowledge management. A proper and reliable recognition and automatic understanding of matrix structures can then be used as a basis for automated analysis of other mathematical expressions that are also constructed using ellipses such as equation systems, sequences and series or category diagrams.

Consider a matrix expression, as it might appear in a mathematical text:

$$A = \begin{pmatrix} a_1 & b & \cdots & b \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & b \\ 0 & \cdots & 0 & a_n \end{pmatrix} \tag{1}$$

Infty can recognise the expression and all its components and arrange them on a rectangular grid in order to produce a high-quality reprint. This structure then serves as an input to the semantic engine that analyses the well-formedness of the ellipses and the regions of the matrix defined by the ellipses.

First experiments with the combination have led to the enhancement of both systems separately as well as enabling us to identify a number of issues that need to be addressed in a fully automated feedback process. In this paper we present the technicalities of the combination as well as the first results of this work.

The paper is structured as follows: We first give an overview of the two systems involved, the *Infty* system that performs the syntactic analysis of the matrix structures (Sec. 2) and the Maple algorithms for the semantic analysis (Sec. 3). We discuss the issues involved in combining the systems in Sec. 4 and illustrate the working combination with an example in Sec. 5. We then present the preliminary results in Sec. 6 and in particular discuss the issues we have identified so far that need to be addressed in future research on that subject, before concluding in Sec. 7.

2 Syntactic Analysis

Infty is an integrated OCR system for scientific documents that contain mathematical expressions, developed mainly at the Suzuki Laboratory of Kyushu University. *Infty* has a number of novel and distinctive features:

[1] Not to be confused with the geometrical *ellipse*, although the plural form *ellipses* is the same for both.

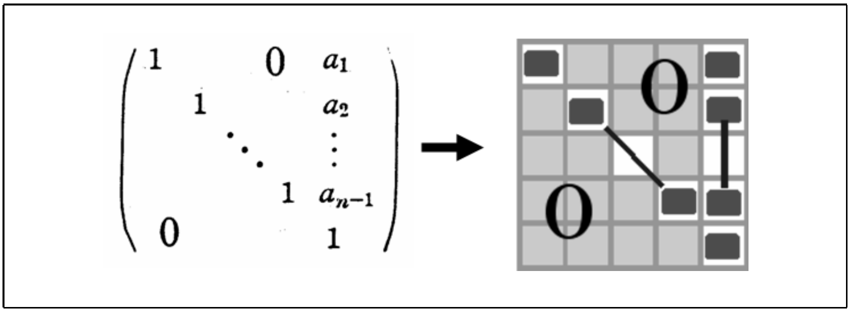


Fig. 1. The matrix elements are put on a 5 × 5 grid and saved as a matrix structure

- *Infty*'s character recognition engine can identify more than 500 kinds of characters and symbols (i.e. English and Greek alphabets, numerals, operators, parentheses, etc.).
- Segmentation of plain text from mathematical expressions is performed during character recognition while using recognition results for contextual information.
- Structural analysis is based on an optimisation framework and is stable in the face of both character/symbol recognition errors and structural ambiguity in the mathematical expressions.

Infty recognises scanned page images and analyses the structure of certain mathematical expressions, including *Abstract Matrices*, contained in the document. It can return the results of the recognition process in its own *IML* markup language (see Sec. 2.1), and in other formats including LaTeX and MathML.

From *Infty*'s point of view, recognising a matrix structure means constructing an internal model of the matrix described by the input image, which consists of a rectangular grid containing all terms and ellipses of the matrix. The result is sufficient for producing a clean, high quality, formatted output version of the matrix, including ellipses, but does not contain the semantic information necessary to work mathematically with the matrix.

The most complex step in this process is to determine the minimal rectangular grid compatible with a properly formatted version of the matrix. This is done by constructing a network of cells of the matrix, extracting horizontal and vertical projections from this network, constructing equations from these projections that capture the relative numbers of rows and columns required in the grid to represent the individual ellipses, and solving these equations to identify the grid dimensions and cell positions (Fig. 1). This results in a rectangular format that is convenient as input to the subsequent semantic analysis process. For details see [7,8].

2.1 IML

Infty outputs its recognition results in its own XML format. This format contains all the information about layout analysis, character recognition, mathematical structure analysis, and so on. The *IML* format is a cleaned up version of the original output that only contains the information necessary for actually rendering the document. Its format for normal text is based on HTML, but the format for the mathematical part is defined as follows:

- A symbol (or a character) is given within `<munit>...</munit>` tags, and all symbols on the same horizontal level are disassembled into separate units. For example, $n - 1$ results in:

  ```
  <munit>n</munit><munit>-</munit><munit>1</munit>.
  ```

- If a symbol has a sub- or superscript, or an upper or lower limit, the `<munit>` for the character contains a child element `<mlink>`. For example, λ_i^2 is represented as:

  ```
  <munit>lambda
         <mlink type="rsub"><munit>i</munit></mlink>
         <mlink type="rsup"><munit>2</munit></mlink>
  </munit>.
  ```

  Here the character is given as "lambda" based on the corresponding LaTeX command. The `<mlink>` type attributes, 'rsub' and 'rsup', indicate that the following `munits` are on a right subscript and superscript position of its parent respectively. This representation for a mathematical structure is one of the differences to MathML. The expressions $\vdots$, $\cdots$ and $\ddots$, are indicated by "vdots", "cdots" and "ddots".

- A matrix is indicated by `<marray>`, which has its rows as a list of `melems`. In each `melem`, columns are distinguished by a `<munit>Tab</munit>` delimiter. For example, a matrix row of the form $[xy, z, x]$ is represented as:

  ```
  <melem>
         <munit>x</munit><munit>y</munit>
         <munit>Tab<munit><munit>z</munit>
         <munit>Tab<munit><munit>x</munit>
  </melem>.
  ```

3 Semantic Analysis

The semantic analysis is realised via a dedicated data type called an *Abstract Matrix*, implemented in the computer algebra system Maple [5].

We have introduced abstract matrices in the context of document analysis in [11] and as a tool for interpreting and representing textbook style matrices as lambda expressions in the context of mathematical knowledge management in [12]. Full details of the algorithms involved are presented in [10]. In this section we give a brief overview of the algorithm and concentrate in particular on the features and restrictions of the input syntax that are necessary to lead to meaningful interpretations.

3.1 Syntax of Abstract Matrices

The input for an *Abstract Matrix* analysis is a rectangular arrangement of symbols in rows and columns, written in a Maple matrix expression, where each symbol is either

1. a *concrete* term, defined by not being one of the other types below.
2. a vertical, horizontal, diagonal, or anti-diagonal ellipsis
3. a single dot, or
4. a fill term.

Ellipses of type 2 are the ordinary suspension points $\vdots$, $\cdots$, $\ddots$ or $\iddots$. Every ellipsis must be terminated at both ends by a concrete term within the confines of the input matrix. Furthermore, the line of ellipsis symbols separating the two terminating concrete terms of an ellipsis must all be the same and compatible with the direction of the said line (e.g., a set of horizontal ellipsis suspension points could not be followed directly by those of a vertical ellipsis). Any diagonal or anti-diagonal ellipsis must be instantiated to a sequence of cells which have the same vertical and horizontal extent.

As example matrices that conform to the input syntax allowed for our semantic analysis consider matrix (1) in Sec. 1 as well as the following three matrices:

$$\begin{matrix} a_1 & & \mathbf{b} \\ & \ddots & \\ \mathbf{0} & & a_n \end{matrix} \qquad (2) \qquad \begin{matrix} 1 & \cdots & \cdots & 1 \\ \vdots & & \iddots & 0 \\ \vdots & \iddots & \iddots & \vdots \\ 1 & 0 & \cdots & 0 \end{matrix} \qquad (3) \qquad \begin{matrix} 1 & \cdots & \cdots & 1 \\ \vdots & \mathbf{0} & \iddots & 0 \\ \vdots & \iddots & \iddots & \vdots \\ 1 & 0 & \cdots & 0 \end{matrix} \qquad (4)$$

Matrix (2) represents the same abstract matrix as (1) in Sec. 1. However, in this case, the triangular regions above and below the diagonal are described by fill terms, that is the enlarged **b** and **0**. A fill term specifies that a certain region in a matrix contains only this particular term as elements. Fill terms can fill entire regions without explicit boundaries or can fill a region that is bounded by terms dissimilar to the fill term as in matrix (4). It is illegal to have two different fill terms within the same region.

Note also that matrix (3) differs from matrix (4) in that the upper triangle of the former contains only values, in this case the term 1, which can be interpolated from the terms on the surrounding boundary. This is indicated by the empty cell within the triangle. As we require our input to be a matrix structure where no cell can be empty, we translate such an empty cell into a single *dot* symbol in our input syntax. A single dot can occur in exactly two different situations:

1. Inside a closed polyline of ellipses and concrete terms it signifies that the region is filled in a manner consistent with the terms specified on the boundary of that region.
2. If it occurs next to a fill term or if there is a connected path of cells, each edge of which is a horizontal or vertical adjacent pair of cells containing dot or fill terms, connecting this dot term to a fill term, then it denotes the expansion of that fill term into the region occupied by the dot.

As concrete input for the Maple implementation of the semantic analyser the above three matrices would actually look like this:

```
[[a(1)    ,dot   ,fill(b)],
 [dot     ,ddots,dot     ],
 [fill(0),dot   ,a(n)    ] ]
```

```
[[1     ,hdots,hdots,1    ],
 [vdots,dot   ,adots,0    ],
 [vdots,adots,adots,vdots],
 [1     ,0     ,hdots,0    ] ]
```

```
[[1     ,hdots   ,hdots,1    ],
 [vdots,fill(0),adots,0    ],
 [vdots,adots   ,adots,vdots],
 [1     ,0       ,hdots,0    ] ]
```

Following Maple's convention for matrix expressions, subscript indices are translated into function applications, e.g. a_1 becomes $a(1)$.

Our approach to semantic analysis is based on interpreting abstract matrices as templates for the whole class of matrices of a particular shape. For instance, matrix (1), and likewise (2), can be interpreted as the class of all square matrices containing elements a_i on the diagonal, 0 below and b above the diagonal. For $n > 1$ we would get an ascending sequence of a_i elements on the diagonal. However, there are a number of subtleties in the interpretation of partially specified matrices. For example, some matrix expressions that appear in the literature have ellipses that are intended to instantiate into descending sequences of index values such as $a_{-1}, a_{-2} \ldots$. To accommodate those cases we also have to allow for descending sequences in (1) if $n < 1$ and, in the case of $n = 1$, it would even be possible that the diagonal contains constant elements a_1 and that the size of the matrix is independent of the value of n altogether.

While mathematically there may be no problem with allowing any arbitrary integer sequence for ellipsis indices, it is extremely rare to find anything other than simple increment or decrement by one for matrix ellipsis sequences in mathematical texts. For this reason, we impose the simplifying restriction that all ellipsis index variables may only change by -1, 0, or +1 between neighbouring cells. Furthermore we assume that every ellipsis is either vertical, horizontal, diagonal or anti-diagonal and that, for the latter two cases, their vertical and horizontal lengths must be equal. Thus a matrix with a diagonal ellipsis from its top left to its bottom right cells can be deduced to be square. Non-square matrices can be specified simply by omitting the diagonal ellipsis to get a rectangular region.

3.2 Parsing Abstract Matrices

The semantic interpretation of abstract matrices consists of three phases: (1) extracting ellipsis length constraints, (2) identifying regions in the matrix, and (3) extracting information on the content of each region.

Ellipsis Length Constraints: Ellipses in a matrix represent an expandable sequence of entries. However they cannot grow or shrink entirely independently of each other. We therefore represent lengths of ellipses as integer variables and use a weighted graph to capture their mutual relationships as a set of additive integer constraint equations, which we call *structural constraints*. This set can be simplified using standard simultaneous equation reduction techniques. If it can be fully solved then we call the matrix *concrete*, in which case a normal, fully specified matrix without any ellipses or fill terms can be generated from the abstract matrix.

Identifying Regions: A region is essentially a minimal cyclic path in the input matrix whose edges consist of concrete terms or ellipses. We define a region as a closed polyline of *generalised positions* of those boundary concrete terms, where a generalised position is a pair of expressions over integers, ellipsis length and dimension variables that defines the row and column position index of the cell in terms of the ellipsis length variables. Generalised positions can be computed by analysing the graph produced as part of the ellipsis length analysis. Once we have obtained the required generalised positions, we use a path-following algorithm for boundary detection on the input matrix to find the minimal ellipsis cycles and a flood-fill algorithm to find the interior of a region.

Extracting Meaning from Terms: In order to determine the correct content of a region we extract information from the concrete elements on its boundary. We use an anti-unification algorithm to construct a *generalised term* that can be unified with each concrete term on the region boundary. Each region must have associated with it a single generalised term. Furthermore, it must be possible to instantiate a region of variable position and extent in an abstract matrix to a region of fully determined position and extent for any consistent instantiations of the ellipsis lengths. This means that, for all positions within a region, we must be able to interpolate values for the unification variables of the generalised term based on the instantiation values of the boundary terms and the positions of those terms. We use a plane fitting algorithm to interpolate the values independently for each unification variable of the generalised term. When concretising an abstract matrix it must then be possible to compute integer values for unification variable for each element the region at least for some instantiations of the ellipsis length variables. Solving this problem adds more linear constraints — so-called *sub-term constraints* — to the system of linear ellipsis length constraints.
A semantic analysis of an input matrix can fail for a number of reasons:

1. The input syntax may be incorrect. For example the ellipses may not be terminated by concrete terms, a sequence of ellipses may not have the correct orientation etc. Such inputs are immediately rejected. On the other hand, if the inputs are correct, then it is guaranteed that the structural constraints, which involve only ellipsis lengths and constants, are satisfiable.
2. Next the anti-unifier may fail to find a suitable generalised term. For example no generalised term can be found for an ellipsis $a_{1,n} \ldots a_m$.
3. Even if suitable generalised terms can be found for all regions, the resulting subterm constraints may be inconsistent. For example consider $a_{1,n} \ldots a_{n,0}$: no possible instantiation of n would satisfy the resulting system.
4. Finally, even if the subterm constraints are satisfiable, there may be no interpolation across the region possible where the unification variables can be bound to integers in each of the cells contained in a region and still match the appropriate values for the concrete boundary terms.

4 Combining the Systems

In this section we briefly sketch how we interface the two systems by explaining how the major components of IML expressions are transformed to the input syntax of the semantic analysis. We point out the handling of some special cases, which corresponds to injecting mathematical knowledge into the parsing itself.

As described in Sec. 2, the IML expression representing an input matrix is a collection of rows enclosed by `melem` tags, where each row consists of a collection of elements enclosed by `munit` tags. Single `munit` cells do not necessarily constitute a single column element in the matrix. Rather we have `Tab` units that describe separations between cells of different columns, but that can also denote empty cells in a column. Thus our transformation algorithm makes a first

$$\begin{pmatrix} \lambda_1 & \delta t_{12} & \delta^2 t_{13} & \cdots & \delta^{n-1} t_{1n} \\ & \lambda_2 & \delta t_{23} & \cdots & \delta^{n-2} t_{2n} \\ & & \ddots & & \vdots \\ & & & \lambda_{n-1} & \delta t_{n-1,n} \\ & & & & \lambda_n \end{pmatrix}$$

```
<munit>delta
    <mlink type="rsup">
        <munit>n</munit>
        <munit>minus</munit>
        <munit>1</munit>
    </mlink>
</munit>
<munit>t
    <mlink type="rsub">
        <munit>1</munit>
        <munit>n</munit>
    </mlink>
</munit>
```

Fig. 2. A matrix from [4], page 28, and the IML description of its upper right term

pass for each row and translates the content of each `munit`. In a second pass it then combines the elements of the row to proper matrix cells and removes `Tab` elements that are column separators while leaving intact those that actually mark empty cells. We outline the two steps of the transformation algorithm considering as example the matrix given in Fig. 2.

In the first round of the transformation algorithm, the matrix is broken up into 5 rows and, for each row, each *unit* (i.e., an element enclosed in `munit` tags) is processed separately. In our matrix, cells such as the top right cell $\delta^{n-1}t_{1n}$ are recognised as two separate units and thus the first row contains 12 but the last row only 5 units. The concrete IML code for the $\delta^{n-1}t_{1n}$ is given on the right in Fig. 2. The first unit contains an additional link indicating an attached superscript term, and the second unit contains a subscript attachment. Both links are again composed of single units, which have to be assembled. Here we have to take a first decision on how to combine the expressions of single units. By default we combine the cells in superscripts using multiplication and those in subscript cells as separate indices. However we allow for certain exceptions; thus we treat arithmetic operators (e.g., `+,-,*`) and certain standard mathematical functions like trigonometric or logarithmic functions separately. For example, the single elements of the expression '`n, -, 1`' in our example will not be combined by multiplication but rather translated into '`n-1`'. If two units are separated by a comma, we always regard them as two separate index functions as in $t_{n-1,n}$. After the first pass of the transformation algorithm, all the units of a row are translated and the resulting matrix will look like this:

```
[[lambda(1), Tab, delta, t(1,2), Tab, delta, t(1,3), Tab,
                                   cdots, Tab, delta^(n-1), t(1,n)],
 [Tab, lambda(2), Tab, delta, t(2,3), Tab, cdots, Tab,
                                            delta^(n-2), t(2,n)],
 [Tab, Tab, ddots, Tab, Tab, vdots],
 [Tab, Tab, Tab, lambda(n-1), Tab, delta, t(n-1,n)],
 [Tab, Tab, Tab, Tab, lambda(n)]]
```

Obviously it is not yet in rectangular form. This is achieved by the second pass of the algorithm, in which each row is traversed once more in order to assemble the proper cells of the matrix. Adjacent cells that do not contain `Tab`, such as '`delta^(n-1) , t(1,n)`' above, are combined. Again we have to make a decision on how to combine the expressions. We essentially employ the same approach we have taken for superscript expressions. That is, unless there is an explicit arithmetic or otherwise recognised function symbol given, we combine two expressions by multiplication. While this approach at handling separated expressions and their sub- and superscripts works well for the case at hand and yields '`delta^(n-1)*t(1,n)`' as the resulting expression, it does not necessarily work in all cases, as we will discuss in Sec. 6.

In addition to combining elements, we also erase all the `Tab` cells that actually represent column separators. That is, a `Tab` is removed if it is in between two non-`Tab` cells, or if it is between a non-`Tab` and a `Tab` cell and there are still non-`Tab` cells in the remainder of the row. This, in particular, ensures that sequences of empty cells at the beginning or end of rows are kept intact. All the remaining `Tab`'s are replaced by the dot symbols used as the empty cell indicator for semantic analysis, and, similarly, *Infty*'s `cdots` elements are replaced by `hdots`. For our example matrix this yields the matrix below, which can now be processed by the semantic analyser.

```
[[lambda(1), delta*t(1,2), delta*t(1,3), hdots      , delta^(n-1)*t(1,n)],
 [dot      , lambda(2)   , delta*t(2,3), hdots      , delta^(n-2)*t(2,n)],
 [dot      , dot         , ddots       , dot        , vdots],
 [dot      , dot         , dot         , lambda(n-1), delta*t(n-1,n)],
 [dot      , dot         , dot         , dot        , lambda(n)]]
```

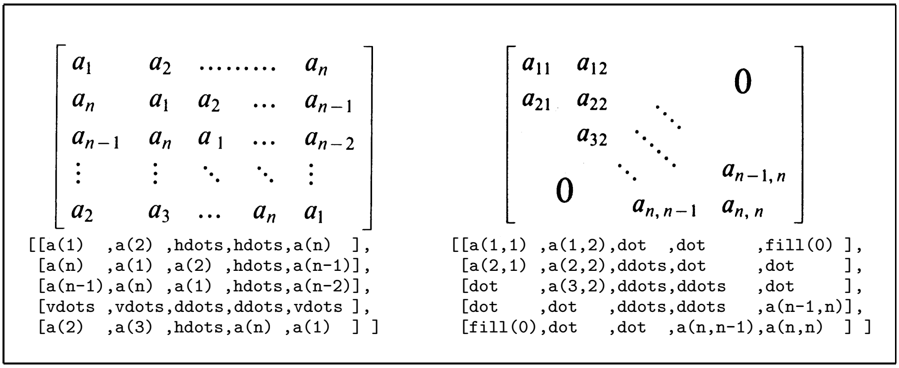
Unfortunately, we cannot generate a valid semantics for the above expression directly, since the lower triangular region does not contain any fill term, and therefore no valid content for the region can be determined. We will discuss this and other problems revealed by the semantic analyser and possible ways to overcome them in Sec. 6. However, we will first present two examples of matrices the analyser can actually assign a proper semantics to in the next section.

5 Examples

In this section we present two examples of matrices from [6] for which the semantic analysis of the input matrices succeeds. Note that we will only give the semantic description informally; for a more formal description of the Maple algorithms and their output we refer the reader to [10].

Figure 3 contains the two example matrices in the top row and their respective intermediate representations directly below them. Observe that the right matrix contains two fill terms. *Infty* recognises fill terms and indicates them in its IML output with an additional attribute "fill" for the `munit`. This is then parsed into the representation suitable for the semantic analyser, i.e. `fill(0)`.

The semantic analysis determines that the matrix consists of 7 regions: (a) Two regions containing a single cell only: a_{11} and a_{21}, (b) Three single ellipses: e_0: $a_{32} \cdots a_{n,n-1}$, e_1: $a_{22} \cdots a_{n,n}$, e_2: $a_{12} \cdots a_{n-1,n}$ and (c) two fill regions, each



$$\begin{bmatrix} a_1 & a_2 & \ldots\ldots\ldots & & a_n \\ a_n & a_1 & a_2 & \ldots & a_{n-1} \\ a_{n-1} & a_n & a_1 & \ldots & a_{n-2} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ a_2 & a_3 & \ldots & a_n & a_1 \end{bmatrix}$$

```
[[a(1)   ,a(2) ,hdots,hdots,a(n)  ],
 [a(n)   ,a(1) ,a(2) ,hdots,a(n-1)],
 [a(n-1),a(n) ,a(1) ,hdots,a(n-2)],
 [vdots ,vdots,ddots,ddots,vdots ],
 [a(2)   ,a(3) ,hdots,a(n) ,a(1)  ] ]
```

$$\begin{bmatrix} a_{11} & a_{12} & & & \mathbf{0} \\ a_{21} & a_{22} & \ddots & & \\ & a_{32} & \ddots & \ddots & \\ & & \ddots & \ddots & a_{n-1,n} \\ \mathbf{0} & & & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

```
[[a(1,1) ,a(1,2),dot   ,dot      ,fill(0) ],
 [a(2,1) ,a(2,2),ddots,dot      ,dot     ],
 [dot    ,a(3,2),ddots,ddots    ,dot     ],
 [dot    ,dot   ,ddots,ddots    ,a(n-1,n)],
 [fill(0),dot   ,dot  ,a(n,n-1),a(n,n)  ] ]
```

Fig. 3. Two example matrices from [6], pages 26 and 28

containing the term 0. The analysis also confirms that the ellipses are compatible, in that their mutual constraints are not inconsistent. In detail, we get that $e_1 = e_2 = e_0 + 1$ and that the boundaries of the fill region are of length $e_0 + 1$. This means that if we determine the length of any one ellipsis we automatically determine the length of all ellipses. The analysis further determines that the boundary terms of the regions (i.e., the end points of the ellipses in our case) can be anti-unified and that the ellipses can indeed be interpolated, for at least some values of n, with sequences of integer values that decrease or increase by one or that remain constant.

For the example matrix on the left in Fig. 3 we get a very similar result. This time we have 3 single cell regions, 3 single ellipses, and two triangular regions, which are, however, not simple fill regions. First, observe that the vertical and diagonal ellipses of these triangles are in columns that also contain single elements. But since these elements are not single cell regions and instead belong to flexible regions (i.e., regions with ellipses) they do not determine the length of the ellipses in question and thus the triangles. Second, note that all ellipses in the matrix are of equal length, say q, (with the exception of the horizontal ellipsis in the top row, which is of length $q + 1$) and thus the regions are isosceles with sides of length q. Finally, the boundary terms can be antiunified, with antiunifier a_α, where α is a unification variable. Given this information the semantic analysis can compute valid interpolation functions for the respective regions.

We demonstrate this for only one of the regions, which is given below together with its interpolation function:

$$p(i, j) = -\frac{3j - 3i - q + 1 - nj + ni}{q - 1} \qquad \begin{array}{ccc} a_1 & \ldots & a_{n-2} \\ & \ddots & \vdots \\ & & a_1 \end{array}$$

Here $p(i, j)$ is a function in the matrix positions i and j, n is the index variable referred to in the upper right term, and q is the ellipsis length as before. Thus, if $q = 3$, possible values for n would be 5 for the subscript of a to increase as one proceeds from left to right along the top row of the triangle, 3 to stay constant and 1 to decrease. q is crucial in determining the positions of the corners of the region. While the upper a_1 is at position $(3, 3)$ in the matrix, a_{n-2} is at position $(3, q+2)$, which corresponds to the end point of the horizontal ellipses between the terms. This, in particular, takes into account the case where the ellipsis is only of length 1 and results only in term a_1 (in fact, the system correctly handles the case when the length is 0, and the triangle collapses to nothing). Similarly, the lower a_1 is at position $(q + 2, q + 2)$, which defines the lower right corner of the matrix.

6 Preliminary Results

We have worked with a set of 29 matrices from three different textbooks [4,6,9]. This is not a random sample of matrix expressions as we explicitly chose a number that had interesting or unusual features as well as some more standard ones. Of those, *Infty* failed to syntactically parse only one. This was due to its current lack of support for antidiagonal ellipses. From the remaining 28, 12 could be successfully converted into a semantic representation immediately. After modifying the semantic analyser to inject fill symbols with a zero term to incomplete boundary regions (see below), an additional 4 matrices could be semantically parsed. An analysis of the failure to assign a proper semantics to the remaining 12 matrices has yielded several issues that have implications both for the syntactic and semantic analyses. We will summarise and illustrate those issues here and, for some, present a possible solution.

Incomplete regions: These are ones which are not fully enclosed by a closed polyline of ellipses and concrete terms (i.e., they extend up to the boundary of the matrix expression itself), and for whom no explicit fill term has been specified. There are three possibilities:

1. A fill term may have appeared in the expression but was not distinctive enough for the syntactic analyser to identify as such. This occurs in matrix (9), where the off-diagonal zeros were not distinguishable from normal concrete terms. A possible solution would be to identify concrete terms on the boundary or interior of the incomplete region that are not end terms of any ellipsis. Such terms are prime candidates for being mis-recognised fill terms and *Infty* could be queried to explicitly test this hypothesis.
2. The original expression that was scanned simply left the reader to interpret the incomplete region by context. An example of this can be seen in matrix (10), where there is no clear indication of what should appear on the upper right and lower left, although the writer almost certainly intended all such values to be zero. In this case, we can *inject* a zero fill term, i.e., assume that any incomplete regions has an implied zero fill term. However, this is not always desirable. For example, it may be difficult to distinguish whether this or the next situation is the true one. Nonetheless, adding this operation as a default behaviour allows four more of the matrices to be correctly and fully semantically parsed and there were two further matrices that had this problem among others.

$$\begin{pmatrix} a_1 & a_2 & \cdots & a_{n-1} & a_n \\ a_n & a_1 & a_2 & \cdots & a_{n-1} \\ & a_n & a_1 & & \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ a_3 & \cdots & a_n & a_1 & a_2 \\ a_2 & a_3 & \cdots & a_n & a_1 \end{pmatrix} \quad (5) \qquad \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 & b_{1\,n+1} & \ldots & b_{1k} \\ 0 & 1 & 0 & \ldots & 0 & b_{2\,n+1} & \ldots & b_{2k} \\ 0 & 0 & 1 & & 0 & b_{3\,n+1} & \ldots & b_{3k} \\ \vdots & & & \ddots & \vdots & \vdots & & \vdots \\ 0 & 0 & \ldots & 0 & 1 & b_{n\,n+1} & \ldots & b_{nk} \end{pmatrix} \quad (6)$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ 0 & a_{22} & a_{23} & \ldots & a_{2n} \\ 0 & 0 & a_{33} & \ldots & a_{3n} \\ \vdots & & & \ddots & \\ 0 & \ldots & & 0 & a_{nn} \end{pmatrix} \quad (7) \qquad \begin{bmatrix} a_0 & a_1 & a_2 & \ldots\ldots & & a_n \\ a_1 & a_2 & a_3 & \ldots\ldots & a_n & a_{n+1} \\ a_2 & a_3 & & \ddots & & a_{n+2} \\ \vdots & \cdot & & & \ddots & \vdots \\ & a_n & & & & \\ a_n & a_{n+1} & a_{n+2} & \ldots\ldots & & a_{2n} \end{bmatrix} \quad (8)$$

$$\begin{pmatrix} \mu_1 & & & & \\ & \mu_2 & & & \\ & & \ddots & & 0 \\ & & & \mu_r & \\ & & 0 & & 0 \end{pmatrix} \quad (9) \qquad \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & -1 & & & & & & & \\ & & & & \ddots & & & & & & \\ & & & & & -1 & & & & & \\ & & & & & & \cos\theta_1 & \sin\theta_1 & & & \\ & & & & & & -\sin\theta_1 & \cos\theta_1 & & & \\ & & & & & & & & \ddots & & \\ & & & & & & & & & \cos\theta_r & \sin\theta_r \\ & & & & & & & & & -\sin\theta_r & \cos\theta_r \end{pmatrix} \quad (10)$$

Fig. 4. Examples of matrices for which no proper semantics could be computed

3. There may be a missing boundary ellipsis. Matrix (7) demonstrates this issue. The right edge of the matrix needs an ellipsis to complete the boundary region. Injecting a zero fill term here instead would give an incorrect interpretation.

Missing ellipses: Ellipses other than boundary ones may be missing. Matrices (6) and (7) both show examples involving a sub-diagonal ellipse of zeros which is missing. Detection of the problem occurs only on checking that interpolation across a region is compatible with the concrete terms on its boundary, as even without the missing ellipsis, there is a complete region boundary.

Ellipsis connections: The semantic analysis input function requires that all ellipses extend from one concrete term to another, with the direction of the ellipsis dots being oriented appropriately. Matrix (7) has a horizontal ellipsis on the bottom row which is adjacent to an empty cell. Although this does not match the semantic analyser's input requirements, it does not cause any problems because *Infty* helpfully expands such ellipses in the appropriate direction.

More seriously, matrix (5) has a row of horizontal ellipses that are terminated at both ends by vertical ellipses. While there are some special cases that we could easily modify our code to handle, such as the one in this matrix where the horizontal ellipsis extends the full width of the matrix except for the vertical ellipses at the outer edges, the general case remains an open problem.

Bridging region cells: There is another problem in matrix (5). The ellipsis on the bottom row, $a_3 \ldots a_n$, occupies 3 adjacent cells in the input matrix. On the row above and the column in the middle of the ellipsis, there is the a_n term. This cell is diagonally adjacent to (i.e., it touches) both the terminal cells of the bottom row ellipsis. The end result, when the paths are calculated, is to force the bottom row ellipsis to be precisely 3 cells wide. This was almost certainly not intended by the author. The two concrete terms a_2, in cell $(2,3)$ and a_{n-1} in cell $(1,4)$ of the same matrix demonstrate the same problem.

Block ellipses: Matrix (10) has an ellipsis, the lower right one, that is intended to indicate that a whole 2 × 2 cell block of the matrix should be the element of iteration along the ellipsis. Currently the semantic analyser cannot handle such block matrix ellipses.

Open ended ellipses: We cannot handle ellipses that are not delimited by concrete terms at both ends. These types of ellipses might nevertheless be meaningful in a matrix representation and indeed do occur in textbooks. As an example consider the following matrices:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ & \cdots & \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}, \qquad \begin{pmatrix} a_{11} & a_{12} & \cdots & \\ a_{21} & a_{22} & \ddots & \\ \vdots & \ddots & \ddots & \\ & & & \end{pmatrix}$$

In principle, modifying the analyser to deal with this pattern of open ended ellipses does not appear to be very difficult, although there may be subtle interactions with possible error cases involving incomplete boundaries.

Badly constructed ellipses: Matrix (6) has an ellipsis that vertically connects 0 to 1. The semantic analyser will interpret that as an ellipsis that increments in steps of 1 starting at 0 and ending at 1, i.e., an ellipsis containing precisely 2 cells. Not only is this the wrong interpretation here, but the region to the left of this ellipsis is not satisfiable. To see this, note that the boundary cells of the region, reading from the 1 on the bottom row in a clockwise direction, are 1, up left to 1, up to 0, right to 0, down to 0 and back down to 1. The interpolation for terms within a region requires fitting a plane to the concrete boundary points of the region where the ellipsis index values of the concrete terms fit on the plane. No such plane fitting is possible in this case. It is not clear what can be done to recover this situation.

Increment other than by 1: In matrix (8), the diagonal ellipsis index increases in steps of 2, instead of 1. Since we make the simplifying assumption that ellipses always increment or decrement in steps of 1, or stay constant, we can not currently handle this situation. The reason for the simplifying assumption was the impossibility of dealing with arbitrary sequences, which, anyway, do not appear in the literature (or so we thought). Clearly we now have a reasonable concrete example to the contrary and we should relax this restriction at least slightly.

Index multiplication and function application: Also in matrix (8), we have an issue to do with the fact that the bottom right index is $2n$. Our system currently recognises this as a double index, $a_{2,n}$, as that is a locally valid interpretation. A related problem is that of function applications which omit the application brackets such as the $\cos\theta$ in matrix (10). This type of problem was discussed in Sec. 4. There are a number of ways that a semantic analyser can deal with it. Here the obvious correction is based on the construction of the anti-unifier that must find a generalised term for the two terminal concrete terms of this ellipsis. Clearly this would only be possible in this case if the $2n$ were interpreted as a multiplication. However, there are other, more subtle, cases, where an ambiguity may remain. In such cases, more than one anti-unifier is possible. For example, an ellipsis of the form $a_{pq} \dots a_{rs}$ has two possible interpretations: where a has a single or a double index. Further context information may be able to help in some such cases.

7 Conclusion

We have presented a first approach at a semantic validation of syntactically recognised mathematical expressions. While our current system is restricted to matrices, we believe that the techniques we develop will be applicable in other mathematical contexts. In particular, matrices exhibit many features that are widely used in many mathematical expressions, such as ellipses, sub- and superscripts, multiple indices, two-dimensional layout, etc. The experiments we have conducted so far have already led to some improvements in both systems.

Our failure analysis of the matrices we cannot handle yet identifies paths for improvement on several layers: (1) The semantic analyser can be enhanced to handle more cases, such as block matrices, as well as to exploit failure information internally. For instance, failure information from the anti-unifier could lead to correction of the input matrix without the need to consult *Infty* again. For example, we could then solve the problem where a_{2n} was translated as a double index instead of the index $2 \times n$. (2) While the failure messages of the analyser already gives us information abut what has gone wrong, no feedback is generated yet that can be directly used by *Infty* to improve its recognition. Direct feedback could be used to clarify the nature of certain terms, to correct possible errors of the recognition, to give alternative interpretations of the input, or to give an element of validation when batch processing large collections of documents. (3) Finally, it has become apparent that there are some problems that can only be disambiguated by using additional context information that is not given in the matrix expression alone. Examples where additional information is necessary are, for instance, treating particular function symbols introduced in a text correctly or determining the correct terms for a region left empty if the default instantiation of a 0 fill term fails.

All this not only suggests that there is significant scope for research into semantic analysis of single expressions, but also that better techniques for extracting and handling important information from mathematical texts in general are needed.

References

1. R. Anderson. *Syntax-Directed Recognition of Hand-Printed Two-dimensional Mathematics*. PhD thesis, Harvard University, Cambridge, MA, 1968.
2. D. Blostein and A. Grbavec. Recognition of mathematical notation. *Handbook on OCR and Document Analysis*, ch.22, p.557–582. World Scientific Publishing, 1997.
3. K. Chan and D. Yeung. Mathematical expression recognition: a survey. *International Journal on Document Analysis and Recognition*, 3(1):3–15, 2000.
4. P. Ciarlet, B. Miara and J. Thomas *Introduction to numerical linear algebra and optimisation*. Cambridge Univ. Press, 1989.
5. A. Heck. *Maple Manuals*. Springer, 3rd edition, 2003.
6. R. Horn and C. Johnson. *Matrix Analysis*. Cambridge University Press, 1990.
7. T. Kanahori and M. Suzuki. A recognition method of matrices by using variable block pattern elements generating rectangular areas. In *GREC'2001*, *LNCS*2390, p. 320–329. Springer, 2002.
8. T. Kanahori and M. Suzuki. Detection of matrices and segmentation of matrix elements in scanned images of scientific documents. *ICDAR'03*, p. 433–437, 2003.
9. R. Kaye and R. Wilson. *Linear Algebra*. Oxford Univ. Press, 2003.
10. A. Sexton and V. Sorge. Abstract matrices in symbolic computation. To appear in the International Symposium on Symbolic and Algebraic Computation 2006.
11. A. Sexton and V. Sorge. Semantic analysis of matrix structures. In *ICDAR'05*, p. 1141–1145. IEEE Computer Society Press, 2005.
12. A. Sexton and V. Sorge. Processing textbook-style matrices. In *Proc. of MKM-05*, *LNCS*3863, p. 111–125. Springer, 2006.
13. M. Suzuki, F. Tamari, R. Fukuda, S. Uchida, and T. Kanahori. Infty: an integrated ocr system for mathematical documents. In *DocEng'2003*, p. 95–104. ACM, 2003.
14. H. Twaakyondo and M. Okamoto. Structure analysis and recognition of mathematical expressions. In *ICDAR'95*, p. 430–437, 1995.

Towards a Parser for Mathematical Formula Recognition

Amar Raja, Matthew Rayner, Alan Sexton*, and Volker Sorge*

School of Computer Science, University of Birmingham, UK
{ug25ayr, ug98mxr, A.P.Sexton, V.Sorge}@cs.bham.ac.uk
http://www.cs.bham.ac.uk/~aps|~vxs

Abstract. For the transfer of mathematical knowledge from paper to electronic form, the reliable automatic analysis and understanding of mathematical texts is crucial. A robust system for this task needs to combine low level character recognition with higher level structural analysis of mathematical formulas. We present progress towards this goal by extending a database-driven optical character recognition system for mathematics with two high level analysis features. One extends and enhances the traditional approach of projection profile cutting. The second aims at integrating the recognition process with graph grammar rewriting by giving support to the interactive construction and validation of grammar rules. Both approaches can be successfully employed to enhance the capabilities of our system to recognise and reconstruct compound mathematical expressions.

1 Introduction

Automatic document analysis of mathematical texts is highly desirable to further the electronic distribution of their content. Having more mathematical texts, especially the large back catalogues of mathematical journals, available in rich electronic form could greatly ease the dissemination and retrieval of mathematical knowledge. To build a robust system with high accuracy in correctly analysing mathematical texts, it is necessary to combine an effective optical character recognition (OCR) system with higher level syntactic and semantic analysis. However, while this is fairly routine for ordinary document analysis, when dealing with mathematics the particularities of mathematical notations and the often two dimensional nature of mathematical expressions have to be taken into account. Therefore, to date there are only very few systems available to integrate both processes. (The *Infty* system is a notable exception [6, 11].)

In [10] we presented a novel approach to OCR for scientific and mathematical texts. It is based on a large database of glyphs[1] together with a recognition algorithm that employs features computed from recursive geometric moment invariants [1, 4]. The approach is well suited for recognising subtle differences in

* The authors' work was supported by EPSRC grant EP/D036925/1.

[1] A glyph is a single, connected, shape of pixels. Characters are often composed of more than one glyph, e.g. "j" contains two glyphs and "≡" contains three.

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 139–151, 2006.

characters such as different fonts and sizes, which are especially important in a mathematical context. The result of the recognition process is, for each recognised glyph, its best match from the database together with a LaTeX command that produces the glyph. The commands can then be used to reproduce an approximation of the input document by placing them at the original position of the recognised glyphs. We are currently extending this recogniser with higher level features that will enable both a more informed recognition process by generating feedback for the OCR as well as more advanced document analysis by recognising compound mathematical formulas and translating them into proper LaTeX expressions.

In this paper we present two higher level features we have recently integrated into our OCR system. The first feature, presented in Sec. 3, uses a well known technique from the document analysis literature, Projection Profile Cutting (PPC) [7, 12], and employs it in an innovative way. While the technique is traditionally used as a preprocessing step to segment mathematical formulas before an OCR step, we use it as a postprocessing step to our OCR system in order to reassemble the original structure of more complex mathematical expressions. This has three advantages:

1. With the information on size and position of glyphs in an expression, gained from the character recognition step, we can simply *compute* profile cuts rather then search for them as in the original technique.
2. Our recogniser is explicitly designed to be a glyph recogniser rather than a character recogniser. This is because problems of character segmentation and problems of character layout and decoration are often difficult to distinguish. The former are usually dealt with in a character recognition phase and the latter in a structural analysis phase. By using a glyph recogniser, these two, often conflicting, issues can be dealt with together during structural analysis. Thus we use PPC to treat character reconstruction from glyphs as just more structural analysis of the same form as reconstructing entire formulas.
3. The knowledge of the recognised glyphs and their original position gives us a uniform handle to overcome the old problem of PPC, namely that it cannot deal in a uniform way with enclosed characters.

The second higher level feature, described in Sec. 4, explores graph grammar rewriting [2, 5] approaches to the structural analysis of mathematical formulas. A graph is constructed where the nodes are the recognised glyphs of the formula and the edges record the spatial relationships between the nodes. A set of graph rewriting rules record subgraph patterns which, when matched, can be rewritten to non-terminal nodes which record the recognised formula sub-expression. The resulting graph can be further rewritten until, finally, the graph consists of a single node containing the fully recognised formula.

Such an approach depends critically on having a database of rewrite rules that describe the graphical grammar of mathematical formulas. This database is unlikely ever to be complete, given not only the huge range of mathematical conventions currently in use but also mathematicians' propensity to invent new ones. Furthermore developing graph grammar rules is notoriously subtle and error-prone. Therefore, there is a need to be able to quickly and easily create new rules and to visualise the graph rewriting process of rule sets on actual graphs of mathematical formulas. We describe our first steps in developing a tool to interactively and easily create such rules from an analysis of images of formulas and to explore the operation of graph rewriting with these rules.

2 Database-Driven Mathematical OCR

In [10] we have presented a database-driven approach to mathematical OCR by integrating a recogniser with a large database of LaTeX symbols in order to analyse images of mathematical texts and to reassemble them as LaTeX documents. The recogniser itself is based on a novel application of geometric moments that is particularly sensitive to subtle but often crucial differences in font faces while still providing good general recognition of symbols that are similar to, but not exactly the same as, some element in the database. The moment functions themselves are standard, but rather than being applied just to a whole glyph or to tiles in a grid decomposition of a glyph, they are computed in every stage of a recursive binary decomposition of the glyph. All values computed at each level of the decomposition are retained in the feature vector. The result is that the feature vector contains a spectrum of features from global but indistinct at the high levels of the decomposition to local but precise at the lower levels. This provides robustness to distortion because of the contribution of the high level features, but good discrimination power from those of the low levels.

Since the recogniser matches glyphs by computing metric distances to given templates, a database of symbols is required to provide them. We have developed a large database of symbols, which has been extracted from a specially fabricated document containing approximately 5300 different mathematical and textual characters. This document is originally based on [8] and has been extended to cover all mathematical and textual alphabets and characters currently freely available in LaTeX. It enumerates all the symbols and homogenises their relative positions and sizes with the help of horizontal and vertical calibrators. The single symbols are then extracted by recognising all the glyphs that a symbol consists of, as well as their relative positions to each other and to the calibrators. Each entry in the database thus consists of a collection of one or more glyphs together with the relative positions and the code for the actual LaTeX symbol they comprise. The basic database of symbols is augmented with the precomputed feature vectors employed by the recogniser.

The recogniser returns all the glyphs it encounters on the input document and for each glyph, a sequence of alternative characters in diminishing order of quality of visual match. In addition it provides information on the coordinates of the original glyph in the input document and on the size of this glyph by specifying its bounding box. The latter information is particularly useful in determining the actual size of a character in question in order to find the matching font size or scaling factor for the database match. The former information is exploited for reproducing the input document. As previously we had no semantic analysis or syntactic parsing of the results to provide feedback or context information to assist the recognition, we could only reconstruct a document by composing a LaTeX picture environment that explicitly places, for each recognised character, the appropriate LaTeX command in its correct location.

3 Projection Profile Cutting

Projection profile cutting (PPC) is a technique that is widely used in different areas of document analysis. For the analysis of mathematical expressions it was introduced by Okamoto *et al* [7] for the case of printed mathematics and by Faure and Wang [12] for the case of handwritten mathematics. Other work that uses related techniques is, for instance, reported in [3]. All these projects have in common that they apply the PPC as a preprocessing step for the actual character recognition step to gain information on the formula structure and thereby to ease the recognition as well as, possibly, to correct symbols. In our case the aims and sequence of operations are rather different: We have already performed the glyph recognition of a mathematical text or expression. Thus we have, for each glyph in the text, its position on the page and the size of its bounding box together with a priority list of glyphs from our database that has been computed as the best matches by the recogniser. We now want to use PPC (1) to support the correct recognition and reassembling of multiglyph characters, and (2) to recursively assemble single characters to larger compound expressions that eventually can be expressed in a meaningful LaTeX command. One advantage of our approach is that we can deal uniformly and effectively with formulas that are both vertically and horizontally enclosed. These are generally inaccessible to traditional PPC techniques, where only some cases can be dealt with by including specialised rules. In the remainder of this section we will first introduce the technique in general and present the advantages of our approach by handling an expression containing a square root.

3.1 Basic Technique

The basic idea of PPC is to put straight projection lines in between components of an expression in order to separate the expression into elements that do not overlap. We start first with a vertical projection and then perform, on the resulting components, a horizontal projection. If no horizontal projection is possible we have obtained an *atomic component* of the expression, otherwise we proceed recursively until all the atomic components are found.

The fundamental element of PPC is the task of grouping the symbols by finding out which symbols overlap. We can define *vertical overlap* of two symbols as the situation where we cannot find a straight vertical projection that passes between the two symbols. Analogously, we can define *horizontal overlap* when we cannot perform a horizontal projection. To illustrate this consider Fig. 1: A symbol is essentially given by the height and width of its bounding box, as

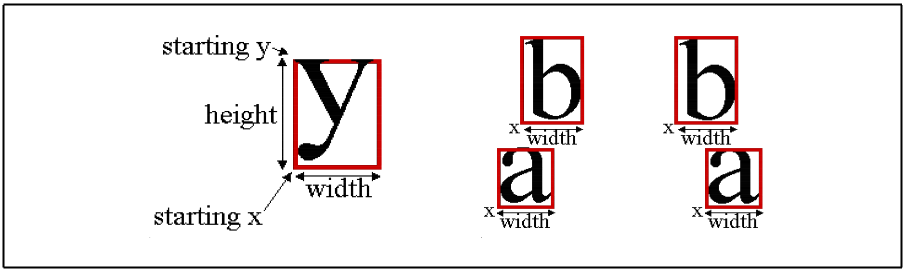
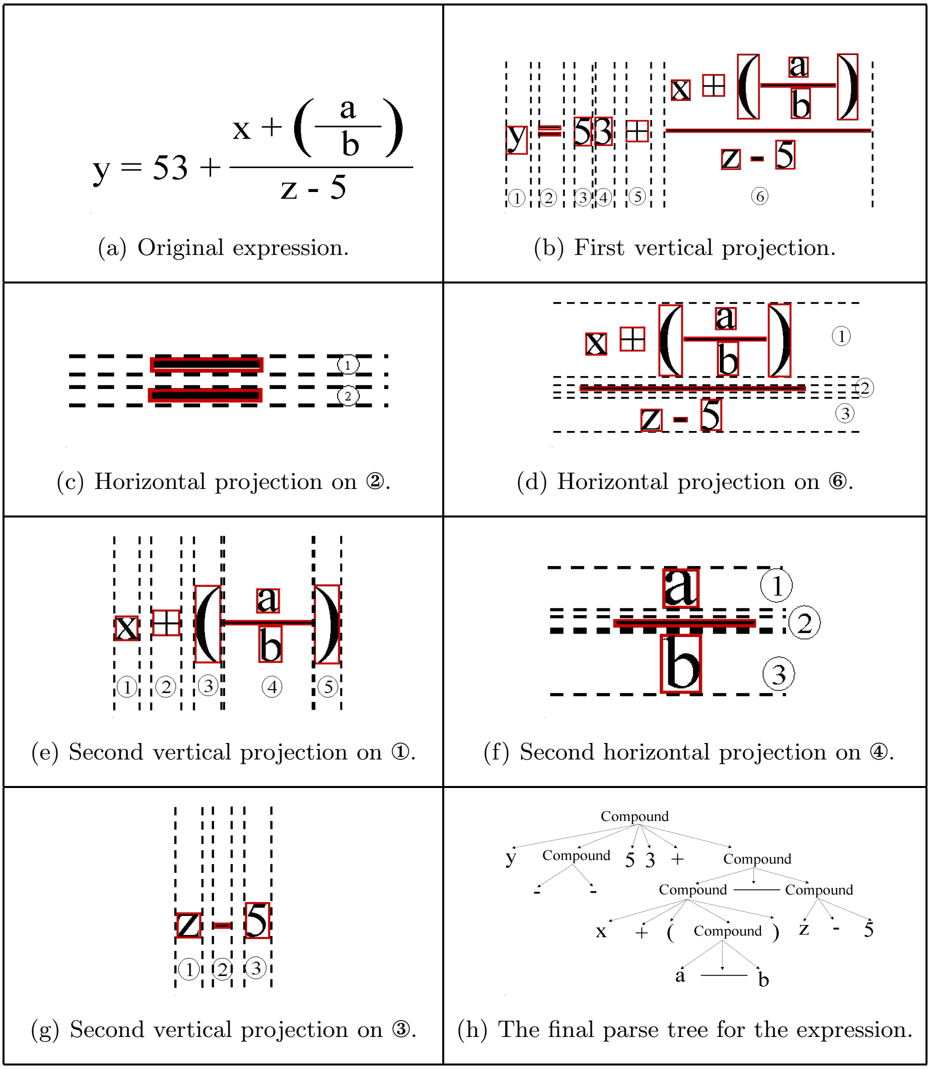


Fig. 1. Bounding box coordinates and overlap of symbols

displayed by the character 'y' on the left hand side. Now if the characters are arranged in a way that the bounding boxes overlap on a vertical line or a horizontal line one cannot perform the respective projection. The two possible cases of vertical overlap are, for instance, given on the right hand side in the figure.

Two symbols don't need to overlap directly for them to be classed as overlapping, they can overlap indirectly via a third symbol. Consider, for example, the expression in Fig. 2(a). The 'a' and the 'z' do not overlap but because the 'a' overlaps with the large divide sign and the 'z' also overlaps with the large divide sign then the 'a' and 'z' are classed as overlapping and are therefore grouped together in a sub-expression. To successfully group the symbols in the right subexpressions the order in which the symbols are checked for overlap is vital. We therefore sort the symbols in descending order by their width for vertical projections, and by their height for horizontal projections. This would mean the first two symbols to be checked for overlap would be the two divide signs for vertical projection. Since they indeed overlap, they will be grouped in the same subexpression together with all other symbols that overlap with them. Observe that this approach cuts out the search for the projection lines that is necessary in the traditional approach when segmentation is applied before the recognition process. In our approach we can simply compute overlaps using the bounding box information of each glyph and thus group the characters without search.

The entire PPC for the expression in Fig. 2(a) is given in the remainder of Fig. 2. Since we have the data from the OCR of the expression we already know exactly the position and sizes of the bounding boxes for the single glyphs, which simplifies the projection phase considerably. The first vertical projection, given in 2(b), then yields six different components. For components ① and ③–⑤ no horizontal projections are possible and we have atomic components. In fact, it is not necessary to test this explicitly since we can infer this already from the OCR data. Component ② on the other hand can be split again by the horizontal projection given in 2(c) to yield two atomic components. For the fraction in component ⑥ we need several recursive projection steps to fully analyse the expression. The first horizontal projection results in three components where only the fraction bar is atomic. While the denominator can be fully decomposed in another vertical projection (2(g)), the vertical projection on the enumerator



(a) Original expression.

(b) First vertical projection.

(c) Horizontal projection on ②.

(d) Horizontal projection on ⑥.

(e) Second vertical projection on ①.

(f) Second horizontal projection on ④.

(g) Second vertical projection on ③.

(h) The final parse tree for the expression.

Fig. 2. Recursive PPC example

(2(e)) yields another embedded fraction as non-atomic component. This one can finally be decomposed in another horizontal projection (2(f)).

The result of the PPC is a parse tree that details the subcomponent relationship of the single glyphs in the expression (see Fig. 2(h)) This tree can now be used to assemble the resulting LaTeX expression that reproduces the original input expression. It is worth noting that fraction bars in the expression can be determined as the LaTeX `\frac` command with the help of the OCR output. The resulting LaTeX expression is then of the form:

```
y = 53 + \frac{x + (\frac{a}{b})}{z - 5}
```

Observe that for clarity and to preserve space, we have omitted a `\mathrm` argument that actually embeds each of the alpha characters. Another noteworthy character in the above expression is the equality sign, whose reconstruction is necessary from the glyphs identified by the recognition engine. The next section explains how the combined ‘=’ is obtained.

3.2 Recognising Multi-glyph Characters

During the OCR process each glyph is recognised separately and matched against the glyphs in the database. This yields a priority list of matching glyphs, with the best matches coming first. Thus, for reasons of noise, distortions due to scaling, artifacts of the scanning process etc., a glyph belonging to a character that is composed of multiple glyphs, might not have, as its identified closest match in the glyph database, the corresponding glyph of the appropriate character. For instance, the best match for a single bar of the equality sign could be a minus sign or the ‘$\cdot$’ of a character ‘j’ might be best matched with dots from any one of multiple other symbols.

We can now exploit the results of the PPC to choose from the list of best matches for each glyph returned by the recogniser and reconstruct the correct character. When we find several atomic components in a single subexpression that are not separated by any non-atomic subexpression and that are within a certain distance threshold, the program parsing the expression tree attempts to find a single multi-glyph character that might match exactly with the components. It thereby considers a fixed number of multi-glyph characters it finds in the list of best matches given by the OCR program.

For example, for the equality sign it would take the two horizontal bars of the subexpression and find, in the matches list, the symbol ‘=’, which indeed contains two bars with the right gap. For the character ‘j’, the OCR system returns ‘$\cdot$’ and ‘$\jmath$’ (i.e., the LaTeX commands `\cdot` and `\jmath`) as best matches. But since the ‘$\cdot$’ of the ‘j’ overlaps vertically with the lower part of the ‘$\jmath$’ they are grouped together in a vertical projection, which allows the program to identify the complete symbol ‘j’ which is in the list of best matches.

3.3 Handling Enclosed Expressions

PPC is a very effective tool for determining the spatial relationships between symbols in mathematical expressions. However, the method fails if symbols in an expression are both horizontally and vertically enclosed. Then neither vertical nor horizontal projection can penetrate the expression to extract all components of the formula. The classical example of a mathematical symbol that is impenetrable is the root symbol. However, there are various other example of symbols that fence expressions from several sides, or even contain them entirely, in Mathematics and, particularly, in Logics. For our discussion consider the following expression:

$$\sqrt{\mathrm{a}+\sqrt{\mathrm{b}+\mathrm{c}}}$$

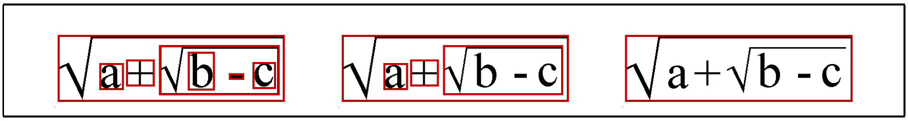


Fig. 3. Descending and assembling an expression with partially enclosed terms

Obviously, first vertical and then horizontal projection will yield no decomposition at all. However, from the information given by the OCR, we know that the expression consists of several non-connected glyphs and thus the projections should yield several atomic components. The program now further exploits the given information on the bounding boxes of the single glyphs to realise that there are indeed some glyphs fully contained in the bounding boxes of other glyphs. The leftmost expression in Fig. 3 shows the situation that presents itself to the program at this point. Before carrying on with the ordinary PPC, the algorithm first descends into the expression and extracts those components that are contained inside another glyph's bounding box. Thus in a first step it would extract $\mathrm{a}+\sqrt{\mathrm{b}+\mathrm{c}}$ from within the outer root symbol and in a second step $\mathrm{b}+\mathrm{c}$ from the inner root symbol. For this expression the regular projection algorithm will take over again and disassemble it into its components. Once the innermost atomic components have been been found the program will recombine those with the enclosing expression, that is the root symbol, and carry on doing so until it reaches again the top layer and the entire expression is analysed. The single steps of this process are depicted in Fig. 3. With the component tree constructed during the entire projection, we can then assemble the corresponding LaTeX command `\sqrt{a+\sqrt{b+c}}`. Again we have omitted the `\mathrm{}` around the 'a', 'b' and 'c'.

4 Graph Grammar Rewriting

Graph grammar rewriting [2, 5] is a very powerful technique that generalises string rewriting from standard string parsing to a graph parsing model. Rules specify a (possibly terminal) graph fragment to match and a non-terminal graph fragment to replace it with. Most such systems, when used for mathematical formula recognition, restrict the replacement fragment to be a simple node.

There are two major differences between our starting position and those of other projects using this technique. First, instead of using a character recogniser, we are using a glyph recogniser as discussed in Sec 2. Thus, part of our aim is to reconstruct characters from glyphs in the rewriter. The point of this is that the perennial problem of segmenting characters, which are sometimes broken, sometimes touching other characters, can often only be resolved with contextual, and sometimes semantic information. It is difficult to provide the necessary range of such information to a character recogniser, as that information is generally discovered by the structural analyser. Transferring responsibility for character assembly from the optical recogniser to the structural analyser provides a more modular structure that allows simpler interactions between the recogniser and the structural analyser. This in turn allows easier development of more sophisticated techniques for applying structural and contextual information to the identification of characters.

Second, our intention is to provide a test framework for exploring different rule sets, and, eventually, different types of graph rewriting, rather than, at this point, a definitive graph rewriting system and a definitive set of rules for mathematical formula recognition. The tool we built to do this provides assistance for matching graph grammar rules to graphs, identifying conflicting rules, choosing and applying a subset of those rules and visualising, graphically, the entire process. The idea is that providing a very convenient way to manually parse the formulas and graphically add new rules dynamically, provides an excellent environment in which to design and develop large and complex rule sets.

4.1 Constructing the Graph

A critical factor in any graph grammar rewriting approach is how to build the initial graph. If too many edges are generated then the resulting graph matching and parsing complexity may be too high. Too few edges mean that important connections between glyphs are overlooked and the formulas cannot be recognised. Lavirotte and Pottier construct their graph using compass point directions from each character [5]. While they report good success with this decision, our situation where we also need to reconstruct characters from glyphs may require finer distinctions. Also they do not report that they can handle enclosing constructs such as square roots.

Our graph is created initially with the best matching glyphs from the database for the image being analysed as the nodes. We could generate the complete graph in order to construct the edges but that would add considerably to the computational cost of the graph submatching algorithm. Also, it would not, we believe, add practical matching options to the system as the rewrite rules tend to work locally in the graph: neighbouring sets of nodes are rewritten into single nodes in a bottom up fashion. Hence connections between nodes corresponding to glyphs that are spatially distant in the image are unlikely to be useful. Therefore we chose to build our edges on a *line of sight* basis. Thus we create an edge between the nodes for two glyphs if there is a direct line between a pixel of one and a pixel of the other that is not obscured by any other glyph. In our current implementation, for performance reasons, we simplify that to a line from the bounding box centre of one to any point in the bounding box of the other. The edges are annotated with the centre point distances and relative directions which can be easily generalised to fuzzy distances and directions. This approach supports enclosures in a direct and simple way.

4.2 Rules

The rewrite rules identify a *principal* node (or principal for brevity) around which the rewrite will take place. Each rule has a name. Terminal glyph nodes are named based on their glyph identifier, non-terminal nodes are named by the rules that created them. Rules contain constraint information about relationships between neighbouring nodes in the graph that must exist before the rule can be triggered. A number of attributes can be set for each connection required by a rule and are used to control the conditions that must be met for the rule to trigger. The system currently supports the following rule connection attributes:

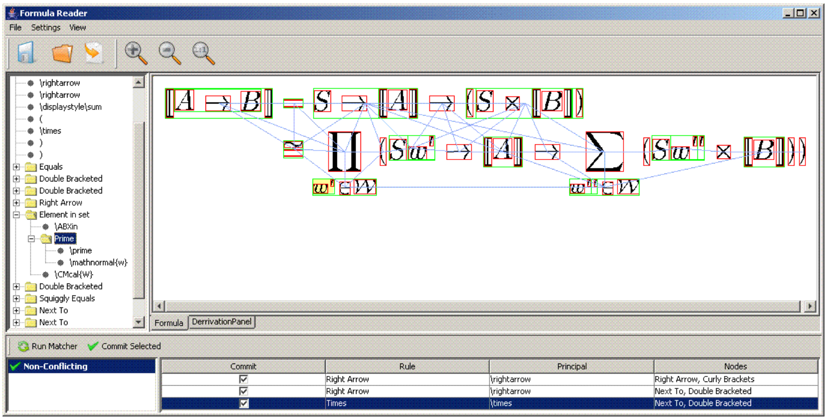
Higher. Top of the destination, but not the bottom, is higher than the principal.
Lower. Bottom of the destination, but not the top, is lower than the principal.
TopLower. Top of the destination is lower than the top of the principal.
BotLower. Bottom of the destination is lower than the bottom of the principal.
Above. Bottom of the destination is above the top of the principal.
Below. Top of the destination is lower than the bottom of the principal.
Left. Right edge of the destination is left of the left edge of the principal.
Right. Left edge of the destination is right of the right edge of the principal.
PartLeft. Destination is not left, but left edge is more left than the left edge of the principal.
PartRight. Destination is not right, but right edge is more right than the right edge of the principal.

All these attributes can be set to values indicating that the specified condition must be met by this connection, must not be met or can be ignored. An extra `tag` attribute is used to identify the nodes that must be found at the end of the connection. This can be an individual node name, a list of possible glyphs or a pattern of node names. Other than the qualitative constraints provided by the above attributes, quantitative constraints can also be specified controlling relative size of the principal and destination nodes (based on height, width or bounding box area) and length of the connection relative to the dimensions of the principal.

4.3 Subgraph Matcher

Our current matcher is simple and intended to provide us with a working test framework within which to explore graph grammar rule sets, visualisation of graph parsing and rule extraction tools before we proceed to a more sophisticated matching algorithm such as that of Rekers and Schürr [9].

The matcher takes the set of rules and systematically applies them to every glyph within the formula, taking the current glyph as the principal. All connections are first filtered by the destination tag and then all resulting combinations are checked. On a successful match the matching glyphs are placed in a collection ready for further analysis. In the event that a principal finds more than one object which match the results of a certain definition, all matches are returned. For example, for the formula $w' \in \mathcal{W}$, we could have, for a naïve implementation of the rule handling set membership, the following potential matches returned: "w element in set $\mathcal{W}$", and "Prime element in set $\mathcal{W}$". These, of course, conflict with the appropriate rule in this case which would also be returned; namely that for matching the "w'". They conflict because every pair of these three rules have



Commit	Rule	Principal	Nodes
☑	Right Arrow	\rightarrow	Right Arrow, Curly Brackets
☑	Right Arrow	\rightarrow	Next To, Double Bracketed
☑	Times	\times	Next To, Double Bracketed

Fig. 4. Formula reader with the rule matcher in operation

at least one node in common that they consume in order to rewrite the subgraph. Of course, only a rewrite of the w' rule will eventually result in a successful parse of the expression.

The ability to identify and explore sets of rules that conflict is an essential aspect of the design, debugging and development of rule sets. We say that two rules *conflict* if the intersection of the set of objects that they match is non-empty. A *conflicting rule set* is defined inductively as follows:

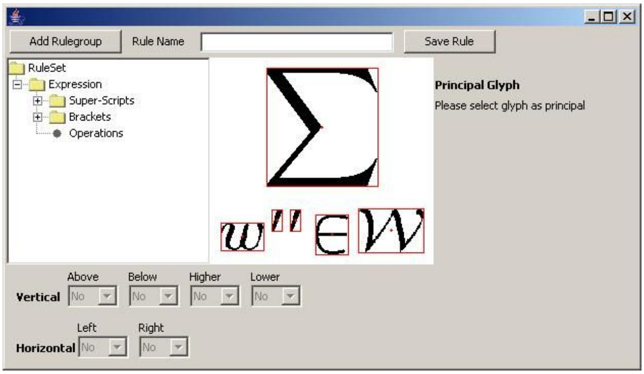
1. If rules r, r' conflict then $\{r, r'\}$ is a conflicting rule set.
2. If C is a conflicting rule set and r is a rule that conflicts with a rule $r' \in C$, then $\{r\} \cup C$ is a conflicting rule set.

A *conflicting rule set* $C \subseteq A$ is *maximal* in A if every $r \in A \setminus C$ does not conflict with any element of C.

Once the matcher has run on all rules, it partitions the set of matching rules into a set of maximal conflicting rule sets. All single element rule sets can be safely applied without conflicting with any other rule in the set of matching rules and are collected together in a *non-conflict group* of rules. Each other maximal conflicting rule set is recorded as a *conflict group*.

Once this phase is complete, the matcher attempts to resolve all the conflicts by finding a sequence of rule applications, one from each of the conflict groups, which will terminate with the least number of un-consumed vertices. This results in the most complete parse compatible with the rule set and is executed by applying each rule in turn and then recursively running the match process again until no further rewrites or matches are possible. In the event that multiple sequences may lead to the same resulting formula, the rule with the highest precedence within the rule set is identified.

The matcher is embedded in a rule visualisation and exploration tool called the *Formula Reader*. An example of this tool running on an expression is shown in Fig. 4. The main pane shows the formula under analysis at a point part way through an analysis. At this stage many of the lowest level rules have been



Principal Glyph

Please select glyph as principal

Vertical Above: No, Below: No, Higher: No, Lower: No

Horizontal Left: No, Right: No

Fig. 5. Rule builder in operation

applied and the graph has been rewritten to a significantly smaller number of non-terminals and remaining terminal nodes. Bounding boxes are drawn around each node as well as the sub-nodes that have previously been consumed. Edges are drawn between nodes that have not yet been consumed in the rewriting. All edges and nodes are colour coded to help the user identify appropriate parts of the formula. The current set of nodes in the graph is shown in the tree list in the upper left. One can drill down to see the internal structure of the node. Clicking on a node in the tree list highlights the corresponding node(s) in the main pane and vice versa. The small pane to the bottom left lists the current groups of non-conflicting or conflicting rules (in this case there is only one such group, which is a non-conflicting one). To the bottom right the set of rules in the group is shown and the user can choose which of them to apply. By selecting an individual rule, the corresponding nodes and edges in the main pane are highlighted to help visualise the current state of the process. Committing the choices will apply the rules and set the system up to run the next round of the matcher. The whole system allows quick and easy visualisation and testing of individual rules, interactions between rules and operation of entire rule sets.

Rules can be created and added to a rule set using the rule builder facility. Fig. 5 shows an example. When working on a formula, a user can select a group of nodes from which to construct a new rule. A *principal* node must be chosen and then rules for each connected node can be added specifying the appropriate relationships for each one. The ability to add new rules dynamically during the process of a manual parse of an equation significantly reduces the difficulties of constructing large and complex rule sets.

5 Conclusion

We have presented an extension to our database-driven mathematical OCR system by adding two higher-level analysis features.

First we have combined the well-known projection profile cutting method with our approach to OCR and improved the ability of our recogniser to find proper matching multiglyph characters and added the ability to compose complex mathematical expressions as compound LaTeX commands. But this work has also yielded a new and robust solution to one of the major flaws of the PPC technique: that of penetrating enclosed expressions. This new approach works uniformly for all enclosing symbols and does not rely on special cases for certain symbols.

Second we have developed a first version of an interactive graphical tool that significantly assists in the visualisation, analysis and development of graph grammar rule sets — one of the major barriers to the study and use of this powerful technology in mathematical formula recognition. This will enable us to develop graph grammar based approaches to the structural analysis of mathematical texts.

References

1. F. L. Alt. Digital pattern recognition by moments. *Journal of the ACM*, 9(2): 240–258, April 1962.
2. A. Grbavec and D. Blostein. Mathematics recognition using graph rewriting. In *Proc. of ICDAR'95*, pages 417–421, 1995.
3. J. Ha, R. M. Haralick, and I. T. Philips. Understanding mathematical expressions from document images. In *Proc. of ICDAR'95*, pages 956–959, 1995.
4. M.-K. Hu. Visual pattern recognition by moment invariants. *IEEE Transactions on Information Theory*, 8(2):179–187, Feb 1962.
5. S. Lavirotte and L. Pottier. Optical formula recognition. In *Proc. of ICDAR'97*, pages 357–361, 1997.
6. M.Suzuki, F.Tamari, R.Fukuda, S.Uchida, and T.Kanahori. Infty — an integrated OCR system for mathematical documents. In *Proceedings of ACM Symposium on Document Engineering*, pages 95–104, 2003.
7. N. Okamoto and B. Miao. Recognition of mathematical expressions by using the layout structures of symbols. In *Proc. of ICDAR'91*, pages 242–250, 1991.
8. S. Parkin. The comprehensive latex symbol list. Technical report, CTAN, 2003. available at www.ctan.org.
9. J. Rekers and A. Schürr. Defining and parsing visual languages with layered graph grammars. *J. Visual Languages and Computing*, 8(1):27–55, 1997.
10. A. Sexton and V. Sorge. A Database of Glyphs for OCR of Mathematical Documents. In *Proc. of MKM'05*, volume 3863 of *LNCS*, pages 203–216. Springer, 2006.
11. M. Suzuki, S. Uchida, and A. Nomura. A ground-truthed mathematical character and symbol image database. Technical report, Kyushu University, 2004.
12. Z. Wang and C. Faure. Structural analysis of handwritten mathematical expressions. In *Proc. of Ninth Int. Conf. on Pattern Recognition*, 1988.

Stochastic Modelling of Scientific Terms Distribution in Publications

Rimantas Rudzkis[1], Vaidas Balys, and Michiel Hazewinkel[2]

[1] Institute of Mathematics and Informatics, Akademijos st. 4,
LT-08663, Vilnius, Lithuania
[2] Centrum voor Wiskunde en Informatica, Kruislaan 413,
NL-1098 SJ Amsterdam, The Netherlands

Abstract. In this paper, we address the problem of automatic keywords assignment to scientific publications. The idea to use textual traces learned from training data in a supervised manner to identify appropriate keywords is considered. We introduce the transparent concept of identification cloud as a means to represent the semantics of scientific terms. This concept is mathematically defined by models of scientific terms stochastic distributions over publication texts. Characteristics of models as well as procedures for both non-parametric and parametric estimation of probability distributions are presented.

1 Introduction

During the last few decades e-publishing has been rapidly replacing conventional paper-based publishing making traditional means of classifying and indexing (manual assignment of keyphrases or category numbers from a subject classification system like MSC2000) unsatisfactory. Due to different requirements and constraints (depending on date, on a particular journal, etc.) and authors' individuality it is impossible to treat keyphrases from different publications in an uniform way. Nevertheless, everyday-growing collections of online mathematical knowledge have to be managed as efficiently as possible. Therefore new automatic online tools designed for indexing and classifying as well as retrieving relevant information are highly appreciated by those in the field.

Automatic assignment of a few scientific terms from controlled list (of so-called keywords and keyphrases) is one of the possible ways to manage the knowledge contained in publications. It may be viewed both as classifying and indexing routine depending on the nature of these terms as well as on the way of performing the assignment, e.g., the number of terms. Even more, the traditional keyword list serves either role — it is not as strictly classifying as the list of subject classifiers and not as exhaustive as the index would supposed to be. Having this mentioned we will further treat our problem as a classification one or more precisely — as a problem of automatic classification by means of statistical analysis of term–context relations over corpus.

We build on the idea of an identification cloud ([6], [7]) which is roughly described as a list of words and phrases (scientific terms) that are likely to be found in a text that is highly relevant to a certain scientific term (the so-called

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 152–164, 2006.

"owner" of the identification cloud). A fragment of the identification cloud in a chunk of text suggests the dominant term which is possibly relevant to that chunk and can be treated as a candidate to be a keyword.

The idea of a contextual track of a certain keyword is not a new one, in fact it is a common concept which lies in the background of almost every one of the text classification methods. However, term–context relations are often hidden from user in the state–of–the–art plain text classification algorithms meanwhile our approach is based on the explicit representation of these relations. The motivation of doing that is a practical one: identification clouds may well be used not only for automatic classification but also for solving many other problems related to scientific text processing ([6], [7]): estimation of the growth rate of a scientific field ([8]), identification and reconstruction of misspelled or incomplete phrases, construction of dialogue-mediated information retrieval engines, evaluation of distances in information spaces, sense disambiguation, text slicing, etc. On the other hand, these clouds are of great value themselves as they present some meta-knowledge of scientific field expressed in the language of the field. The reasoning above motivates and justifies this approach over the straightforward application of problem–specific methods, even though we realize that specific methods are always more efficient than the general ones. We also do get clear signals that such general methods and tools would be highly appreciated by those involved in e–publishing and related fields.

A substantial amount of researches related to our problem (classification by means of statistical term–context relations learned from corpus) have been conducted since early 1990s though in rather different fields including automatic translation, word sense disambiguation, information retrieval, document indexing and clustering, etc. The two main approaches differing in the manner of learning are used. The first one relies on the unsupervised learning (learning over unlabelled data) of distributional patterns of co-occurring data ([4], [9], [1]). In 1990 Deerwester et al. ([4]) proposed the Latent Semantic Analysis (LSA) approach which performs Singular Value Decomposition of feature–document matrix to retrieve latent factors claimed to represent common underlying concepts. In 1999 Hofmann ([9]) presented the Probabilistic Latent Semantic Analysis (pLSA) approach which had a strong statistical foundation as opposed to the work by Deerwester et al. In his paper, he introduced a probabilistic generative model of documents treating them as a mixture of latent topics represented by distributions over words. Finally, in 2003 Blei et al. ([1]) presented yet another approach — the Latent Dirichlet Allocation (LDA) which models corpus as a collection of documents represented as random mixtures over latent topics where each topic is described by a distribution over words. The pLSA and LDA present the generative approach rather than a discriminative one therefore statistical inference relies on model fitting to data using maximum likelihood or Bayesian methods. The reported experimental results confirm high efficiency and usability of these methods.

The unsupervised learning approach has a substantial advantage over supervised one as there is no need for labelled data and the concepts are identified by co-occurrence analysis. However, such methods deal with latent topics that are represented by distributions of words ([1], [9]) or linear combinations of words ([4]) learned from data and there may be some problems with interpretation of these topics. On the other hand, these topics should be somehow mapped to the predefined classes (keywords in our case) used for classification. That implies that the technique of latent analysis may well be used for clustering of documents, for indexing or feature space dimensionality reduction but at least some additional work has to be done in order to use it for classification.

The other approach builds on so called supervised learning, i.e., learning over pre-labelled corpus. The algorithms perform by analysing positive and negative examples of classes or categories and building discriminative rules so that they classify learning data as correct as possible. There are a few state-of-the-art methods that gained high popularity and are widely used ([13]). One of the most simple ones is the naive Bayes approach which builds on the assumption of word independence over the text and makes use of Bayes rule to compute scores for classes. Instance based algorithms such like k–nearest neighbours ([15]) skip the phase of learning and make decisions by analysing expert decisions of documents closest to the one to be classified. Kernel based algorithms like Support Vector Machines (SVM) ([14], [10]) or linear regression based Linear Least Squares Fit (LLSF) ([17]) represent much more sophisticated yet computationally complex approaches. There is a great number of other methods that could also be mentioned including genetic algorithms, neural nets, decision trees but we limit ourselves to only those we are going to compare our proposed algorithms to.

The identification clouds are constructed by learning in a supervised manner over corpus containing labelled documents. In that way our approach is similar to the well-known machine learning based automatic text categorization algorithms [13] as opposed to the unsupervised learning based approaches. The mathematical interpretation of identification cloud is somewhere in between generalized mutual information ([12]) and 'salience' of salient word ([16]) and it presents a quantitive estimation of amount of discriminative information that certain chunk of text contains about a scientific term. That is the second meaning of phrase 'identification cloud' first being the heuristic definition given by M. Hazewinkel ([6], [7]). There is also a third one defined by the part 'Parametric Estimation' of this paper where non-informative (from the point of discriminative power) elements are virtually removed and the empirical estimates are fitted to some parametric model.

In this paper, we focus on the introduction, definition and mathematical formalization of the concept of identification cloud as well as on algorithms of a model statistical identification and procedures of classification. The results are partially published in [2].

2 Definition of an Identification Cloud

According to the publications [6], [7] where this concept is introduced and used, an identification cloud of a scientific term or phrase w is heuristically described as a set of words or short phrases with a functional defined on this set whose values are interpreted as weights. The weight of an element v of the identification cloud w is proportional to the chances that a text containing v also has the keyword w.

We first introduce a set of identification cloud owners — possible keywords. It is clear that assigning keywords to a certain text is up to a point equivalent to classifying that text. Let K denote some classification system of scientific publications which is identified with a set of all possible keywords in that system. A keyword is described as a scientific term or a group of terms which uniquely defines the class of a text fragment with that keyword assigned in the classification system K. For each $w \in K$ the class of all texts that are characterized by this keyword is also denoted by w.

Let V be a set of terms of a certain scientific field such that the results of classification of a text (in system K) depends on frequencies and positions of these terms in the text. We assume that the classification of a certain article a depends only on these words from the article that belong to the set V. The chronologically numerated vector of article's a elements (lower index for the word which is read earlier) $(a_1, \ldots, a_d)$, $d = d(a)$, where $a_i \in V$ and not necessarily $a_i \neq a_j$ is called the projection of the article a.

Remark 1. It is obvious that by extracting only scientific terms from the text we lose some valuable information — non–term words, punctuations, structural information, etc. A part of this information could be used to improve algorithms that are proposed further in this paper. A very simple generalization of the projection of an article could be to use a vector of pairs $((a_1, \lambda_1), \ldots, (a_d, \lambda_d))$ where $\lambda_i \in R$ is a value of some measure of distance between the terms a_{i-1} and a_i in the text. The distance may be taken into consideration when estimating the strength of relation between terms in the text.

Sometimes it is convenient to identify the projection of an article a with an infinite sequence $(a_1, a_2, \ldots)$, where $a_i = 0$ for all $i > d(a)$. Here $0 \in V$ denotes an additional zero term which does not exist in reality. Let A be a set of projections of all articles or other publications from a certain scientific field.

In what follows we omit the word "projection" and $a \in A$ is called just an article. From the point of view of classification an article is not necessarily a homogenous piece of text — in the general case, it consists of $q = q(a) \geq 1$ continuous homogenous parts which are classified as different in system K. Non-intersecting intervals of indices $I_j(a) \subset \{1, 2, \ldots, d(a)\} \overset{def}{=} N(a)$ and keywords $w_j(a) \in K, \quad j = 1, \ldots, q$ correspond to these parts. Here $\bigcup_{j=1}^{q} I_j(a) = N(a)$ and $w_j \neq w_{j-1}, j = \overline{2, q}$: if two adjacent parts of the text are attributed to the same class they must be joined into one.

Remark 2. The assumption that the text consists of several non-intersecting continuous chunks relevant to different keywords may seem a bit doubtful. However, at the moment we put no constraints on the length of these chunks while the assumption gives a justification and motivation for implementing quite simple iterative step-by-step text slicing procedures when classifying.

Now we get back to the heuristic definition of the identification cloud of a keyword $w \in K$. We describe the identification cloud in terms of probability distributions related to classification of a random text. Let N be the set of natural numbers. An article $a \in A$ and a set of indices $I \subset N$ are chosen randomly so that the part of an article $\{(a_\tau, \tau), \tau \in I\}$ is homogenous: $I \subset I_\nu(a), \nu \in \{1, \ldots, q\}$. This part is attributed to the class $\eta = w_\nu(a)$ in the system K. A common problem of classification is to determine the unknown keyword η using the observed vector $a_I = (a_\tau, \tau \in I)$ (e.g. classify the introduction of an article, using only the first page of the introduction). Since (a, I, η) is the result of a random experiment, the probability distribution in the set K is defined by

$$Q(w) = \mathbb{P}\{\eta = w\}, \quad w \in K \ . \tag{1}$$

In classification theory $Q(w)$ is called an a priori probability that the random text belongs to the class w. Let Y be a set of all possible values of a_I. In the set Y the following conditional probability distributions are defined:

$$P(y) = \mathbb{P}\{a_I = y \big| \ |I| = d(y)\} \ , \tag{2}$$

$$P(y|w) = \mathbb{P}\{a_I = y \big| \ |I| = d(y), \eta = w\}, \quad w \in K \ , \tag{3}$$

where $d(y) = dim\ y, |I| = card\ I.$

If η and $|I|$ are independent, after observing a_I, the a posteriori probability of the random event $\{\eta = w\}$ is determined by the equality

$$Q(w|a_I) = Q(w) \cdot \psi_w(a_I) \ , \tag{4}$$

where

$$\psi_w(y) = P(y|w)/P(y), y \in Y \ . \tag{5}$$

The functional ψ_w mathematically describes the concept of an identification cloud of the keyword w. It reflects how the probability for the random text to belong to the class w depends on the frequency of terms in the text as well as on their positions. It is easy to see that ψ_w also depends on distribution of the pair (a, I); therefore the most general definition of the identification cloud would be a family of functionals $\Psi_w = \{\psi_w(\cdot|H), H \subset N\}$, defined on the set Y, where $\psi_w(y|H)$ is defined analogously to $\psi_w(y)$ under the condition $I = H$. In what follows we assume the distribution of (a, I) fixed.

Using the distributions, described in (1) and (3), we can define a Bayes classifier which minimizes mean classification losses. Let $l(w, v)$ be the amount of loss when a text from the class v is assigned to the class w. As always, $l(\cdot) \geq 0$ and $l(w, w) = 0$. If the classification is performed using the observation a_I, the Bayes classification rule is defined by the equality

$$\widehat{\eta} = \arg\min_{w \in K} \sum_{v \in K} P(a_I|v)Q(v)l(w, v) \ . \tag{6}$$

In case the loss function is trivial, i.e.,

$$l(w,v) = \begin{cases} c, w \neq v \\ 0, w = v, \end{cases}$$

where $c > 0$, it follows from (6) that

$$\widehat{\eta} = \arg\max_{w \in K} P(a_I|w)Q(w) \ . \tag{7}$$

In equalities (6) and (7), $\psi_{(\cdot)}(a_I)$ can be substituted for $P(a_I|\cdot)$.

3 Discriminant Analysis

While solving applied text classification problems it is unlikely that the distributions P and Q, used in (6) and (7), are known. The plug-in method which substitutes statistical estimates calculated from the learning samples for unknown distributions may be used. In this section, we discuss possible ways of obtaining these estimates.

Suppose that we know keywords of n homogeneous texts and also have observations of some parts of these texts. For simplicity, we call these texts just articles and assume that their observed parts are continuous and their beginnings coincide with beginnings of the corresponding articles. Thus, the sample X is composed of n pairs, $X = (y(1), \eta(1)), \ldots, (y(n), \eta(n))$, where $\eta(i) \in K, y(i) \in Y$. Here $Y = \{y = (y_1, \ldots, y_d) : y_i \in V, d \in N\}$.

3.1 Non-parametric Estimation of Distributions

First we discuss non-parametric estimation of the functionals, defined in equalities (1) and (5). The empirical analogue of $Q(w)$ is determined by the equality

$$\widehat{Q}(w) = \frac{1}{n} \sum_{j=1}^{n} 1_{\{\eta(j)=w\}} \ . \tag{8}$$

It is much more difficult to estimate $\psi_w(y)$, where $y \in Y, w \in K$. The k-nearest neighbours method could be used. First we define a measure of dissimilarity of elements from Y: let, for all $y, z \in Y$, $\rho(y,z)$ be a non-negative functional whose values are called a pseudo-distance from element z to element y. For a fixed $y \in Y$, we choose k "nearest neighbours" from the sample. Let $J(y) \subset \{1, \ldots, n\}$ be a set of k indices of observations $y(1), \ldots, y(n)$, for which the pseudo-distance to y the smallest ones. The estimate of $\psi_w(y)$ is determined by the equality

$$\widehat{\psi_w}(y) = \frac{1}{\widehat{Q}(w) \cdot k} \sum_{j \in J(y)} 1_{\{\eta(j)=w\}} \ . \tag{9}$$

Here $0/0 = 1$. The variable $k = k(n)$ depends on the size of the sample and conditions $k \to \infty, k/n \to 0$, as $n \to \infty$ hold.

It is natural to define the pseudo-distance between the articles y and z as to take into consideration both the frequencies and positions of scientific terms in ones. A simple example of such a pseudo-distance is the following. For each $y = (y_1, \ldots, y_d) \in Y$ and $v \in V$, let

$$\alpha_y(v) = \frac{1}{d} \sum_{i=1}^{d} 1_{\{y_i = v\}} \ ,$$

$$\beta_y(v) = \frac{1}{\alpha_y(v) d^2} \sum_{i=1}^{d} i \cdot 1_{\{y_i = v\}} \ . \tag{10}$$

Here $\alpha_y(v)$ is the frequency of the term v and $\beta_y(v) \in [0, 1]$ is the standardised centre of gravity of v positions in the text.

Let

$$\rho(y, z) = \sum_{v \in V} [|\alpha_y(v) - \alpha_z(v)| + c|\beta_y(v) - \beta_z(v)|] \ , \tag{11}$$

where $c \geq 0$ is the weight of the functional β.

Any other measure of distributional similarity ([11]) could also be used instead of (11).

By substituting estimates (8) and (9) for $Q(w)$ and $\psi_w(\cdot)$ in (7) we get the well known k–nearest neighbours method of classification [15]. However, it is difficult or even impossible to preserve the explicitness and compactness of identification clouds unless dimension of y is equal to 1.

3.2 Restrictions to the Model

Non-parametric estimation of distributions is highly sensitive to the size of learning samples. If the samples are relatively small, the estimates will be unreliable when using the k–nearest neighbours or any other non-parametric method of estimation (e.g., kernel based approach). So now we discuss the parametric estimation of these distributions and first of all we introduce some definitions and notation.

Let the same assumptions as in section 2 (on the random character of (a, I, η)) hold and let the index $\tau \in I$ be a random variable.

The distribution on set V is defined by

$$P(v) = \mathbb{P}\{a_\tau = v\} \tag{12}$$

and the corresponding conditional distribution is given by

$$P(v|w) = \mathbb{P}\{a_\tau = v | \eta = w\}, \quad w \in K \ . \tag{13}$$

It is easy to see that (12) and (13) are simplified cases of (2) and (3) which are obtained in case $y \in Y$ is scalar.

Assumption 1 (conditional stationarity and independence). *Let for all $y \in Y$ and $w \in K$ the following equality hold*

$$P(y|w) = \prod_{i=1}^{d} P(y_i|w) \ , \tag{14}$$

where $d = d(y)$ as before is the dimension of the vector y. Now the definition of the identification cloud (5) can be changed to

$$\psi_w(v) = P(v|w)/P(v), v \in V, w \in K \ , \tag{15}$$

while the Bayes classification rule for classifying the observed a_I ((6), (7)) is determined now by the equalities

$$\widehat{\eta} = \arg\min_{w \in K} \sum_{v \in K} \left[Q(v) l(w,v) \prod_{\tau \in I} P(a_\tau|v) \right] \ , \tag{16}$$

$$\widehat{\eta} = \arg\max_{w \in K} \left[Q(w) \prod_{\tau \in I} P(a_\tau|w) \right] \ . \tag{17}$$

In (16) and (17), $\psi_{(\cdot)}(a_\tau)$ can be substituted for $P(a_\tau|\cdot)$.

Once more we arrive at the analogue of a well known algorithm of classification of plain text — a so called naive Bayes approach which is known to be not highly efficient but on the other hand very simple in implementation. It is obvious that the assumption of conditional independence of the terms (14) is quite optimistic and does not hold in reality.

Remark 3. Numerous papers have shown that complexification of this assumption usually does not yield significant improvements in results of classification. Even more, Domingos and Pazzani in [5] showed that under certain conditions the violation of the assumption of independence does not harm the quality of classification.

The definition of the identification cloud based on this assumption (15) ignores information that can be derived from the order of the terms in the text. Thus, we introduce a weaker assumption.

Assumption 2 (conditional stationarity and Markovian property). *Let for all $y \in Y$ and $w \in K$ the following equality hold*

$$P(y|w) = P(y_1|w) \prod_{i=1}^{d-1} \left[P(y_i, y_{i+1}|w)/P(y_i|w) \right] \ , \tag{18}$$

where $P(v,u|w) = \mathbb{P}\{a_\tau = v, a_{\tau+1} = u | \eta = w\}$.
So a_I is a Markov chain taking values from V.

In this case, the identification cloud is described by two functionals: $\psi_w(v)$ defined in (15) and

$$\psi_w(v,u) = P(v,u|w)/P(v,u), v,u \in V \ , \tag{19}$$

where $P(v,u) = \mathbb{P}\{a_\tau = v, a_{\tau+1} = u\}$.

Let $I = \{r, r+1, \ldots, m\}$, then the Bayes rule of classification is obtained by substituting the corresponding right-hand side of (18) for $P(a_I|v)$ in (6) and (7):

$$\widehat{\eta} = \arg\min_{w\in K} \sum_{v\in K} \left[Q(v)l(w,v)P(a_r|v) \prod_{i=r}^{m-1} [P(a_i,a_{i+1}|v)/P(a_i|v)] \right] \ , \tag{20}$$

$$\widehat{\eta} = \arg\max_{w\in K} \left[Q(w)P(a_r|w) \prod_{i=r}^{m-1} [P(a_i,a_{i+1}|w)/P(a_i|w)] \right] \ . \tag{21}$$

Here $\psi_{(\cdot)}(v)$ and $\psi_{(\cdot)}(v,u)$ can be substituted for $P(v|\cdot)$ and $P(v,u|\cdot)$:

$$\widehat{\eta} = \arg\max_{w\in K} \left[Q(w)\psi_w(a_r) \prod_{i=r}^{m-1} [\psi_w(a_i,a_{i+1})/\psi_w(a_i)] \right] \ . \tag{22}$$

We can analogously introduce more general assumptions with 3, 4 and more neighbouring text elements taken into account. Even more, we could use some kind of adaptive choosing (depending on distances between terms) of neighbouring text elements if using a richer representation of an article (see Remark 1).

3.3 Parametric Estimation of Distributions

Suppose we have the same earlier defined sample X. Let $d(j)$ denote a dimension of an observed vector $y(j)$. We discuss one of the simpliest ways of parametric estimation of the statistical distributions.

First we calculate empirical estimates of the probabilities $P(\cdot)$, $P(\cdot,\cdot)$, $P(\cdot|\cdot)$ and $P(\cdot,\cdot|\cdot)$:

$$\widetilde{P}(v) = \sum_{j=1}^{n} \sum_{k=1}^{d(j)} 1_{\{y_k(j)=v\}} / \sum_{j=1}^{n} d(j) \ , \tag{23}$$

$$\widetilde{P}(v,u) = \sum_{j=1}^{n} \sum_{k=1}^{d(j)-1} 1_{\{y_k(j)=v, y_{k+1}(j)=u\}} / \sum_{j=1}^{n} (d(j)-1) \ , \tag{24}$$

$$\widetilde{P}(v|w) = \sum_{j=1}^{n} \sum_{k=1}^{d(j)} 1_{\{y_k(j)=v, \eta(j)=w\}} / \sum_{j=1}^{n} d(j) 1_{\{\eta(j)=w\}} \ , \tag{25}$$

$$\widetilde{P}(v,u|w) = \sum_{j=1}^{n} \sum_{k=1}^{d(j)-1} 1_{\{y_k(j)=v, y_{k+1}(j)=u, \eta(j)=w\}} / \sum_{j=1}^{n} (d(j)-1) 1_{\{\eta(j)=w\}} \ . \tag{26}$$

We obtain empirical identification clouds $\widetilde{\psi}_w(v)$ and $\widetilde{\psi}_w(v,u)$ if we substitute the estimates just described for the corresponding probabilities in (15) and (19).

First of all we modify the values of $\widetilde{\psi}_w(y)$ that were obtained by using too little observations. Let $\nu(v)$ denote the number of observations of the term v in the sample X and $\mu \in N$ denote the minimal value of one required for calculating $\widetilde{\psi}_w(v)$. Let

$$V_w = \{v : \nu(v) \geq \mu, \nu_w(v) \neq 0\} \ , \tag{27}$$

$$\underline{\psi}_w = \min_{v \in V_w} \widetilde{\psi}_w(v) \ , \tag{28}$$

$$\overline{\psi}_w = \max_{v \in V_w} \widetilde{\psi}_w(v) \ . \tag{29}$$

The functional

$$\widetilde{\psi}^*_w(v) = (\widetilde{\psi}_w(v) \vee \underline{\psi}_w) \wedge \overline{\psi}_w \tag{30}$$

will be used instead of $\widetilde{\psi}_w(v)$ for the parametric estimation of $\psi_w(v)$. Here $\vee$ and $\wedge$ denote the maximum and minimum operators. Further in this paper by writing $\widetilde{\psi}_w(\cdot)$ we mean the modified one $\widetilde{\psi}^*_w(\cdot)$.

The functionals $\psi_w(\cdot)$ and $\psi_w(\cdot,\cdot)$ determine the arrangements of set V for every $w \in K$ and every pair (w,v), $v \in V$: $V(w) = (v_1,\ldots,v_h)$ and $V(w,v) = (u_1,\ldots,u_h)$, where $h = \text{card } V$ and the following conditions hold:

$$\psi_w(v_1) \geq \psi_w(v_2) \geq \ldots \geq \psi_w(v_h) \tag{31}$$

and

$$\psi_w(v,u_1) \geq \psi_w(v,u_2) \geq \ldots \geq \psi_w(v,u_h) \ . \tag{32}$$

We use these arrangements for the construction of parametric models of the functionals $\psi_w(\cdot)$ and $\psi_w(v,\cdot)$. Let the condition that, for all $w \in K$ the following equalities are valid, be fulfilled

$$\psi_w(v_k) = g_1(k,\theta), \quad \theta = \theta(w) \tag{33}$$

and

$$\psi_w(v,u_k) = g_2(k,\lambda), \quad \lambda = \lambda(w,v) \ . \tag{34}$$

Here $g_i(\cdot)$ are some chosen functions, θ and λ are unknown parameters (generally multidimensional) to be estimated.

We discuss the procedure of estimation only for $\psi_w(\cdot)$, because $\psi_w(v,\cdot)$ can be estimated analogously. First of all, we arrange set V the way it is described in (31) with estimate $\widetilde{\psi}_w$ substituted for ψ_w. As the value of h can be very high, while the size of samples is usually not so large, only a part of terms is used for the parametric estimation of the identification cloud of word w:

$$\widehat{\psi}_w(v_k) = 1, \text{ if } s < k < h - l \ . \tag{35}$$

The identification cloud includes terms $v_1,\ldots,v_s$ and $v_{h-l},\ldots,v_h$, — only those terms $v \in V$ for which the value of $\widetilde{\psi}_w(v)$ highly differs from 1. First of all, we discuss finding of the values of s and l. Consider the hypothesis

$$H_0 : \ \psi_w(v) = 1 \tag{36}$$

as opposed to the alternative

$$H_1 : \psi_w(v) > 1 \tag{37}$$

and let $\overline{\alpha}(v)$ denote a p–value. The value of s is determined by the equality

$$s = \max\{k : \ \overline{\alpha}(v_k) < \alpha\} \ , \tag{38}$$

where α is the chosen level of significance, for example, $\alpha = 0.1$. Analogously

$$l = \max\{k : \ \underline{\alpha}(v_{h-k}) < \alpha\} \ , \tag{39}$$

where $\underline{\alpha}(v)$ denotes a p–value for the alternative

$$H_1 : \psi_w(v) < 1 \ . \tag{40}$$

Assume that H_0 holds and the indicators $1_{\{a_\tau = v\}}$, $\quad \tau = 1, 2, \ldots, d$ are conditionally independent and identically distributed taking value 1 with probability $P(v) \overset{def}{=} p$, if the condition $\eta = w$ is fulfilled. Then the conditional distribution of empirical probability $\widetilde{P}(v|w)$ under the condition $\sum_{j=1}^{n} d(j) \cdot 1_{\{\eta(j)=w\}} = m$ is a binomial distribution and the critical level of significance $\overline{\alpha}(v)$ is determined by the equality

$$\overline{\alpha}(v) = \sum_{m_0 \le k \le m} \binom{k}{m} p^k (1-p)^{m-k} \ , \tag{41}$$

where $m_0 = \sum_{j=1}^{n} \sum_{k=1}^{d(j)} 1_{\{y(j)_k = v, \eta(j)=w\}}$. Analogously $\underline{\alpha}(v)$ is determined by the equality

$$\underline{\alpha}(v) = \sum_{0 \le k \le m_0} \binom{k}{m} p^k (1-p)^{m-k} \ . \tag{42}$$

The value of p is often unknown therefore it may be substituted by $\widetilde{P}(v)$ in equations (41) and (42). In order to estimate $\psi_w(v)$, we choose a certain class of parametric functions: let

$$\gamma(k, \theta) = \theta_0 + \theta_1 k^{-\theta_2}, \quad \theta = (\theta_0, \theta_1, \theta_2) \ . \tag{43}$$

Then

$$\log \psi_w(v_k) = \begin{cases} \gamma(k, \theta), \ 1 \le k \le s, \\ \gamma(k, \theta^*), \ h - l \le k \le h. \end{cases} \tag{44}$$

Estimates of the parameters θ_j and θ_j^*, $j = 0, 1, 2$ are found by the equalities

$$\widehat{\theta} = arg \min_{\theta} \sum_{k=1}^{s} \Big| \log \widetilde{\psi}_w(v_k) - \gamma(k, \theta) \Big|^{\beta} \ , \tag{45}$$

$$\widehat{\theta}^* = arg \min_{\theta} \sum_{k=h-l}^{h} \Big| \log \widetilde{\psi}_w(v_k) - \gamma(k, \theta) \Big|^{\beta} \ , \tag{46}$$

where $\beta = 2$ or $\beta = 1$.

In case $\beta = 2$, we have a common least squares (LS) method, for which procedures for calculating estimates (also weighted ones) are usually implemented in any larger statistical package. Unfortunately, LS estimates are very sensitive to the empirical distribution $\log \widetilde{\psi}_w(\cdot)$ declinations from the theoretical ones $\gamma(k, \theta)$, therefore a more robastic median method ($\beta = 1$) is worth considering. If an appropriate procedure is not implemented in the statistical package being used, then a simple iterative algorithm for estimating θ can be used:

$$\widehat{\theta}(j+1) = arg \ \min_{\theta} \sum_{k=1}^{\delta} \frac{(\log \widetilde{\psi}_w(v_k) - \gamma(k, \theta))^2}{|\log \widetilde{\psi}_w(v_k) - \gamma(k, \widehat{\theta}(j))|}, j = 0, 1, \ldots \ . \qquad (47)$$

Here $\widehat{\theta}(j)$ is the LS estimate. The estimate of θ^* is calculated analogously.

4 Discussion and Conclusions

In this paper, we introduce stochastic models for identification clouds which can be used in solving problem of keyphrase assignment to scientific publications. The models present various levels of restrictions on the arrangement of scientific terms in a text, including the most general one with no restrictions (equivalent to word n-gram model), as well as the models with structural relations of some kind among terms. The latter ones may not necessarily represent reality very well but they suit better for applications due to a relative simplicity of calculations. Such restrictions on relations include a naive assumption on the conditional independence of terms (unigram model or bag of words approach) and a more general assumption on the Markovian property of dependence (two-gram model). It must be mentioned that some issues have not been addressed in this paper and they are to be discussed thoroughly in the following papers. That includes iterative procedure of homogenous parts identification, questions related to quantitive estimation of efficiency, and also questions related to practical aspects of identification clouds.

Practical experiments on real data are needed to evaluate the performance of proposed algorithms. For such experiments to be performed, constructive procedures, practical recommendations and data arrays of a certain size are needed. Along with the analysis of effectiveness of the proposed methods in classifying, it is also natural to compare these methods to other popular state–of–the–art methods which are highly effective and represent quite different ideas — SVM, k–nearest neighbours, LLSF, etc. The unsupervised learning approach (LSA, pLSA, LDA) based algorithms must also be considered both as stand-alone methods as well as feature space dimensionality reduction methods.

Although some investigations have been completed, their results should not be treated as highly significant because they were conducted on the basis of really poor data bases. The new database kindly provided by IMS (Institute of Mathematical Statistics) and VTEX Ltd is being prepared to run tests on. It consists of approximately 14000 full text articles from the field of probability theory and mathematical statistics with keywords and MSC classifiers assigned to. The results of experiments are to be presented soon.

Acknowledgement

Researches are funded by Lithuanian State Science and Studies Foundation (G–177).

References

1. Blei, D., Ng, A., Jordan, M.: Latent Dirichlet Allocation. Journal of Machine Learning Research **3** (2003) 993–1022
2. Balys, V., Rudzkis, R.: Stochastic models for keyphrase assignment. Proceedings of the VII International Conference "Computer Data Analysis and Modelling" (2004)
3. Church, K. W., Hanks, P.: Word Association Norms, Mutual Information, and Lexicography. Computational linguistics **16** (1990) 22–29
4. Deerwester, S., Dumais, S. T., Landauer, T. K., Furnas, G.W., Harsham, R. A.: Indexing by Latent Semantic Analysis. Journal of the American Society of Information Science **41** (1990) 391–407
5. Domingos, P., Pazzani, M.: Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier. Proceedings of the 13th International Conference on Machine Learning (1996) 105–112
6. Hazewinkel, M.: Topologies and metrics on information spaces. CWI Quarterly **12** (1999) 93–110
7. Hazewinkel, M.: Dynamic stochastic models for indexes and thesauri, identification clouds, and information retrieval and storage. In: R.Baeza-Yates a.o. (ed), Recent advances in applied probability. KAP, (2004) 181–204
8. Hazewinkel, M, Rudzkis, R.: A probabilistic model for the growth of thesauri. Acta Applicandae Mathematicae **67** (2001) 237–252
9. Hofmann, T.: Probabilistic Latent Semantic Analysis. Proc. of Uncertainty in Artificial Intelligence, UAI'99 (1999)
10. Joachims, T.: Text Categorization with Support Vector Machines: Learning with Many Relevant Features. Proceedings of ECML-98, 10th European Conference on Machine Learning (1998) 137–142
11. Lee, L.: Measures of Distributional Similarity. ACL 99 (1999) 25–32
12. Magerman, D. M., Marcus, M. P.: Parsing a Natural Language Using Mutual Information Statistics. National Conference on Artificial Intelligence (1990) 984–989
13. Sebastiani, F.: Machine Learning in Automated Text Categorization. ACM Computing Surveys (2002)
14. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (1995)
15. Yang, Y.: Expert Network: Effective and Efficient Learning from Human Decisions in Text Categorization and Retrieval. Proceedings of SIGIR-94 (1994)
16. Yarowsky, D.: Word-Sense Disambiguation using Statistical Models of Roget's Categories Trained on Large Corpora. Proceedings of COLING-92 (1992) 454–460
17. Yang, Y., Chute, C. G.: A Linear Least Squares Fit Mapping Method for Information Retrieval from Natural Language Texts. Proceedings of COLING-92, the 15th International Conference on Computational Linguistics (1992)

Capturing the Content of Physics: Systems, Observables, and Experiments

Eberhard R. Hilf[1], Michael Kohlhase[2], and Heinrich Stamerjohanns[2]

[1] Institute for Science Networking, Oldenburg
hilf@isn-oldenburg.de
[2] Computer Science, International University Bremen
m.kohlhase@iu-bremen.de
h.stamerjohanns@iu-bremen.de

Abstract. We present a content markup language for physics realized by extending the OMDoc format by an infrastructure for the principal concepts of physics: *observables*, physical *systems*, and *experiments*. The formalization of the description of physics observables follows the structural essence of the operational theory of physics measurements. The representational infrastructure for systems and experiments allow to capture the distinctive practice of physics: natural laws are supported by evidence from experiments which are described, disseminated and reproduced by others.

1 Introduction

The distributivity of information and services over the Internet has changed all aspects of life, and science is not an exception. We anticipate that the systems currently investigated in the community will eventually change scientific practice and that they will have a strong societal impact, provided that they can inter-operate to cover the whole work-flow of scientific research, education and application.

To further this vision we need to develop, implement, and provide semantic-based and context-aware techniques for acquiring, organizing, processing, sharing and using knowledge in science.

Our starting point is the view of the *scientific method* as a spiral (see Fig. 1), where we have our focus on physics here. In this view, scientific research in physics moves in a spiral trajectory from original ideas to results and even applications. Ideas pass through the processes of observation of natural

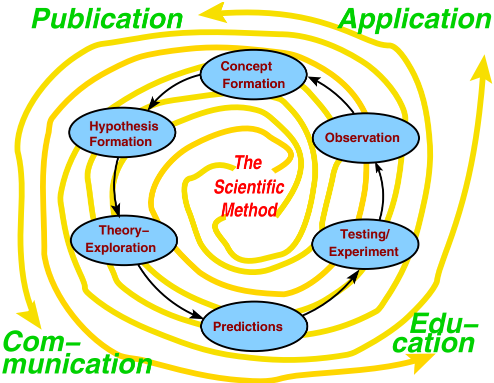


Fig. 1. The Scientific Method

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 165–178, 2006.

processes, then of concept formulation to describe these. These allow scientists to express initial theories about (quantitative laws of nature governing) them, which are then explored (what are the consequences of the model assumptions) leading to predictions about processes that can be verified or falsified (to a certain degree) experimentally. These experiments usually lead to new observations, starting the next round in the spiral until a quantitative (mathematically formulated) *theory* predicting exclusively correct results from experiments is formulated. Observables in physics have to be suitably found such that they can be physically measured, their algebraic counterparts being then candidates for building stones of a theory. The semantics of mathematics as such is more confined, searching for logically correct sets of rules.

At the moment, most of the steps in Fig. 1 are separately supported by software systems, e.g. literature searches in Google Scholar or WikiPedia, theory exploration in computer algebra systems like Mathematica, and experiments in simulation systems. But the systems are, by and large, not able to inter-operate since they use differing data formats, make differing model assumptions, and are bound to an implicitly given context that is only documented in publications about the systems. For instance, copy-and-paste from Google Scholar or WikiPedia to Mathematica or a simulation system is impossible because of this format problem. Moreover, where possible, copy and paste can be very dangerous, since computer algebra systems make differing assumptions on the Computercode-libraries, the simulation systems are based on[1].

We are set here to arrive at a content markup format for physics. Early concept discussions and visions [Hil05a, ERH05, HMS03, Hil05b] have not led to a realization in terms of an encoding, since the problem was attacked from the ground up. In this paper we will build the bridge from vision to a usable markup language by extending the OMDoc (Open Mathematical Documents) format [Koh06b] by an infrastructure for (physical) systems, observables and experiments and call this new module and the extended system PhysML (Physics Markup Language). Since we can now share all the infrastructure — in particular the theory and statement levels — with mathematics, the language design for PhysML becomes feasible.

2 Desiderata for a Physics Markup Language

The design of a semantic markup language for a learned field is more sophisticated than it might seem. The reason is that, in order to be useful it has to map the way research is organized. This leads to language designs centered around the *principal objects* of the respective research field. In chemistry, the Chemical

[1] A simple example, where the lack of explicit context led to a very expensive failure was the September 1999 loss of a $125 million Mars orbiter, which crashed on Mars. The cause was that NASA used for its specifications metric units, but the Lockheed Martin engineers misinterpreted the data assuming they were given using Imperial units of measurement.

Markup Language CML [CML05] was designed with the *name of a molecule* as principal object, to which properties and its chemical reactions are attached as properties[2].

In mathematics, the earliest discipline to have a dedicated markup language, we have MathML and OpenMath as markup formalisms that take mathematical objects as principal objects. The OMDoc format [Koh06b] extends them with content markup for *statements* (like definitions, axioms, theorems, and proofs) and *theories* (conceived as principal objects on a higher level).

In physics, since the times of Galilei ('*Experimental results are the highest authority*'), the young Einstein ('*A theory is to be accepted if it describes and predicts all possible experiments,— independent of the feelings and 'intuition' of the scientist*' [EB72]), and most important of P. Bridgeman [Bri27], who elucidated first the logical steps of physics learning, by analyzing the *operational* steps in doing research, it is now consensus [Mit70, Fal70b, Fal70a, Sak93] that research is accepted as physics if it does physical experiments with apparatuses which represent physical observables. This simple sounding requirement will be the entry point for us to design a specific Physics Markup Language, a construction which tries to mirror the way physicists operationally think.

2.1 Physics as a Science of Measurements

We start the same way as a physicist would enter a new field: by *operationally* following the consecutive steps:

DP1. Decide to work on a specific field, and gather *'pre-scientific' available knowledge.*[3]

DP2. Define an *observable*; In physics this has to be done by *constructing a physical device*[4], called an apparatus, with which measurements can be made giving real valued numbers[5] depending on the specific experimental setup.[6]

DP3. *Set an iterative operational construction rule* to refine stepwise the design of the apparatus such that (just as in 'proof by induction') by applying the rule iteratively, it will be accepted that successively more precise apparatuses can be built in principle.[7]

[2] We think that an alternative approach would have had more merits, to designate *chemical reactions* as the principal objects — the principal action, a chemist does.

[3] By this we concentrate on fields of interest, where we at least *assume* that by preparing physical experiments we may gain new knowledge.

[4] Historically, in the 1960s, there has been a long debate, whether in classical Mechanics, in contrast to all other fields, instead of *building* a physical device, already the *description* of how to build that device is sufficient [Mit70].

[5] In the modern *Theory of Measurements* observed numbers are to be mapped to *Eigenstates* of a *Hermitian Operator,* which is the mathematical image of the physical apparatus.

[6] This rule separates physics from other fields, such as mathematics.

[7] This absolutely essential rule assures that we stay with doable physics experiments. The condition that the rule has not to depend on the status of actual refinement assures that the limit (see next rule) to a virtual ideal mathematical counterpart of the observable will be secure and correct.

DP4. *Take this construction to its limit*, and define the virtual outcome of such an 'ideal device' as the *physical observable*, which then can directly be related (mapped) to the respective mathematics.[8] By this mapping to mathematical objects and their algebra, theoretical physics can be done with the aim to reproduce all previously conducted experiments and correctly predict any doable experiment in the field within the construction-dependent uncertainties of the apparatuses.

DP5. *Do a set of experiments, map to theory, check with the predictions* — if all are borne out we have a new natural law (Otherwise the set of assumptions and results are called '*model*').

DP6. *Distribute the results in a way so that the experiments and calculations can be repeated by others in the world.* Physics results (relations between observables) are independent of representations chosen for the mathematical objects needed, and independent of where and when (space and time chosen). They should be repeatable by other physicists at other laboratories in the world. Therefore the actual spreading of the information on the findings to other laboratories in the world *is* part of the operational procedure to gain physics insight.

To strengthen our intuition about the crucial step **DP3**, let us consider an example: Assume we want to measure the position in space of a physical object in classical mechanics. First we design a physically constructible 'detector' covering a finite space area $(x_i, \Delta x_i)$ which can distinguish whether the object is inside the detector area or not. Then we buy very many of these detectors and plaster (non-overlapping, touching detectors) the physical space. By checking all of them we learn in which detector $(x_i, \Delta x_i)$ the object is to the precision Δx_i. Repeating the experiment but with (may be a more expensive) detector set with *finitely* smaller detector space, say $\Delta x_{i+1} = \Delta x_i/2$ will give a better precision of the experiment. Repeating the application of the rule, which is obviously independent of the absolute value of precision gained in a certain step, would give us the ideal physics result. However we cannot experimentally do or pay for many refinement steps, and have to fear that the correctness of the experiment will break down if we physically go too far. That is why the limit process for the *physical observable* is done by virtually, not physically, going to the limit and mapping the result to a mathematical object as the mathematical representative of the observable. Each of the assumed algebraic properties has then to be tested by respective physical experiments. Thus only after experimental testing e. g. all commutative algebra

[8] A Hilbert operator for the ideal apparatus, a Hilbert State for its actual physical momentary realization, and *eigenvalues* for its measurement results. We confess that in practice, most scientists use real continuous variables for convenience, say for the position in space of a classical mechanical object, but with the strategy given here, we assure that we arrive at the correct quantum mechanics first and gain the classical mechanics statement by averaging over space from there using the standard Ehrenfest principle. The price for the convenience is high: we have to use Banach spaces instead of Hilbert space, any proof has to be done by iterating back to Hilbert space, use of distribution theory instead of functions, etc.

properties of the mathematical representative of the space position observable we can identify it with a vector in Euclidian space.

In short, we need the process given in the **DP** steps to ensure the choice of the related mathematical object and to get the best strategy for a semantic encoding of physics in a markup language. But how does this formal *operational definition* fit to the actual practical fixing of physics observables by international committees, e.g. the CGPM (Conference Generale des Poids and Measures), CODATA, IUPAP (International Union of Pure and Applied Physics), and SUNAMCO (Standards, Units and Nomenclature, Atomic Masses and Fundamental Constants)? This question is the domain of *Metrology*, an active research field of its own (see [Pen06] for a recent summary). The international metrologic commissions dwell on the next step of fixing observables once the operational definition has been set, focusing on

Precise measurement procedures extending the practical measurement of observables to very large and very small scales is achieved (for the length scale from cosmological to subatomic).

Determining physical constants by finding quantitative natural laws which connect real observations and thus can be reformulated to define a *physical constant* which is given by a physical process (such as the gravitational constant, the speed of light, etc.). Examples are: the scale for the time is set to be the *second* fixed as 9.192.631.770 periods of the hyperfine split light radiation of the atom *Cesium*. The *metre* is the length of the path traveled by light in vacuum during a time interval of 1/299.792.458 of a second, thus replacing the Ur-metre at Paris measured by a length ruler.

Hunting for higher precision which is especially necessary when long time unique series of measurements of given observables have to be trusted such as in geophysics, astrophysics.

2.2 Principal Objects for a Physics Markup Language

Given the above, we have to model the following principal concepts in a content/context markup language for physics.

Observables. As described, an observable is defined by the operational description of the defining apparatus, an iterative refinement rule, and properties such as dimension, scale, and attached algebraic object. The relations in which this observable occurs, etc. can be represented in OMDoc.

Experiments. Physics is distinct from other sciences by strictly sticking to reproducible experiments' outcomes as the source of knowledge. Reproducible means: to be able to tell others about the experiment so that they can reproduce it. This is in contrast to other sciences such as meteorology, history, or biology, which have records (data recorded over time) as principal source.

Apparatuses. Experimental measurements are done using apparatuses. An apparatus $\mathcal{A}$ is defined by a detailed description on how to build it, so that others may redo the experiment. Alternatively an apparatus can be fully described by *all* its simultaneously measurable observables[9]. A set of given values for all its properties defines a **State** $|a_i\rangle$, $i = 1, 2, 3, \ldots$ of $\mathcal{A}$. An experiment is conducted by bringing $\mathcal{A}$ into contact with another apparatus $\mathcal{B}$. The logical asymmetry of a typical experiment comes only with the mind of the observer, the experimentalist. She uses $\mathcal{B}$ to get information about the state of $\mathcal{A}$.

Again, an example is in order: Assume we have an apparatus *gas-filled bottle*, with a set of observables such as density of *gas*, *size*, *color*, and *material*, and one thermostatic observable, the *temperature*. We choose $\mathcal{B}$ to be a device which mostly has the same observables, such that if brought into thermal contact with $\mathcal{A}$ it does not affect the properties of $\mathcal{A}$ significantly but adapts its temperature to that of $\mathcal{A}$. We call $\mathcal{B}$ a *thermometer* and $\mathcal{A}$ a *system* in this case. Our interest here is on the value shown by the thermometer as a result of the value of the temperature of the apparatus. We neglect the (inevitable) changes of other observable's values, both of the system and the measurement device: such idealizations of experiments are common in physics.

3 Extending OMDoc for Physics

In this section we will extend the OMDoc format[10] by an infrastructure for (physical) systems, observables and experiments.

With the existing representational infrastructure in OMDoc we can already represent structured collections of interrelated concepts and statements about them via OMDoc `theory`[11] contexts. One of the central concepts in physics, the theory of measurable quantities can be set up in this way using OMDoc `symbols`.

We start with a simple example, the dimensions of the SI units.

[9] We note that in physics the list of properties of an apparatus is either finite or countably infinite (in contrast to e.g. biological systems). This assures a Hilbert space of states and real numbered values for the observables as the eigenvalues of the Hermitean Operator representing the Observable. This restriction to at most countable infinite property list is absolutely essential for physics. Only by that we get, after mapping to the formal mathematical context the correct observation that in all physics experiments measured numbers are real, as assured by the Hilbert state space and the Hermitean Operators therein.

[10] Due to space restrictions we cannot introduce the format here; we refer the reader to [Koh06b] for the language definition and examples.

[11] The nomenclature in mathematics, which gave rise to the element names in OMDoc and the naming conventions in physics clash here. In physics a set of assumptions about the physical world are called a “model” until they are generally accepted, only then are they called a “theory” (e.g. the Nuclear shell-model; however: quantum theory, general relativity theory).

Element	Attributes		Content
	Req.	Optional	
`observable`	`name`	`algebra, xref`	`metadata?, opdef, refinement, type?`
`refinement`		`xml:id, xref`	`metadata?, CMP*, FMP*`
`opdef`		`xml:id, xerf`	`metadata?, CMP*, FMP*`
`system`		`xml:id, xref`	`metadata?, realization?, observable*, preparation?, state?`
`realization`		`xml:id`	`metadata?, CMP*, FMP*`
`preparation`		`xml:id`	`metadata?, CMP*, FMP*`
`state`	`of`	`xml:id, xref`	`metadata?, value*`
`value`	`for`	`xml:id, xref`	⟪*mobj*⟫
`experiment`		`xml:id, xref`	`metadata?, CMP*, FMP*, measurement*`
`measurement`		`xml:id, xref`	`metadata?, state, state`
`evidence`	`for, type`	`xml:id,`	`metadata? CMP*, FMP*,interpretation`
`interpretation`		`xml:id`	`metadata?, CMP*, FMP*`
where `metadata`, `CMP`, `FMP and type are` OMDoc `elements described in [Koh06b] and where` ⟪*mobj*⟫ `is (OMOBJ |m:math |legacy)`			

Fig. 2. The Structure of PhysML Elements

Listing 1.3. Introducing Basic Concepts in a OMDoc Theory

```
<theory xml:id=”dimensions”>
  <symbol name=”mass”/><symbol name=”length”/><symbol name=”time”/>
  <symbol name=”charge”/><symbol name=”temperature”/>
  <symbol name=”volume”/>
  <definition for=”volume” type=”simple”>$\boxed{length^3}$</definition>
</theory>
```

We can introduce derived dimensions like the dimension for volume as defined concepts. Note that all of the `symbol` declarations make the concepts available for the use in OpenMath-encoded formulae via `OMS` elements and for the markup of technical terms via the OMDoc `term` element. Both identify a concept by its `name` and home theory (called a *content dictionary*; hence the attribute `cd`). Here as in the following, we use mathematical notation in boxes to abbreviate the OpenMath objects in the listings to save space.

We will use these dimensions as a type system for quantities, and introduce the units as constructors for the dimensions (note that we introduce the symbols with a `type`[12]).

Listing 1.4. A Theory of SI Units

```
<theory xml:id=”units”>
  <symbol name=”gram”><type system=”dimensions”>$\boxed{mass}$</type></symbol>
```

[12] In the example, we have not executed this, but it is possible to extend the type system to model ranges of numerical values in quantities in this type system: Instead of simply specifying that the unit K is of type `\temperature` we give K the complex type $\langle temperature, \mathbb{R}^* \rangle$ and adjust the dimension-types of the arithmetic operators, so that they check for range admissibility. This puts a considerably higher load on the type checking algorithm, but gives more control and quality assurance. As OMDoc encoding tolerates multiple type systems, we need not even choose one, but can accumulate the knowledge in the representations and use the one appropriate to the task at hand.

```
  <symbol name=”Kelvin”><type system=”dimensions”> temperature </type></symbol>
  <presentation for=”Kelvin”>K</presentation>
  <symbol name=”Celsius”><type system=”dimensions”> temperature </type></symbol>
  <presentation for=”Celsius”>°C</presentation>
  <definition for=”Celsius” type=”implicit”> ∀x ≠ 0.K = (x − 273.15)°C </definition>
</theory>
```

As usual, we can define the intended notation of a concept via `presentation` elements (see section 4) and we can introduce derived units via definitions. With this machinery, we can also state natural laws:

Listing 1.5. A Natural Law Expressed as an OMDoc `axiom`

```
<axiom xml:id=”force_mass_acceleration” type=”natural_law”>
  <CMP>Force is mass times acceleration.</CMP>
  <FMP> F = m · a </FMP>
</axiom>
```

Note that in OMDoc terminology we are dealing with an axiom, i.e. with an assertion that cannot be mathematically proven[13] but has to be assumed about the world. In physics a relation between observables has to be supported by sets of experiments, with no counter-evidence within the range of the variables of the involved observables.

3.1 Observables

Above we have determined the notion of an *observable* as a primary object of physics. As any observable — e.g. the temperature, or velocity — of a given physical system can be used in formulae describing the system, we need to extend the OMDoc format by a new statement-level language element that is definition-like. The `observable` element introduced by the PhysML module in OMDoc (see Figure 2 for an overview) has three relevant children[14] `opdef`, `refinement`, and `type`, to model the properties of observables we have identified in Section 2. The `opdef` and `refinement` elements contain *mathematical vernacular*, i.e. structured text interspersed with mathematical formulae. Mathematical vernacular is represented in OMDoc by a multilingual group of `CMP` (commented mathematical property) elements with mathematical text, and (possibly) a multi-system group of `FMP` elements with formalizations of the properties expressed in the `CMP`s. The `opdef` element is used for describing the *operational definition* of the observable, i.e. the defining process of measurement, whereas

[13] There may be physical evidence that supports it though.

[14] Here and in the following, we will not explicitly describe the `metadata` element, which is used in OMDoc to accommodate bibliographic and administrative metadata, specifying titles, digital rights, licensing, authorship, timestamping, etc. or the `xml:id` attribute which is used for identification. Details can be found in the OMDoc specification [Koh06b].

the refinement element is used to specify the rule of iterative refinement that takes the measurement process to its (idealized) limit.

The *dimension* of the observable is specified as a type element. Here we can directly use the type system for dimensions we have introduced in the last section. In our example in Listing 1.6 this is just the temperature.

The observable element carries a name attribute, which is used by OMDoc to introduce a symbol that can be referenced by an OMS element just like the symbol element. Furthermore, it carries an optional algebra attribute that contains a pointer to an OMDoc representation to the mathematical object introduced by this observable.

All of these elements also carry an optional xref attribute that allows to refer to an already existing representation of the same element via an URI reference; the effect is that the referred object is virtually copied in to the place of the referring one.

Listing 1.6. An Observable for the Temperature

```
<observable name="temperature">
  <opdef><CMP>Measure with a thermometer.</CMP></opdef>
  <refinement><CMP>Make the thermometer stepwise smaller.</CMP></refinement>
  <type system="dimensions"> temperature </type>
</observable>
```

3.2 Physical Systems and Their States

One of the basic building block of PhysML is the system element that is used to represent a physical system. The system is described via the mathematical vernacular in a realization element which is the first relevant child. As we have seen above, a physical system can be characterized by a (in practice very finite) set of observables, i.e. physical variables that can be measured independently. These are represented by a non-empty set[15] of observable children. Listing 1.7 shows a very simple system, which we will use as a concrete measuring apparatus later.

Listing 1.7. A Simple Physical System

```
<system xml:id="thermometer">
  <realization><CMP>A thin glass tube with mercury in it.</CMP></realization>
  <observable xref="#temperature"/>
</system>
```

[15] Enjoy the special cases: By use of an apparatus, which cannot measure anything (that is: has no observable) one cannot learn anything. The respective mathematical operator would be the identity. Less trivial is the case, where we prepare a system in state $|a\rangle$, then try a measurement 'is the system in state $|a\rangle$'? If it is already in that state, one does not learn anything new, and that is: no-one can decide whether the experiment took place or not. Example: heat a system and a thermal measuring device to 40 deg. Then measure the temperature of the system by the device: Your result 40 deg can by no means be distinguished from the suspicion you did not do the experiment.

In this setup, we have represented only the observable we are interested in: all other physical traits of the apparatus are irrelevant for our current purposes. If other physical properties also matter, then we can add other observables. However, we have to make sure that we fix the states of all of the observables that we do not want to measure. This can be done informally in mathematical vernacular in the optional `preparation` element, which may follow the `observable` elements, and more formally in a `state` element. A `state` element specifies a set of values for observables in the system it refers to (either its parent `system` or the system specified to in the optional `of` attribute) via a set of `value` children. A `value` element specifies the observable it refers to by referring to it's name in the required `for` attribute. Its content is a representation of a physical quantity as an OpenMath, content MathML, or OMDoc `legacy` element. In the example below, we have (somewhat arbitrarily) prepared a gas cylinder for an experiment by making it red.

Listing 1.8. A Physical System Prepared for an Experiment

```
<system xml:id=”gas_cylinder”>
  <realization><CMP>A gas−tight wooden cylinder</CMP></realization>
  <observable xref=”#pressure.obs”/>
  <observable xref=”#density.obs”/>
  <observable xref=”#color.obs”/>
  <preparation><CMP>We make the cylinder red!</CMP></preparation>
  <state><value for=”color”> red </value></value>
  </state>
</system>
```

3.3 Experiments

Physical experiments are represented by the `experiment` element in PhysML. The body of this element consists of two `system` elements followed by a set[16] of `measurement` elements. The first child represents the system which is measured, the second the measuring device. The `measurement` elements contain two `state` elements as described above which correlate the state of the system on which the measurement is performed with the state of the system of the measuring device. In the following example, we represent the result of measuring the temperature of a gas cylinder with varying density and pressure.

Listing 1.9. Experiment: measuring the temp. of a gas cylinder

```
<experiment xml:id=”ex_pressure_vs_temp”>
  <CMP>Measuring the pressure vs. temperature of a compressed gas cylinder</CMP>
  <system xref=”#gas_cylinder”/>
  <system xref=”#thermometer”/>
  <measurement xml:id=”m_213”>
    <state of=”#gas_cylinder”>
      <value for=”pressure”> 332.49586psi </value>
```

[16] We explicitly allow an empty set of measurements here in order to describe future, planned or failed experiments that have not yielded measurements (yet).

```
    <value for=”density”> 19g/l </value>
  </state>
  <state of=”#thermometer”><value for=”temperature”> 17.52K </value></state>
 </measurement>
</experiment>
```

Note that this only represents the raw data from an experiment. We can link experiments and natural laws, such as the one stated in Listing 1.5 via the `evidence` element. The main insight here is that as we cannot “prove” natural laws, but only observe them. We can only keep on experimenting in physics and collect evidence or counter-evidence for any relations between observables. The `evidence` element contains a non-empty set of `experiment`s followed by an `interpretation` element that allows to detail any interpretative steps, e.g. an account how the data was fitted to a curve, etc. Its `for` attribute specifies the relation it concerns, and the `type` attribute specifies whether the evidence supports it (value `for`) or falsifies it (value `against`).

In reality one is left with a residual ambiguity because physical experiments are conducted with real apparata, while the physics law gives a mathematical relation between the idealized quantities of the physical observables and apparata obtained as the (virtual) limit of the stepwise refinement iteration rule.

4 Reading, Writing and Arithmetic with PhysML Documents

Of course, the XML-based PhysML format presented here is not directly suited for humans to read and write. And indeed it is not intended to be; humans should use adaptive presentations for reading and invasive editors [KK04] for manipulating PhysML documents.

The OMDoc style sheets have been extended appropriately for the PhysML-specific elements. With these, PhysML documents can be converted to XHTML documents with MathML formulae that can be displayed in a browser or to PDF documents for printing via the LaTeX formatter.

PhysML inherits a well-established notation declaration language and presentation system from the OMDoc format: for new concepts that are introduced via `symbol` elements notation information can be specified via OMDoc `presentation` elements: In the presence of the following declaration,

```
<presentation for=”#Celsius”>
 <use format=”html|pmml”>&#x000B0;C</use>
 <use format=”TeX”>{}^{\circ}C</use>
</presentation>
```

The OpenMath object representing the temperature in of the thermometer in Listing 1.9 will indeed look like the visualization in the box.

To write PhysML documents, we have concentrated on the LaTeX workflow that is well-established in physics. Concretely, we have extended the semantic TeX system sTeX [Koh06a] by PhysML functionality.

Listing 1.10. Writing the PhysML for Listing 1.4 in sTeX

```
\begin{module}[id=units,uses=dimensions]
  \symdef[type=$\mass$]{gram}{g}
  \symdef[type=$\temperature$]{Kelvin}{K}
  \symdef[type=$\temperature$]{Celsius}{{}^{\circ}C}
  \begin{definition}[for=Celsius]
    $\allcdot{x>0}{x\Kelvin=(x-273.15)\Celsius}$
  \end{definition}
\end{module}
```

For more choice in invasive editors, we will extend the OMDoc wiki system [LK06] and the PowerPoint plugin for OMDoc [KK04] to PhysML.

The explicit, and standardized content representations for physical documents in PhysML will allow us to offer added-value services that cannot be offered on conventional representations. Examples are the dimension check comparing the physical dimensions, and the units used in an equation presented in a paper. If the dimensions on both sides of an equation do not match (say *kg* on one side, and *meter* on the other, the equation is physically openly wrong, if different units for the same dimensions were used on both sides this is called 'unlawful sloppiness' (say K on one side, $^{\circ}C$ on the other). Other checks will include the algebraic matching of both sides of an equation (say if *vector* on one side and *coaxial vector* on the other, this equation is bluntly incorrect). But more intelligent codes could also read the semantics delivered and offer mapping of algebraic results in different representation (say: integral instead of differential formulation, vector vs. vector-component or exterior form, etc.) thus directly assisting the reader to not having to read clumsy formulations of theoretical results from old times, but get it in the present used representations and notations.

5 Conclusion and Further Work

We have demonstrated that a Markup Language for the *content* of physics can be designed by extending the content and context markup format OMDoc with a representational infrastructure for the principal objects of physics: observables, systems, and experiments. The resulting language PhysML is able to catch the logical and operational structure specific to physics, differentiating this field from others. The extension presented in this paper is part of the ongoing enterprise to extend the OMDoc format to the **STEM** fields (Sciences, Technology, Engineering and Mathematics).

The next step is now to evaluate the language by marking up a larger body of knowledge in physics in PhysML. We have started work on the technically ubiquitous and basic field of thermostatics. This should give us a clear indication whether PhysML is adequate for all of physics, or pinpoint the necessary changes to the language design. An international collaboration on the further development of PhysML is looked for, including experts from theoretical and applied physics and related fields, in particular mathematics and chemistry.

New and powerful services can be implemented once the scientific content can be semantically encoded, retrieved, and reused digitally. In physics, these include the search for other experiments on the same observables, dimension and algebraic checking of mathematical equations, mapping to other mathematical representations of the same theoretical physical expression, etc.

Using the approach of analyzing the operational and logical practices of a scientific discipline field, and map this to field-specific modules extending the semantic markup language OMDoc will allow to spread semantic content markup to other scientific fields.

With authors to increasingly make use of markup languages, and retrieval engines following suit to offer intelligent search algorithms making use of the known markup languages, users will gain effective tools to increase the reachout of their scientific work, having the *content*, not just the text, of the work of others at their fingertips.

References

[Bri27] Percy Bridgeman. The logic of modern physics. *Comment: the book could not be traced but is well cited*, 1927. and all publications following this on operational foundation of the theory of measurement.

[CML05] Chemical Markup Language CML. Web page at `http://www.ch.cam.ac.uk/CUCL/staff/pm.html`, seen July 2005.

[EB72] Albert Einstein and Max Born. *Briefwechsel 1916 - 1955.* Rowohlt Verlag, Reinbek, 1972.

[ERH05] Julika Mimkes Eberhard R. Hilf. Zu einem verlustfreien Publizieren und Archivieren. Web page at `http://www.isn-oldenburg.de/~hilf/vortraege/mathdiss02/`, seen July 2005.

[Fal70a] Gottfried Falk. *Theoretische Physik I und Ia.* Heidelberger Taschenbücher. Springer Verlag, 1970.

[Fal70b] Gottfried Falk. *Theoretische Physik II Thermodynamik.* Heidelberger Taschenbücher. Springer Verlag, 1970.

[Hil05a] Eberhard R. Hilf. Kann man TeX beibringen, Physik zu verstehen? In *Der Aufbau eines deutschen TeXDoc-Centers an der SUB Göttingen, — Chancen von TeX-Dokumenten für eine dauerhafte Verwertung und Publikation.* Workshop des DFG-TeXDocC-Projektes, Die Deutsche Bibliothek Frankfurt, seen 2005-07-24 2005.

[Hil05b] Eberhard R. Hilf. Physml. In *Sesame 2005 Workshop Bremen.* MKM Mathematical Knowldege Management, seen July 2005. Web page at `http://www.mkm-ig.org/meetings/sesame05/program.html`, talk at Web page at `http://www.isn-oldenburg.de/ hilf/vortraege/sesame05/index.html`.

[HMS03] Eberhard R Hilf, Julika Mimkes, and Helmut Schottmüller. Die Zukunft des wisenschaftlichen Publizierens — Vom Publizieren zum Austausch von Informationen in der Wissenschaft. Web page at `http://www.isn-oldenburg.de/~hilf/vortraege/guestrow03/`, 2003.

[KK04] Andrea Kohlhase and Michael Kohlhase. CPoint: Dissolving the author's dilemma. In Andrea Asperti, Grzegorz Bancerek, and Andrej Trybulec, editors, *Mathematical Knowledge Management, MKM'04*, number 3119 in LNCS, pages 175–189. Springer Verlag, 2004.

[Koh06a] Michael Kohlhase. A LaTeX-based workflow for omdoc. [Koh06b]. to appear, manuscript at `http://www.mathweb.org/omdoc/pubs/omdoc1.2.pdf`.

[Koh06b] Michael Kohlhase. OMDoc *An open markup format for mathematical documents (Version 1.2).* LNAI. Springer Verlag, 2006. to appear, manuscript at `http://www.mathweb.org/omdoc/pubs/omdoc1.2.pdf`.

[LK06] Christoph Lange and Michael Kohlhase. A semantic wiki for mathematical knowledge management. In *Proceedings of the 1st Workshop: "SemWiki2006 - From Wiki to Semantics"*. 2006.

[Mit70] Peter Mittelstaedt. *Klassische Mechanik*, volume 500/500a of *Hochschultaschenbuecher*. Bibliographisches Institut, 1970.

[Pen06] L. R. Pendrill. Metrology: time for a new look at the physics of traceable measurement? *Europhysics News*, 37(1):25, 2006.

[Sak93] J. J. Sakurai. *Modern Quantum Mechanics.* Prentice Hall, 1993. ISBN 0201539.

Communities of Practice in MKM: An Extensional Model

Andrea Kohlhase[1] and Michael Kohlhase[2]

[1] Dept. of Computer Science and Mathematics, University Bremen
kohlhase@informatik.uni-bremen.de
[2] Computer Science, International University Bremen
m.kohlhase@iu-bremen.de

Abstract. We explore the social context of mathematical knowledge: Even though, the community of mathematicians may look homogeneous from the outside, it is actually structured into various sub-communities that differ in preferred notations, the choice of basic assumptions, or e.g. in the choice of motivating examples. We contend that we cannot manage mathematical knowledge for human recipients if we do not take these factors into account. As a basis for a future extension of MKM systems, we analyze the social context of information in terms of Communities of Practice (CoP; a concept from learning theory) and present a concrete extensional model for CoPs in mathematics.

> People don't learn to become [... mathematicians] by memorizing formulas; rather it's the implicit practices that matter most. Indeed, knowing only the explicit, mouthing the formulas, is exactly what gives an outsider away. Insiders know more. By coming to inhabit the relevant community, they get to know not just the "standard" answers, but the real questions, sensibilities, and aesthetics, and why they matter. *John Seely Brown in [Bro05]*

1 Introduction

In mathematics the production of knowledge is as dependent on social factors as in any other scientific discipline — even though this is not always realized from within, since mathematicians as a group can more easily agree or disagree on statements than other comparable groups. They use Georg Pólya's technique of "plausible deduction" that serves to differentiate between reasonable hypotheses and less reasonable ones (for a revealing ethnographic perspective on mathematics see [Hei00, 144]). Their objects of research have typically no important referent in day-to-day life, so that "truth" or "reason" is not a question of passion but of logic. At the core of mathematical identity is the concept of a proof as a process which ascertains reason [Hei00, 210]. Therefore, at first glance mathematicians build a huge, unified community and for outsiders, they seem to have the same practices all over the world. Indeed, these practices of formalization and proving can be easily distinguished from e.g. the one of experimentation by

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 179–193, 2006.

botanists. A closer look however reveals differences inside the field as well. For instance, the research objects, proof methods, proof evaluation methods, and the respective language about it differ quite dramatically even between subgroups as large as geometers and number theorists. We can discern communities of applied and pure mathematicians, which differ in the research motivation or analyticists and algebraicists, which use different mathematical tools and reasoning styles. Even on a very fine-grained level, there are communities that share or reject distinct practices, so that they can be rather small or short-lived: E.g. any research collaboration team might develop special notations (see 4.1) for their object of study and a pool of pertinent examples that are always ready at hand to test conjectures. Other examples of small, short-lived social units include the "students of a particular course".

In this paper we want to focus on the relevance of the social context for mathematical knowledge management (MKM). In particular, we want to apply the concept of "Community of Practice" (CoP) to the field of mathematics and draw consequences for the design of MKM technologies. We need to acknowledge that the context of mathematical knowledge is not only the intrinsic logical context that we model by MKM formats up to now, but also the social context. MKM can learn from this — after all, communities of mathematicians are quite efficient "mathematical knowledge management systems" and mathematicians insist that the core of mathematics lies as much in "doing" as in knowing (see e.g. [Bar02, 221]). In short — we contend that to understand mathematical knowledge management, we will have to understand its social aspects and hence to model CoPs in our systems. Otherwise we run e.g. the risk of inscribing our own CoPs into the systems, turning off users with differing practices.

2 Mathematical Communities of Practice

In 1991, Jean Lave and Etienne Wenger introduced the concept of "Communities of Practice" as the context in which learning takes place and knowledge is produced[1]. By now it is a well-established analysis tool in various fields and has experienced several extensions like "Communities of Innovation" (e.g. [Sch05, 43]) or "Communities of Knowledge" (e.g. [DP98, 66]).

2.1 Defining Communities of Practice

In order to adapt it for the field of MKM, we will now introduce the basic idea of **Communities of Practice (CoP)**, recall its definition, and argue for its relevance in MKM by interpreting Wenger's introduction of the term in [Wen99, 45][2]:

[1] They reacted with this situated learning approach against the dominant AI scheme of human intelligence as a complex computer program.

[2] In [KK05] we chose to introduce the concept via the learning object itself, mutating from raw data to information to a knowledge object within a community of practice (based on [BD00] and [PRR97]).

> *"Being alive as human beings means that we are constantly engaged in the pursuit of enterprises of all kinds, from ensuring our physical survival to seeking the most lofty pleasures. As we define these enterprises and engage in their pursuit together, we interact with each other and with the world and we tune our relations with each other and with the world accordingly, we learn. [...] Over time, this collective learning results in practices that reflect both the pursuit of our enterprises and the attendant social relations. These practices are thus the property of a kind of community created over time by the sustained pursuit of a shared enterprise. It makes sense, therefore, to call these kinds of communities communities of practice."*

2.2 Mathematical CoPs as Social Context for MKM

Unsurprisingly, mathematicians are as human as any other scientific species and as such we define and engage in common grounds, we interact and tune the relations among us and others. By doing this we produce and acquire mathematical knowledge. The interesting point that Wenger indicates here consists in his dictum "*we learn*". Even though MKM is concerned with knowledge, i.e. the product of the learning process, it seems to be interested in learning more in the form of e-learning systems as an application that be supported by MKM techniques than as a process that leads to mathematical knowledge and has to be understood for successful MKM. **Learning** is defined e.g. in Wikipedia as "*the process of acquiring knowledge, skills, attitudes, or values, through study, experience, or teaching, that causes a change of behavior that is persistent, measurable, and specified or allows an individual to formulate a new mental construct or revise a prior mental construct (conceptual knowledge such as attitudes or values)*". In this sense "**knowledge**" is a set of learned objects in an individual. Obviously, knowledge is very subjective: it depends on the learning subject. This begs the question how knowledge can become "objective" i.e. commonly accepted and understood, which is one of the central assumptions in the MKM community. In particular, how can human beings share a knowledge context? Not to drift off into philosophy, we just mention that the phenomenological concept of "intersubjectivity", i.e. the "mundane" *social* agreement on meaning, plays a decisive role in this process (for more information we suggest [Dou03, 99-126]).

Wenger continues that collective learning results in specific practices that differ with the respective community in which the knowledge was built up. Note that the term practice does not refer to a practical engagement in opposition to a theoretical engagement: "*Even when it produces theory, practice is practice*" [Wen99, 49]. Such communities of practice exist in mathematics as well (as mentioned above) even though — as they are rather informal — they don't tend to come into focus of the MKM community. We argue that in order to manage mathematical knowledge we have to pay attention to the context of production of knowledge objects. "Captured knowledge" in data bases was not only written by an author but also produced in a community of practice. Moreover, it shall be made use of by users who could be members of different communities of practice.

The supposed "common sense" or even "truth" of statements is worked out in the (social) context of CoPs[3].

2.3 What Constitutes Practice?

Etienne Wenger states that meaning must be negotiated between the interlocutors within a CoP and identifies two main, inter-operating processes in this: **participation** (action and connection) and **reification** (objectification and evaluation). In [Wen99, 63] he states "*in their complementarity, participation and reification can make up for their respective limitations*". Participation alone is too loose and confusing to establish coherent and consistent practice — therefore we e.g. take minutes in meetings. On the other hand, reified practices quickly become too inflexible to guide practice through everyday challenges hence we need to hire judges to interpret our laws.

2.4 Mathematical Practice

In MKM, we seem to have focused on just one of those processes: reification. We manage knowledge about mathematical objects via their reifications, in the same way as we have to use language objects to communicate about certain contents. At most the agreement on form can be viewed as a form of participation e.g. as valid substance equivalences. In [KK05] we present the Mathematical Knowledge Space (MKS), which we can now interpret under a CoP perspective. Wenger explicates that any "*community of practice produces abstractions, symbols, stories, terms, and concepts that reify something of that practice in a congealed form*" [Wen99, 59]. In the following we try to uncover the congealed practices.

Let us clarify this with an example: a mathematician at work. Typically, as a main part of her working life she will work by herself at her desk. She will review and evaluate what she already knows, what was said on a recent conference, or in a published journal. She might set up hypotheses and prove, postpone or drop them. Then she will elaborate on her results by writing a paper. Inbetween she will talk to colleagues, attend colloquia and conferences. Even though she works essentially by herself, she participates in the practice of mathematicians by basing her efforts on the results and values of her experience and doing it along the established (though informal) ways of the community.

2.5 Artifacts of Mathematical Practice as Living CoP Object

Clearly, authoring and studying documents are important mathematical practices. Hence, we can consider these documents as artifacts of mathematical practice. Documents result from the reification process of a practice. But similarly and at the same time documents are part of the participation process of this

[3] Remember the very controversial discussion about Hilbert's formalism versus Brouwer's intuitionism at the beginning of the 20^{th} century (see [Hei93] or [Bar02]).

practice, especially if they have been produced collaboratively. Documents are geared for the CoP-public and are a token of engagement with the specific CoP. Moreover, newcomers use these artifacts to become e.g. a mathematician, i.e. as John Seely Brown puts it in [Bro05] to "*inhabit the relevant community, they get to know not just the 'standard' answers, but the real questions, sensibilities, and aesthetics, and why they matter*". Thus, it is noticeable that reification *as well as* participation are inscribed into the working documents of the field. This property prompts us to center a CoP model around the collection of documents (in which the respective CoP practices result). In particular, we can consider it as a **living CoP object** in which many of the CoP essentials are contained and might be mined. Note that the judgmental characteristics of a CoP described by Brown blur the boundaries of such communities to be intersecting and rather fuzzy which we will make use of later on.

Now, we need to take an even closer look at the practices of a community for the modeling process.

2.6 Dynamics in Mathematical Practices

According to Wenger [Wen99, 4,49] the internal dynamics of a CoP are determined by the (interdependent) emergent characteristics of practice "meaning", "learning", "community", and "boundary", which we will now exemplify by describing them in terms of mathematical practice.

- *Practice as social negotiation of meaning:* Even though our mathematician works essentially by herself, she actively participates in the community by accepting the reifications of knowledge of her colleagues (in research documents) and sharing her own whenever possible. By this practice mathematical language can be understood among the members of the math community.
- *Practice as learning:* She works along the established (though informal) ways of the community, i.e. she reads and writes journal articles, conference papers, or listens to and speaks about colloquia talks. She takes into account the knowledge of the past in this CoP by basing her efforts on it and continuing it.
- *Practice as community:* She uses and establishes the coherence within the CoP by her engagement in the community, by working on a joint enterprise, and using a shared repertoire.
- *Practice as boundary:* She feels herself as a member of the community of mathematicians and will identify herself as such in a professional frame. But her practices will also set her apart from other communities with other common features.[4]

[4] Note that boundaries are subjective. For example, scientists in the humanities and social sciences discern CoPs of "techies" and "people people". From *that* perspective the current paper would certainly place the authors in the "techie" spot. In the MKM community we also have a division (maybe less pronounced) into corresponding CoPs. But from *this* standpoint the same paper positions the authors in the "human factor" CoP.

At the end we want to arrive at a model that is explicit enough to be implemented in MKM systems and that provides a basis to offer new services that support more of these aspects than the systems can currently offer (see section 4).

3 Modeling CoPs for MKM

In the attempt to model CoPs for Mathematical Knowledge Management we encounter the problem of a community's dynamics as the modeling process itself zeroes in on petrification. Therefore, the first question we have to deal with is: how can we get a handle on (mathematical) CoPs without inscribing the status quo — disregarding the fluid movements in a CoP? We might argue that a snapshot of the present context in the document itself is at least a first approximation. Unfortunately, this essentially yields a reduction of the idea of "living documents" to mere static (e.g. paper) documents.

3.1 Document Collections as CoP Models

Here, our approach is based on the idea that the identity of a scientific CoP is inscribed in the collection of documents this community produces. In contrast to the static characteristics of a single document, a **collection** of documents, i.e. a developing set that is structured by the respective CoP's participating members, maintains dynamic properties as the participation part of practice is involved, even embedded. From the perspective of a document in a collection, we speak e.g. of a "life cycle" of a document. Scientific developments and changing paradigms influence not only the content and form of new documents, but also the evaluation of older documents. We can even discern dynamics within a single document from this standpoint e.g. notational conventions that hold for some time but might change at any given time. Mathematical proofs serve as another example: the state of the art is not only determined by its content (search for new theorems), but also by its form (search for new proofs). If we consider a proof as combination of guarantee and explanation (see [Zin04, 4]), the explanation part is exactly the collection point of view.

In short, document collections seem to be a good starting point for modeling CoPs in MKM. Concretely, if we can assign CoP characteristics to its collection of documents, then the collection can be viewed as a *dynamic, living CoP object*, that can change without destroying the captured properties. Another advantage is that we do not need to make all properties about a collection explicit, building on the emergent effect of the composition of the documents.

3.2 Fuzzy Document Collections

Before we can start the search for CoP-characteristics in a document collection, we need to address another potential problem: CoPs do not have clear-cut *boundaries*, but a set of documents does. We take refuge in a standard idea from knowledge engineering that generalizes "sets" to "**fuzzy** sets", where set membership is generalized to a real-valued function (with values in $[0, 1]$) rather than a binary predicate. Following [Zad65] we interpret the fuzzy set membership function as an evaluation function indicating the degree of membership in a CoP-defining collection, or in other words of the value of the particular document to the respective community. It seems dubious that a one-dimensional value function will suffice to express the delineation of a CoP.

Moreover, the granularity of this multidimensional value function seems to be too coarse if we only evaluate entire documents. Sometimes only specific chapters in a book, or even a single definition carry value for a specific CoP. Therefore we will identify values on “knowledge entities” or “micro-content”, i.e. document fragments that make sense as a (possibly compound) unit of knowledge.

In our approach we make use of the fact that an individual person usually does not have a crisp delineation of which documents are relevant. We will take value judgments on documents to open up the boundary of a set. In particular, we use the concept of “value judgments” to define a CoP-determining document collection as a fuzzy set. Note that this set will be fuzzy in multiple dimensions, and we will use this multi-dimensionality to support various mathematical practices in section 4.

3.3 A Multi-dimensional Value Judgment Scheme

A natural first approach to capture such a value judgment scheme consists in using evaluation schemes that are already used regularly for peer review at conferences or journals. These should mean something for the respective community, otherwise they wouldn’t ask reviewers to give feedback on these points. We will attempt to model communities of practice for mathematical knowledge management by a set of value judgments on knowledge items in documents that a community endorses.

Concretely, we will model a community of practice as a semantically closed set of documents with judgment statements on its knowledge elements[5]. A **judgment** consists of one of the following dimensions d, a reference to a knowledge element o, and a numerical value v that expresses to which degree o has dimension d.

Relevance: Is the knowledge expressed in this knowledge element relevant to the CoP?

Soundness: Are the assertions conveyed here consistent with the assumptions made by the CoP? As a special case: are they internally consistent?

Presentation: Is the presentation (not all knowledge is expressed formally) likely to be understood by the CoP members?

Originality: Does the element contain new ideas?

Significance: Will the knowledge have an impact in the community?

[5] We will specialize this when we apply our model to a concrete knowledge representation format in section 4.

The dimension of “relevance” is arguably the most important (and generic) one. It determines the mathematical knowledge endorsed by the CoP. Note that we assume that the relevance judgment is semantically closed, i.e. if an element S semantically references an element T (e.g. if S belongs to a theory that imports T), then T must be relevant to the CoP as well[6]. Note that not all properties apply to all kinds of knowledge elements, e.g. for notation declarations only the relevance property makes sense, it is used to prioritize diverse possible notations (see section 4.1). Originality is a value judgment that takes the dynamics of the knowledge creation process into account. Research-oriented CoPs usually value original ideas higher than re-iterations. The significance judgment can be used as an interim estimate or preview of actual CoP-relevance for newly contributed material.

3.4 Capturing Value Judgments for Documents

The simplest way to determine actual values for the various dimensions of a judgment consists in authoring the necessary value judgments manually; this may be suitable for an explicitly administered CoP like the aforementioned community of the students of a given lecture. Here the teacher may chose to supply not only the course materials, but also the (intended) value judgments.[7]

Another way to obtain the necessary value judgments would be to mine existing resources, e.g. from the scientific refereeing process: We already have an established process for passing value judgments on mathematical documents there. For a CoP that is centered around a particular conference (e.g. the MKM community around the annual conference on Mathematical Knowledge Management, published in this volume), we could mine the referee reports to update the CoP representation for the newly published knowledge. We imagine that referee comments would be anonymized (possibly weighted by the referee’s standing in the CoP and competence). We want to point out that the judgmental dimensions above naturally coincide with those commonly used in conferences.

3.5 An Extensional Model for Mathematical CoPs

Note that the approach of identifying mathematical CoPs by the collection of documents and value judgments about it only gives us an *extensional model*, i.e. a basis for modeling certain behaviors of the community (or its members). In particular, the model does not say anything about the internal structure of the specific CoP, how membership is established or revoked, or about motivations

[6] Mathematical documents inscribe the (universal) assumption that a statement can only be accepted as reasonable, if it and all statements it depends on have been checked (i.e no “proof by authority”). Therefore we feel it is justified to inscribe semantic closure into the definition of mathematical CoPs. For other disciplines, this condition may have to be liberalized.

[7] The students may of course form a distinct CoP with their own value judgments that may or may not coincide with the teacher’s.

for membership. All of these concerns are important questions[8], but currently lie outside the focus of this enterprise. Only note that since the extensional model does not address the intensional level, it does not preclude anything, and therefore may very well form the basis of future modeling efforts.

It is legitimate to ask what benefits MKM might reap from this approach, and we will answer this question in the next section by specifying some important aspects of mathematical practice that cannot currently be supported by MKM systems, since we do not have a representation of CoPs. This also shows that in the model at hand mathematical CoPs are more than mere groups of people, but a cultural phenomenon that is determined by joint practices, which determine CoP membership as a secondary aspect. Thus the model conforms to Lave and Wenger's original theory of communities of practice [Wen99].

4 Added-Value Support for Mathematical Practices

In order to evaluate, whether the implementation of the CoP model adds value to the services MKM technology can offer, we will now consider support services for mathematical practices afforded by this model from the perspectives of "meaning", "learning", "community", and "boundary" introduced above.

As we are considering concrete practices that derive from the CoP model, we need to set it in a concrete MKM representation format. The OMDoc format [Koh06] is a good basis for this, since it already contains an infrastructure for some of the mathematical practices that we want: a structured notion of theory context and an infrastructure for notation definitions. Any other format that covers these would do just as well for our purposes.

All of the applications of the CoP model we present here are related to the presentation of mathematical knowledge to humans at different levels: from notation flexibility over intra-document and inter-document discourse optimization up to the social level . That is to be expected, since the communication practices of a CoP are essential to its existence.

4.1 Meaning: CoP-Specific Notation

One of the most immediate practices in mathematics is the creation of CoP-specific languages which are represented as mathematical formulae. Their notation is one of the most visible components of mathematical documents. OMDoc represents them as objects in the OpenMath format. For instance, the equation

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{1}$$

would be represented as the following string in the OpenMath XML encoding:

[8] Especially if we want to increase the participation of authors in MKM projects, see [KK04, KK05] for a discussion.

Listing 1.1. Content Markup for (1) in OpenMath

```
<OMOBJ>
  <OMA><OMS cd=”relation1” name=”eq”/>
    <OMA><OMS cd=”combinat1” name=”binomial/>
      <OMV name=”n”/><OMV name=”k”/>
    </OMA>
    <OMA><OMS cd=”arith1” name=”divide”/>
      <OMA><OMS cd=”combinat1” name=”factorial/><OMV name=”n”/></OMA>
      <OMA><OMS cd=”arith1” name=”times”/>
        <OMA><OMS cd=”combinat1” name=”factorial/><OMV name=”k”/></OMA>
        <OMA><OMS cd=”combinat1” name=”factorial/>
          <OMA><OMS cd=”arith1” name=”minus”/>
            <OMV name=”n”/><OMV name=”k”/>
          </OMA>
        </OMA>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ>
```

Unfortunately, the machine-oriented OpenMath syntax in (1.1) is painful for humans to read, therefore it is transformed to a more human-readable form, e.g. that in (1). Note that there are varied “standard notations” for binomial coefficients: $\binom{n}{k}$, ${}_nC^k$, C^n_k, and C^k_n, that are specific to the third notation various CoPs. The third one is used by French mathematicians, whereas the last one is the Russian one.

Listing 1.2. Two Notation Declarations for Binomial Coefficients

```
<presentation for=”binomial” role=”applied” fixity=”infix”>
  <use format=”TeX” lbrack=”\left(” rbrack=”\right)”>\atop</use>
</presentation>

<presentation for=”binomial” role=”applied” xml:lang=”fr”>
  <style format=”TeX”>
    <text>{\cal C}^{</text>
    <recurse select=”*[2]”/><text>}_{</text><recurse select=”*[3]”/>
    <text>}</text>
  </style>
</presentation>
```

In OMDoc, we can define notations by embedding one of the XML fragments in Listing 1.2 into the document that defines binomial coefficients. These declarations then inform the OMDoc presentation engine which notation to generate.

Note that with a notation declaration infrastructure (see [Nay02, MLUM05] for other proposals), we can *represent* notational diversity in the OMDoc format, but not *manage* it. The attempt at a management interface manifest in the `xml:lang` attribute on the `presentation` element is a first step that allows to adapt the notation to the primary language of the document. But what about a situation, where a French mathematician writes a paper for a Russian journal, or a German professor giving a class to French students based on a Russian textbook? In such situations, OMDoc proposes to fine-tune the notation via a `class` attribute on the OpenMath `OMS` element that selects the corresponding `presentation` element.

In the model, which identifies CoPs by the collection of documents, we can generate document presentations for arbitrary CoPs instead of having to rely on target languages. Instead of embedding the notations from Listing 1.2 into the defining documents, we could have a document containing one of them in a CoP-specific notation declaration document, which would be used to generate the relevant CoP-specific style sheets used in the production of the document presentations.[9]

Note that a similar account also holds for natural language names for mathematical concepts, which behave somewhat like notations. For instance, a "ring" can be an algebraic structure for algebraicists, a subset of $\mathbb{R}^2$ that is bounded by two concentric circles for geometers and something you wear on your finger for everybody else.

4.2 Learning: CoP-Specific Discourse Building

Content-oriented MKM formats like OMDoc separate content and presentation of mathematical knowledge allowing to generate latter from the former based on general (didactic) principles and user preferences. In the last section we have seen how an explicit representation of CoPs can help manage notation choice at a formula level. At the discourse level, we also have a distinction between content and presentation: In the current view of MKM (see for instance [Far04]), mathematical knowledge is organized into a richly structured network of theories, which define mathematical objects and concepts, prove properties about them, and store examples for them. This content is then organized into discourse-level presentations — documents that contain narrative text interleaved with the content elements, and that are tailored to a particular CoP.

Applications that automate the discourse-level presentation, like the ActiveMath system [MAF$^+$01] that generates individually geared math courses, have to make choices which parts of the material to present. Here the CoP data about the micro-content together with a user's CoP membership data can be used to make informed choices. For instance, we often have multiple examples to choose from that illustrate a given construct. These will usually come from theories that are different from the theory that contains the concept to be exemplified. Of course, the examples that come from documents that are highly relevant to the reader's CoP are especially familiar and therefore have a high didactic value. The learning effect will be especially great, if a concept can be explained with an example from another CoP the reader is a member of.

4.3 Community: CoP-Specific Reference Network

Just as a we can use the CoP information to optimize the choice of material presented to a reader for intra-document discourse optimization, we can optimize

[9] Thus a CoP-defining document collection is a principled resting place for notation declarations, which previously had a somewhat problematic status in OMDoc documents.

the presentation of the relations of the document to the others, i.e. for inter-document discourse optimization. Each document usually refers to several other documents. If a single document is cited a lot, its importance for the discipline is supposed to be very high. In particular, its content is central for the CoP - its presence, its past, and its future even though the value judgment about it might shift over time. The practice of referencing reveals this shift.

Using the CoP value judgments we can e.g. weigh bibliographic references by their CoP "relevance" value. Less relevant references might be made less visible or left out altogether. To improve this further, we need to distinguish linkages in documents. In OMDoc we distinguish between semantic links (usually given as theory inclusions or theory inheritance relations), bibliographic references, and ordinary hyper-references. All of these should react differently to the CoP value judgments. To obtain the reference network data for the CoP model, we might rely on algorithms for citation relevance e.g. used in CiteSeer [cit] or GoogleScholar [Goo05].

4.4 Boundary: CoP-Specific Value Judgments

For the management of more informal CoPs we imagine to adapt techniques from *social bookmarking*, an increasingly popular way to locate, classify, rank, and share Internet resources through the use of shared lists of user-created Internet bookmarks. The reported effect of the practice of social bookmarking consists in the engaging latitude of individual tagging and the socially informed inference drawing process based on mass data. At this point we don't want to judge the pros and contras of this approach, we just draw on the emergent dynamics of this new technology (see e.g. [Wei05] for a discussion). If such tags are visualized e.g. via tag clouds then a user obtains a feeling of belonging and participation, i.e. we can integrate the boundary effects for MKM technologies.

We envision that (in analogy to general social tagging systems like "delicio.us" for documents or "flickr" for photos, or scientific ones like "Connotea") CoP members store value judgments of knowledge items and make these lists publicly accessible. The value judgments necessary for representing a CoP can then simply be computed from the harvested value judgment lists of the members. This way the developmental cycles of a CoP are mirrored in the CoP model as well.

5 Conclusion, Related and Further Work

We have argued that the MKM community is still turning a blind eye towards the "*social life*" of knowledge and is thus missing out on valuable chances to offer personalized added-value services: mathematical knowledge does not live in a social vacuum, and neglecting that will rob MKM systems of the flexibility to scale up to larger and thus more diverse user communities. We cast the discussion in terms of "Communities of Practice", which we adapt to the context of mathematical knowledge, and propose a simple extensional model that is very well-integrated into MKM practice. We have shown the usefulness of this model by exhibiting knowledge management applications that feed on CoPs thus modeled and use the information to tailor the presentation process to the (CoP of the) user.

Unfortunately, we have not yet covered a very important practice in mathematics from the CoP angle of view: mathematical proofs. Of course, mathematical documents contain proofs, and as such they are encompassed by our model, but questions like proof style, or their peculiar status as a communicative acts between guarantee and explanation (see [Zin04, 4]) will merit further study.

Our work here is related to user modeling and community building work in other MKM systems. For instance, the OMDoc-based ActiveMath [MAF⁺01] system employs a user modeling component to infer the prior knowledge of a reader and employs it for the user-adaptive generation of narrative documents leading up to a chosen concept. Compared with our model, the user model is more detailed (it contains graded assumptions about "knowledge", "understanding" and "application"), and less comprehensive (for instance it does not contain information about notation preferences and "user models" for groups of users are not envisioned). But the ActiveMath user model could be viewed as the representation of a one-person CoP, e.g. by interpreting the "knowledge" property as "relevance". On the other hand, CoPs could be used to prime the user model in ActiveMath by assuming that a CoP-member knows, understands and can apply concepts proportional to their relevance. This would allow the user to simply identify her (pre-existing) CoP rather than giving confidence values for a large set of mathematical concepts.

Some MKM systems try to increase author involvement by providing community features (as e.g. this is what drives the runaway success of the WikiPedia). For instance the Connexions project [CNX06] has recently added "community pages" to their namesake system [CNX05] and plans to use a community-driven post-publication system called "lenses" for quality assurance. The former offer communities a forum for discussions and a way to identify a collection of relevant course modules. The lenses in Connexions are rating systems allow communities and institutions to endorse certain course modules. However, as the CNXML system can model less "practices" than OMDoc, its reach is limited to document selection.

The next step in our research enterprise will be to implement the CoP model presented in Section 3 and the added-value services sketched in Section 4 in an MKM system. We hope that by offering added-value services we will entice users to enter value judgments that can be used to represent and identify CoPs (extensionally). A feedback/rating system with an interface like the ones used in `amazon.com`, `Slashdot` or `ebay`, could turn an MKM system into a data collection tool for studying CoPs of mathematicians — although the necessary preselection of mathematicians who are willing to use MKM systems will probably introduce a strong bias.

To this effect we are currently extending the Connexions system to cope with OMDoc knowledge. We plan to build on its existing community features and extend them with this CoP model. As the system is used quite heavily for E-Learning in diverse communities ranging from music theory to electrical engineering, we hope to gain valuable insights into the inner workings of CoPs and their relation to value judgments. Another direction we would like to pursue is the extension of the OMDoc format that it can represent more mathematical practices, not just notation declarations, and the specification of technical terms.

All in all, we view the requirements coming from CoPs as essential guidelines for the further development of MKM formats. From an analysis concerning the relations between knowledge and practice, we can deduce the relevance of mathematical practices for MKM technologies as Osterlund and Carlile conclude that "*the relational core of a knowledge sharing theory easily falters. [...] We end up instead with a perspective that focuses on the storage and retrieval of explicit knowledge represented in information systems. Knowledge becomes an object shared within and across community boundaries* **without consequence for the community in which it originated**." [OC03, 18].

References

[Bar02] John D. Barrow. *Ein Himmel voller Zahlen. Auf den Spuren mathematischer Wahrheiten.* Rowohlt, 2002. Original publication 1992 "Pi in the Sky".

[BD00] John Seely Brown and Paul Duguid. *The Social Life of Information.* Harvard Business School Press, 2000.

[Bro05] John Seely Brown. Learning in the Digital Age. online, 2005. seen on 2006-02-07 at `http://www.johnseelybrown.com/learning\_in\_digital\_age-aspen.pdf`.

[cit] CiteSeer.ist scientific literature digital library. web page at `http://citeseer.ist.psu.edu`.
viewed March 2006.

[CNX05] Connexions. Project home page at `http://cnx.rice.edu/`, seen January 2005.

[CNX06] Connexions: Sharing knowledge and building communities. White paper at `http://cnx.org/aboutus/publications/ConnexionsWhitePaper.pdf`, February 2006.

[Dou03] Paul Dourish. *Where the Action Is: The Foundations of Embodied Interaction.* MIT Press, 2003.

[DP98] Thomas H. Davenport and Laurence Prusak. *Working Knowledge.* Harvard Business School Press, 2000 edition, 1998.

[Far04] William Farmer. Mkm: A new interdisciplinary field of research. *Bulletin of the ACM Special Interest Group on Symbolic and Automated Mathematics (SIGSAM)*, 38:47–52, 2004.

[Goo05] Google scholar. Web page at `http://scholar.google.com`, viewed January2005.

[Hei93] Bettina Heintz. *Die Herrschaft der Regel: Zur Grundlagengeschichte des Computers.* Campus Verlag, 1993.

[Hei00] Bettina Heintz. *Die Innenwelt der Mathematik. Zur Kultur und Praxis einer beweisenden Disziplin.* Springer-Verlag Wien, 2000.

[KK04] Andrea Kohlhase and Michael Kohlhase. CPoint: Dissolving the author's dilemma. In Andrea Asperti, Grzegorz Bancerek, and Andrej Trybulec, editors, *Mathematical Knowledge Management, MKM'04*, number 3119 in LNCS, pages 175–189. Springer Verlag, 2004.

[KK05] Andrea Kohlhase and Michael Kohlhase. An exploration in the space of mathematical knowledge. In Kohlhase [Koh05], pages 17–32.

[Koh05] Michael Kohlhase, editor. *Mathematical Knowledge Management, MKM'05*, number 3863 in LNAI. Springer Verlag, 2005.

[Koh06] Michael Kohlhase. OMDOC *An open markup format for mathematical documents (Version 1.2)*. LNAI. Springer Verlag, 2006. to appear, manuscript at `http://www.mathweb.org/omdoc/pubs/omdoc1.2.pdf`.

[MAF$^+$01] E. Melis, J. Buedenbender E. Andres, Adrian Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. The ACTIVEMATH learning environment. *Artificial Intelligence and Education*, 12(4), winter 2001 2001.

[MLUM05] Shahid Manzoor, Paul Libbrecht, Carsten Ullrich, and Erica Melis Ma. Authoring presentation for OPENMATH. In Kohlhase [Koh05], pages 33–48.

[Nay02] William Naylor. Mappings between presentation markup and semantic markup for variable-sized objects. In *Second International Conference on MathML and Technologies for Math on the Web*, Chicago, USA, 2002.

[OC03] Carsten Osterlund and Paul Carlile. How Practice Matters: A Relational View of Knowledge Sharing. In Marleen Huysmann, Etienne Wenger, and Volker Wulf, editors, *Communities and Technologies*. Kluwer Academic Publishers, 2003.

[PRR97] G. Probst, St. Raub, and Kai Romhardt. *Wissen managen*. Gabler Verlag, 4 (2003) edition, 1997.

[Sch05] Tim Schloen. Expertennetzwerke als Innovationsschmiede - das Konzept der Communities of Innovation. In Sylke Ernst, Jasmin Warwas, and Edit Kirsch-Auwärter, editors, *wissenstransform*, pages 40–53. LIT Verlag, 2005.

[Wei05] David Weinberger. Tagging and Why it Matters. online at `http://cyber.law.harvard.edu/home/uploads/507/07-WhyTaggingMatters.pdf` seen 2006-05-22, May 2005.

[Wen99] Etienne Wenger. *Communities of Practice: Learning, Meaning, and Identiy*. Cambrigdge University Press, 1999.

[Zad65] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[Zin04] Claus Zinn. *UnderstandingInformalMathematicalDiscourse*, volume 37 of *Arbeitsberichte des Instituts für Informatik*. Universitä Erlangen-Nürnberg, Institut für Informatik, September 2004.

From Notation to Semantics: There and Back Again

Luca Padovani[1] and Stefano Zacchiroli[2]

[1] Information Science and Technology Institute, University of Urbino
padovani@sti.uniurb.it
[2] Department of Computer Science, University of Bologna
zacchiro@cs.unibo.it

Abstract. Mathematical notation is a structured, open, and ambiguous language. In order to support mathematical notation in MKM applications one must necessarily take into account presentational as well as semantic aspects. The former are required to create a familiar, comfortable, and usable interface to interact with. The latter are necessary in order to process the information meaningfully.

In this paper we investigate a framework for dealing with mathematical notation in a meaningful, extensible way, and we show an effective instantiation of its architecture to the field of interactive theorem proving. The framework builds upon well-known concepts and widely-used technologies and it can be easily adopted by other MKM applications.

1 Introduction

Mathematical formulae can be encoded at different levels of human and machine understandability [1]. Formulae at the *notational level* are encoded on the basis of their rendering, in the same spirit of the MathML Presentation markup language [8]. Interaction with the user happens on formulae at this level: the user feeds the application with formulae in some notation, the system renders the formulae in some notation.

Formulae at the *semantic level* are those which the application has the deepest understanding of and on which it can better perform *computations*. In the fields of Computer Algebra Systems and theorem provers, examples of such computations include evaluation, simplification, automatic (dis-)proving, and typechecking. This level is intrinsically application-specific.

In between is an intermediate level, which we call *content level*, whose aim is to encode the structure and, to a limited extent, the semantics of mathematical formulae. MathML Content and OpenMath [14] are examples of markup languages that encode formulae at this level. The content level is the most effective vehicle of interoperability across MKM applications not sharing semantic foundations.

A framework that deals with meaningful mathematical notation has a naturally layered architecture where the same mathematical object is encoded in

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 194–207, 2006.

different ways according to the activities it is subjected to. The layers are connected with each other, and the encodings must be kept synchronized accordingly. In this sense we distinguish notation, which is a purely presentational tool, from *meaningful notation* that blends together both presentational and semantic aspects. From the perspective of the framework's designer, the fact that notation is extensible is a source of considerable additional complexity. It means that the layers cannot be fully described *a priori*, and that their connections must be updated dynamically as the system is enriched with new notation and new mathematical objects. It should be noted that a system supporting extensible notation in an exclusively presentational fashion is much simpler but also of limited use.

We consider the following features as characterizing such framework:

Extensibility: The framework must permit its users to define their own notation in an incremental way, using a basic set of primitive constructs along with all the notation has been defined earlier.

Remote control: Notation should provide handles for enabling indirect manipulation of the (possibly hidden) information encoded at the content and semantic levels.

Ambiguity: The framework must tolerate (and encourage) ambiguity, which is common practice in traditional mathematical artifacts.

Interoperability: The framework must not hinder communication with other software.

In this paper we show how a framework supporting extensible, meaningful notation can be designed, and we demonstrate the effectiveness of our approach

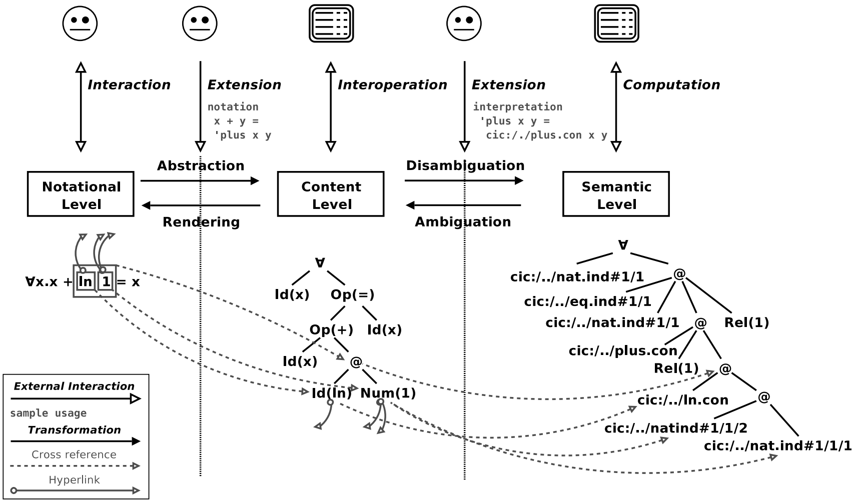


Fig. 1. Architecture of the notational framework

in the context of a theorem proving application. Figure 1 depicts the framework's architecture at work, where a first order logic formula is encoded at the three levels, the classical notational level on the left and the corresponding semantic encoding in the Calculus of (Co)Inductive Constructions [15] on the right. In the figure we use Helm [2] URIs as object references.

Transformations among the levels (horizontal solid arrows) are initiated by the need of interaction between the user and the application. When the need is to input a formula, *abstraction* brings the formula to the content level and *disambiguation* recovers a fully semantic encoding of the formula. When the need is to output a formula, *ambiguation* strips the formula of any application-specific semantic information and *rendering* creates a familiar representation. The transformations are driven by sets of bidirectional rules: a set of *notational equations* drives abstraction and rendering, while a set of *interpretations* drives disambiguation and ambiguation. The *extensibility* is pictorially represented as changes to these sets.

Cross references and hyperlinks account for *remote control*. Cross references relate corresponding pieces of information across the different encodings, so that the rendering engine can feature semantically driven forms of selection, cut and paste, and editing. Hyperlinks are one-to-many mappings from atomic objects to resources. Typically they link objects to their definitions.

Relevance to MKM and contribution. This paper complements [13] by investigating the technical issues related to the design of a user-extensible, interactive environment for the development and the management of mathematical knowledge in a semantically driven way. In particular, it proposes an architecture that has proven effective in mixing presentational as well as semantic aspect of the processed information. This is an improvement with respect to the currently available tools related to MKM which typically focus on one, but not both, of these equally important aspects.

Previous work [2] describing a similar architecture to that discussed in this paper did not address the issues related to extensible input support, and it only described informally how hyperlinks and cross references were propagated from the semantic to the presentation level. In this paper we describe these important features in a more abstract, but also more formal way, hoping to provide useful guidelines for future implementations.

Paper organization. The rest of the paper is organized as follows: in Section 2 we give a definition of notation by showing the relevant pieces of information that are affected by the notational equations. We do so by modelling levels with terms and transformations with functions on these terms. In Section 3 we complete the architecture by instantiating the semantic level in the particular case of a theorem proving application. Section 4 shows all the aspects of the framework at work on a concrete example. Section 5 discusses some related work and Section 6 concludes with some considerations about our implementation of the framework and some possible extensions.

2 Syntax and Semantics of Notation

In order to define precisely what notation is and how the information it conveys is processed during abstraction and rendering, we need a description of the languages encoding formulae at the notational and content levels.

Table 1. Syntax of presentation (E^{p}) and content (E^{c}) expressions

$E^{\mathsf{p}} ::=$			$E^{\mathsf{c}} ::=$		
	x	(identifier)		x	(identifier)
$\mid$	$l@H$	(literal)	$\mid$	$s@H$	(symbol)
$\mid$	$A\{E^{\mathsf{p}}\}$	(annotation)	$\mid$	$A\{E^{\mathsf{c}}\}$	(references)
$\mid$	$L[E^{\mathsf{p}}_1, \dots, E^{\mathsf{p}}_n]$	(layout)	$\mid$	$C[E^{\mathsf{c}}_1, \dots, E^{\mathsf{c}}_n]$	(constructor)
$\mid$	$B[E^{\mathsf{p}}_1 \cdots E^{\mathsf{p}}_n]$	(box)	$\mid$	α	(variable)
$\mid$	α	(variable)			

Table 1 shows the grammars for two streamlines languages of *presentation* and *content* expressions capturing the essence of notation. The two grammars are parametric in the following sets: a set of *layout schemata* L representing basic constructs of mathematical notation such as fractions, square roots, vectors, and so on; a set of *box schemata* B for annotating presentation expressions with line-breaking hints; a set of *identifiers* x, a set of *literals* l representing characters, numbers; a set of *symbols* s representing the basic elements in the ontology language of the content level (in MathML Content this set is predefined, in OpenMath it is completely unspecified, in either case it is open-ended and can be extended at will); a set of *constructors* C of the content level for building compound objects such as sets, lists, functions, relations. Literals and symbols are annotated with sets of *hyperlinks* H. We write l and s for $l@\emptyset$ and $s@\emptyset$ respectively. Both presentation and content expressions may be annotated with sets of cross references A. We omit the annotations p and c when it is clear that we are talking about presentation and content expressions, respectively.

A well-formed presentation pattern is a presentation expression E without identifiers, hyperlinks and cross references and such that any variable in E occurs exactly once. A presentation term is a presentation expression without variables. Content patterns and terms are defined similarly from content expressions.

A *notational equation* is a pair of well-formed patterns

$$P^{\mathsf{p}} \iff P^{\mathsf{c}}$$

that simultaneously defines (1) an *abstraction* from the notational level to the content level, and (2) a rendering from the content level to the notational level.

Example 1. The notational equation

$$\alpha \mathtt{=} \beta \iff \mathtt{apply}[\mathtt{eq}, \alpha, \beta]$$

defines a notation for the infix, binary operator = which is represented at the content level as an apply constructor whose first child is the eq symbol followed by the two operands in order. □

2.1 Abstraction

Abstraction is the process of instantiating the content term corresponding to a presentation term. Conceptually this is done in two steps: first, the presentation term is parsed according to the notation that is available where the term occurs and its parsing tree is determined. Then, the tree is navigated and a corresponding content tree is instantiated in a bottom-up fashion.

Let us discuss parsing first. Let $\mathcal{G}_0$ be the grammar that defines the *built-in notation* of the framework and let T be the grammar nonterminal symbol producing terms. The definition of new notation causes $\mathcal{G}_0$ to be extended incrementally as follows:

$$\begin{tikzcd}
\mathcal{G}_0 \arrow[r, "P_0^{\mathsf{p}} \iff P_0^{\mathsf{c}}"] & \mathcal{G}_1 \arrow[r, "P_1^{\mathsf{p}} \iff P_1^{\mathsf{c}}"] & \mathcal{G}_2 \arrow[r, "P_2^{\mathsf{p}} \iff P_2^{\mathsf{c}}"] & \cdots \arrow[r, "P_k^{\mathsf{p}} \iff P_k^{\mathsf{c}}"] & \mathcal{G}_k
\end{tikzcd}$$

where each grammar $\mathcal{G}_{i+1}$ results from $\mathcal{G}_i$ by the addition of a the production for T derived from $P_i^{\mathsf{p}} \iff P_i^{\mathsf{c}}$ and P_i^{c} is a content pattern parsed with $\mathcal{G}_i$ (this way notation can be defined incrementally on top of previously defined notation). In particular, the added production is $T \to \mathsf{exp}(P^{\mathsf{p}})$ where the function $\mathsf{exp}(P)$ converts a presentation pattern into a sequence of terminal and nonterminal grammar symbols as follows:

$$\begin{aligned}
\mathsf{exp}(l) &= l \\
\mathsf{exp}(\alpha) &= T \\
\mathsf{exp}(B[P_1 \cdots P_n]) &= \mathsf{exp}(P_1) \cdots \mathsf{exp}(P_n) \\
\mathsf{exp}(L[P_1, \dots, P_n]) &= L[\mathsf{exp}(P_1), \dots, \mathsf{exp}(P_n)]
\end{aligned}$$

Note that boxes are discarded in the expansion process as they play no role in the parsing phase and their content is juxtaposed.

A delicate technical problem related to grammars is ambiguity. An ambiguous grammar is one such that there may be multiple parse trees for the same term. In the most common cases ambiguity can be resolved by declaring precedence and associativity of productions. Thus, the language may provide additional constructs (see Section 4) so that the user can specify, for instance, that the symbol * has precedence over + and that * is left-associative. The remaining cases of ambiguity can be treated as errors (and the notation causing the ambiguity could be rejected or ignored), or they may be admitted provided that the implementation accommodates a form of content validation that can discriminate, among the various content terms that can be built starting from the very same presentation term, which ones are semantically meaningful. This validation phase usually entails a deeper understanding of content terms than it is available at the content level, thus some cooperation with the semantic level becomes fundamental for settling structural ambiguities.

Now we take care of the instantiation step. Given a presentation term t, the parser yields a parsing tree for t which we denote with $\hat{t}$. In particular, it determines a pattern P_i^{p} and a substitution σ that associates variables occurring in P_i^{p} with subterms of $\hat{t}$ such that $P_i^{\mathsf{p}}\sigma = \hat{t}$ (equality here is considered up to cross references and hyperlinks). We abbreviate this writing $t \in P_i^{\mathsf{p}} \leadsto \sigma$.

Example 2. assuming that the + operator has precedence over =, we have that

$$1 \texttt{ + } 2 \texttt{ = } 3 \in (\alpha \texttt{ = } \beta) \leadsto [\alpha \mapsto (1 \texttt{ + } 2), \beta \mapsto 3]$$

where we use parentheses to indicate a generic box schema. □

Abstraction is a function $\mathcal{A}(\cdot)$ defined as follows:

$$\mathcal{A}(t) = P_i^{\mathsf{c}}\sigma' \text{ where } t \in P_i^{\mathsf{p}} \leadsto \sigma \text{ and } \sigma'(\alpha) = \begin{cases} \mathcal{A}(\sigma(\alpha)) & \text{if } \alpha \in \mathsf{dom}(\sigma) \\ \text{undefined otherwise} \end{cases}$$

The function $\mathcal{A}(t)$ is well-defined as long as the terms in the image of σ are all proper subterms of $\hat{t}$.

2.2 Rendering

Rendering creates a presentation term from a content term. Like abstraction, we can think of this as a two-step transformation: during the first phase the structure of the content term t is inspected for finding those parts of the term matching the right-hand side of a notation $P_i^{\mathsf{p}} \iff P_i^{\mathsf{c}}$. Then, the left-hand side is instantiated accordingly. Unlike abstraction annotations and hyperlinks must be propagated to the presentation term and this is what makes rendering tricky. Table 2 shows the pattern matching of a content term t against a content pattern P as a system of inference rules. We use the notation

$$t \in P \leadsto_A \sigma, A', H$$

meaning that given an initial set of cross references A, the matching of the term t against a pattern P yields a substitution σ, a final set of cross references A', and a set H of hyperlinks harvested from the symbols in t.

We define the rendering function $\mathcal{R}(\cdot)$ as

$$\mathcal{R}(t) = A\{I_\sigma^H(P_i^{\mathsf{p}})\} \text{ where } t \in P_i^{\mathsf{c}} \leadsto_\emptyset \sigma, A, H$$

and the instantiation function $I_\sigma^H(\cdot)$ as

$$\begin{aligned} I_\sigma^H(l) &= l@H \\ I_\sigma^H(L[P_1,\dots,P_n]) &= L[I_\sigma^H(P_1),\dots,I_\sigma^H(P_n)] \\ I_\sigma^H(B[P_1,\dots,P_n]) &= B[I_\sigma^H(P_1),\dots,I_\sigma^H(P_n)] \\ I_\sigma^H(\alpha) &= \mathcal{R}(\sigma(\alpha)) \end{aligned}$$

In the process rendering a content term t annotations of subterms of t are preserved only in two occasions: either when they are found at the top level of t,

Table 2. Pattern matching of content terms

$$\textsc{(Symbol)}\quad s@H \in s \rightsquigarrow_A \varepsilon, A, H \qquad \textsc{(Variable)}\quad t \in \alpha \rightsquigarrow_A [\alpha \mapsto A\{t\}], \emptyset, \emptyset \qquad \textsc{(Annotation)}\quad \frac{t \in P \rightsquigarrow_{A \cup A'} \sigma, A'', H}{A\{t\} \in P \rightsquigarrow_{A'} \sigma, A'', H}$$

$$\textsc{(Constructor)}\quad \frac{(t_i \in P_i \rightsquigarrow_\emptyset \sigma_i, A'_i, H_i)^{i \in 1..n}}{C[t_1, \dots, t_n] \in C[P_1, \dots, P_n] \rightsquigarrow_A \sigma_1 \cdots \sigma_n, A, H_1 \cup \cdots \cup H_n}$$

in which case they become annotations for the resulting presentation term, or when they wrap proper subterms of t that have been bound by variables, in which case they will wrap the rendered subterms. As there is no obvious way of relating the other annotations, they are simply discarded (see the (Constructor) rule in Table 2). Hyperlinks, on the other hand, are handled pattern-wise. All the hyperlinks found in the part of a term matched by a content pattern are gathered together and sprinkled over the literals of the corresponding presentation pattern. That is to say, any visible part of the term is considered the concrete rendering of its symbols and should thus be linked to their definitions.

The definition of $\mathcal{R}(\cdot)$ omits two secondary details: (1) the function $\mathcal{R}(\cdot)$ must provide appropriate rendering for all the built-in notation defined in $\mathcal{G}_0$; (2) precedence and associativity of the productions are used to spot the subterms that must be protected by fences, in order to guarantee a presentation term that is consistent with the structure of the content term.

Example 3. Consider the notational equation

$$\alpha \neq \beta \iff \mathtt{apply}[\mathtt{not}, \alpha = \beta]$$

where we assume that the notation for the equality = has been given as in Example 1. The content term

$$t = i_1\{\mathtt{apply}[i_2\{\mathtt{not}@h_1\}, i_3\{\mathtt{apply}[i_4\{\mathtt{eq}@h_2\}, i_5\{\mathtt{1}@h_3\}, i_6\{\mathtt{2}@h_4\}]\}]\}$$

represents the inequality $1 \neq 2$ where the two constants 1 and 2 are located at h_3 and h_4 and are identified by i_5 and i_6 respectively. The whole term has reference i_1, the symbol `not` has reference i_2 and is located at h_1, while the symbol `eq` has reference i_4 and is located at h_2. The term t would be rendered as

$$i_1\{i_5\{\mathtt{1}@h_3\} \neq @\{h_1, h_2\}\ i_6\{\mathtt{2}@h_4\}\}$$

where we note that the reference of the whole term is preserved, whereas the references of the `not` and `eq` symbols have been lost (there is no natural rendered subterm corresponding to them). There are two links associated with the ≠ literal corresponding to the locations of the `not` and `eq` symbols. Finally, the symbols `1` and `2` have been rendered with all the information preserved (in the rendering we have omitted explicit box schemata for simplicity). □

3 Handling Ambiguity in MATITA

Since disambiguation and ambiguation (the transformations from content to semantics and back) inherit the quality of being application-specific from the semantic level, we cannot give a fully general recipe for handling them. We will therefore present their instantiation in the context of MATITA,[1] a document-centric proof assistant being developed at the University of Bologna. Nevertheless, as we will see shortly, we only require the semantic language to be compositional, as most structured languages are.

In MATITA the semantic language is the Calculus of (Co)Inductive Constructions [15] (CIC for short), a typed λ-calculus enriched with inductive data types. In this setting, an *interpretation* is a pair

$$s\ \alpha_1 \cdots \alpha_n \iff t[\alpha_1, \ldots, \alpha_n]$$

where s is a content symbol of arity $n \geq 0$ and $t[\alpha_1, \ldots, \alpha_n]$ is a CIC term with n holes labelled $\alpha_1, \ldots, \alpha_n$. The intention is to give *one of* the possible meanings for the symbol s when applied to n content terms $t_1, \ldots, t_n$, in terms of the CIC term t in which the hole α_i has been replaced by the meaning of t_i. The "one of" is to remark that there can be multiple interpretations for the same symbol s, not necessarily having the same arity.

3.1 Disambiguation

Of the two transformations dealing with the semantic level, disambiguation is the most challenging, since it has to resolve the ambiguity of content terms with respect to semantic terms.

When the semantic level is CIC, the ambiguity is induced by the one-to-many mapping of symbols to CIC term, which in turn is induced by overloading of operators and missing information at the notational level.[2] Consider the content level expression obtained after the abstraction of Example 2. Its ambiguity with respect to CIC derives from the overloading of + (two different plus do exists in the standard library of MATITA), and from the missing type argument of =, which is needed by the CIC encoding of Leibniz's equality.

Example 4. The following interpretations taken from the MATITA standard library show this ambiguity:

```
interpretation "natural plus" 'plus x y =
 (cic:/matita/nat/plus/plus.con x y).
interpretation "integer plus" 'plus x y =
 (cic:/matita/Z/plus/Zplus.con x y).
interpretation "Leibniz's equality" 'eq x y =
 (cic:/matita/logic/equality/eq.ind#xpointer(1/1) _ x y).
```

[1] http://matita.cs.unibo.it/

[2] Numbers and unbound identifiers also induce ambiguity. For the sake of brevity in this paper we treat them as 0-ary symbols for which the appropriate interpretations have been given.

The first two provide for overloading of `+`, the last uses an implicit CIC term (`_`) to represent the missing argument. □

Intuitively, *disambiguation* is a two phase process. In the first phase all possible CIC terms corresponding to a content term, according to the current set of interpretations, are built. In the second phase they are filtered by means of an oracle able to decide whether a term is valid or not. Such an oracle for CIC is the *refiner* described in [12]. The actual disambiguation algorithm implemented in MATITA exploits the type inference capabilities of the refiner and is far more efficient than the naive algorithm entailed by this intuition. The interested reader can find a detailed description of the disambiguation algorithm, as well as a discussion on its computational complexity, in [13].

3.2 Ambiguation

We call *ambiguation* the reverse transformation that creates a content term from a CIC term. It is simpler than disambiguation since the mappings from CIC to content are not ambiguous (they may be non-injective though). This step resembles rendering in many ways: ambiguation works by pattern matching on CIC terms, and it instantiates content terms according to the matching interpretations. As usual, the system provides a finite set of built-in mappings for transforming uninterpreted CIC terms to the corresponding content terms. Propagation of cross references and hyperlinks can be implemented in exactly the same way as described in Section 2.2, the URIs appearing in interpretations are the original sources of hyperlinks.

4 A Full-Scale Example

In this section we provide a complete example of notation in use in the MATITA proof assistant: existential quantification. The purpose of the example is twofold. On one hand it presents all together the aspects of notation from presentation to semantics. On the other hand, it allows us to glance at some features of the notational framework offered to the user for describing notational equations and interpretations that we had to omit from Sections 2 and 3 due to lack of space.

The existential quantifier is not built-in in CIC, but it is defined as an inductive data type in the `logic/connectives` module of the MATITA standard library. Its notation is given thus:

```
notation "hvbox(\exists ident i opt (: ty) break . p)"
 right associative with precedence 20
for @{ 'exists ${ default
  @{ \lambda ${ident i} : $ty. $p }
  @{ \lambda ${ident i} . $p }
 }}.
```

The presentation pattern is enclosed in double quotes. It consists of variables (`i`, `p`, and `ty`) that stand for arbitrary CIC sub-terms, and literals

(`\exists`, `:`, and `.`) assembled together in a box schema. The special keyword `break` indicates the breaking point and the box schema `hvbox` indicates a horizontal or vertical layout, according to the available space for the rendering. The `opt` indicates a meta-operator that surrounds an optional part in the presentation pattern. Given this presentation pattern, MATITA's input syntax is extended so that, for example, `\exists x:nat. x \le y` is a valid presentation term. Because of the `opt` meta-operator, the type annotation `:nat` can be omitted, the resulting term still being syntactically valid.

The line beginning with `right associative`... is self explicative: it specifies associativity and precedence of the notation, thus determining the binding strength of the existential quantifier during parsing and giving the renderer appropriate information for inserting parentheses when needed.

The content pattern begins right after the `for` keyword and extends to the end of the declaration. Parts of the pattern surrounded by `@{...}` denote verbatim content fragments, those surrounded by `${...}` denote meta-operators and meta-variables (for example `$ty`) referring to the meta-variables occurring in the presentation pattern. The content pattern of the example defines the application of the content symbol `exists` to a λ-abstraction. In this case there are two possibilities according to the presence or absence of the type annotation in the presentation term that matched the pattern. For this reason there is a corresponding meta-operator at the content level, named `default`, that has two branches which are chosen depending on the matching of the optional subexpression. In the example this is used to account for the optionality of type annotation on the quantified name, since its type can be inferred during disambiguation. Thus, if the type is given, the content term created after parsing has the form `'exists (\lambda x:nat.(x \le y))`. Otherwise, the resulting content term has the form `'exists (\lambda x.(x \le y))`.

Our notational language supports additional meta-operators for dealing with variable-size terms having a regular structure: the `list` operator, which can be used for describing sequences of presentation terms and literals, has a corresponding `fold` operator, which describes trees at the content level. Like for `opt` and `default`, `list` and `fold` together express a bi-directional relationship between the presentation and the content level.

In MATITA, the interpretation of the `exists` symbol is as follows:

```
interpretation "exists" 'exists \eta.x =
 (cic:/matita/logic/connectives/ex.ind#xpointer(1/1) _ x).
```

where the word `"exists"` enclosed in double quotes is an informal comment that can be used for keyword-based searching. In this interpretation the `exists` symbol has arity 1 and its only argument is required to be a function. This is expressed by the variable `x` being annotated with `\eta.`. Indeed, the content pattern shown previously regarding the `exists` symbol matches only when the symbol's argument is a function. Since this is not guaranteed at the CIC level, an η-abstraction is performed when necessary: if the CIC term matching `x` is not a λ-abstraction, a content term will be created for λ*fresh*.(`x` *fresh*) instead.

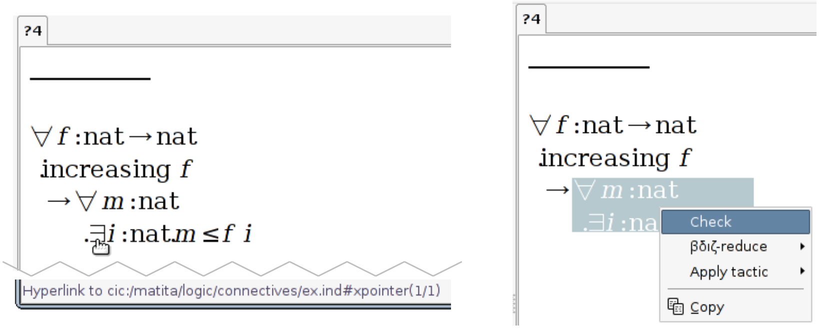
The following statement can now be used to start a new proof

```
theorem increasing_to_le:
 \forall f:nat \to nat. increasing f \to
 \forall m:nat. \exists i. m \le f i.
```

and the initial sequent of the proof is rendered as shown on the left of Figure 2. Notice that when entering a formula in the system the user is allowed to use a TEX-like concrete syntax, and the system can render the formula both on a textual terminal in the same concrete syntax, or in a graphical canvas like that of Figure 2 where the layout schemata of the formula have been properly encoded using MathML Presentation markup. This second view offers a more familiar rendering and it also enables point-and-click functionalities, like those for remote control. In this particular example the system figures that `i` must have type `nat`. In case more than one interpretation for the entered formula is feasible, the system lists them in a dialog box and asks the user to pick the desired one.

Remote control is exploited in MATITA in two ways: the first is *hypertextual browsing* of objects in the library. As can be seen on the left of Figure 2, the URI of the `"exists"` interpretation flowed through the levels reaching the literal ∃ as an hyperlink, which can be recognized at the bottom of the figure, in the status bar of the application. By clicking on the literal, the corresponding object from the library is shown to the user. If multiple hyperlinks are associated with the same symbol, a pop-up window appears and the user decides which one to follow. Incidentally, this gives the user some information about how a mathematical construct is encoded at the CIC level.

The second form of remote control, *semantic selection*, exploits cross references to constraint the selection on the presentation markup to CIC subterms. On the right of Figure 2 for instance, the GUI inhibits the selection of $\forall m : nat$ despite it corresponds to a proper subterm at the presentation level, since it has no corresponding subterm at the semantic level. Contextual semantic actions can then be safely offered to the user: the pop up menu in the figure shows



Hyperlink to cic:/matita/logic/connectives/ex.ind#xpointer(1/1)

Fig. 2. Remote control in action: hyperlinks on the left hand side, semantic selection and contextual actions on the right hand side

actions for type-checking, reducing, and using the term as a parameter for the next reasoning step. A classical copy operation to copy the subterm into the clipboard is also available.

5 Related Work

The layered architecture that we have proposed is similar in structure to that of previous projects in which notation played a major role. In [2,9] ambiguation and rendering are implemented by XSLT stylesheets [16] and they can only be extended by adding XSLT templates. Support for further notation is thus limited to the system designers. From the point of view of maintenance of the transformations, an improvement is the introduction of meta-stylesheet [7] that generate XSLT templates starting from a slightly higher-level specification. A somehow similar approach is proposed by Naylor and Watt [10] for supporting alternative notations. In any case, all the solutions mentioned are one-way only and cannot be inverted, both because XSLT is a very general transformation language, and also because the reverse path must reconstruct information that is not always available.

Our transformation language is not as general as XSLT but has been carefully designed so as to guarantee invertibility (the meta-operators mentioned in Section 4 are all invertible). Furthermore, it has a purely declarative style and is thus more appropriate for users who do not have any programming experience. The notational level consists of a finite set of layout schemata, basically those that are found in MathML Presentation [8] and TEX, box schemata for line-breaking inspired by previous work on pretty-printers [5], and a few meta-operators (like `opt` and `list`) inspired to the constructs of BNF grammars. The content level is an internal version of MathML Content and OpenMath [14], with the addition of meta-operators corresponding to those of the notational level.

The Coq proof assistant [4] provides a similar language for extending notation, with two main differences: it does not supply a content level and it does not deal directly with remote control. Our language represents a more open and interoperable solution, and the implementation shows that remote control can be achieved effectively even when notation is extensible, limiting built-in transformations to a bare minimum.

6 Conclusions and Future Work

In this paper we have characterized meaningful mathematical notation as a tool that necessarily mixes presentational as well as semantic aspects. We have identified a set of requirements that any MKM application supporting meaningful notation should fulfill and we have proposed an adequate architecture that builds upon the three well-known levels of formulae encoding: notation, content, and semantics. As an assessment of the generality of the architecture, we have given a formal dressing to the concept of notation which makes a minimum set of assumptions, and we have described an instantiation of its application-specific parts to the MATITA proof assistant.

The described architecture has been fully implemented in MATITA. The actual code has been written in OCaml[3] reusing components of the MATITA code base, most notably the code for disambiguation [13] and transformation from CIC to content and from content to MathML Presentation. Ambiguation and rendering have been implemented efficiently using a variant of the pattern matching algorithm in functional languages [3,6], which has been enriched with more expressive backtracking capabilities for dealing with meta-operators. Abstraction has been implemented using Camlp4, an extensible top-down parser with limited support for ambiguous grammars. This choice does not allow us to deal with structural ambiguity, that is with presentation terms admitting more than one corresponding content term. We plan to relax this constraint by implementing one of the several extensible parser generators that can be found in the literature (see [11] for an example).

Remarkably the proposed architecture does not deal with numbers in a practically useful way, since it assumes that there exists an infinite set of interpretations for them. In the Coq proof assistant, which basically shares the same semantic language used in MATITA, support for numbers is hard-coded in the application and thus it cannot be easily extended. We are currently investigating a declarative, finite interpretation scheme for numbers in MATITA, exploiting the regularity of their encoding in CIC, but it is still not clear whether this scheme is sufficiently general to make sense in different settings as well.

A major extension that we are considering is support for *local notation*, that is notation associated with content level binders that is in effect only in their scope. Local notation is a frequently asked feature in the formalization of algebraic theories, where quantification over notational symbols (as in "let $\odot$ be a binary operation over...") is a common mathematical practice. Since local notation requires an even tighter cooperation between the notational and the content levels, this could be a challenging test bench for verifying the scalability of our framework.

References

1. A. A. Adams. Digitisation, representation and formalisation: Digital libraries of mathematics. In J.H. Davenport A. Asperti, B. Buchberger, editor, *Proceedings of Mathematical Knowledge Management 2003*, volume LNCS, 2594, pages 1–16. Springer-Verlag, 2003.
2. Andrea Asperti, Ferruccio Guidi, Luca Padovani, Claudio Sacerdoti Coen, and Irene Schena. Mathematical knowledge management in HELM. *Annals of Mathematics and Artificial Intelligence*, 38(1-3):27–46, May 2003.
3. Lennart Augustsson. Compiling pattern matching. In Jean-Pierre Jouannaud, editor, *FPCA 1985: Functional Programming Languages and Computer Architecture, Proceedings*, volume 201 of *LNCS*, pages 368–381. Springer-Verlag, 1985.

[3] http://caml.inria.fr/

4. The Coq proof-assistant. `http://coq.inria.fr`.
5. M. de Jonge. A pretty-printer for every occasion. In Ian Ferguson, Jonathan Gray, and Louise Scott, editors, *Proceedings of the 2nd International Symposium on Constructing Software Engineering Tools (CoSET2000)*, pages 68–77. University of Wollongong, Australia, June 2000.
6. Fabrice Le Fessant and Luc Maranget. Optimizing pattern-matching. In *Proceedings of the 2001 International Conference on Functional Programming*. ACM Press, 2001.
7. Pietro Di Lena. Generazione automatica di stylesheet per notazione matematica. Master's thesis, University of Bologna, 2003.
8. Mathematical Markup Language (MathML) Version 2.0. W3C Recommendation 21 February 2001, `http://www.w3.org/TR/MathML2`, 2003.
9. The MoWGLI Proposal, HTML version. `http://mowgli.cs.unibo.it/html_no_frames/project.html`.
10. Bill Naylor and Stephen Watt. Meta-stylesheets for the conversion of mathematical documents into multiple forms. *Annals of Mathematics and Artificial Intelligence*, 38(1-3):3–25, May 2003.
11. Jan Rekers. *Parser Generation for Interactive Environments*. PhD thesis, University of Amsterdam, 1992.
12. Claudio Sacerdoti Coen. *Mathematical Knowledge Management and Interactive Theorem Proving*. PhD thesis, University of Bologna, 2004. Technical Report UBLCS 2004-5.
13. Claudio Sacerdoti Coen and Stefano Zacchiroli. Efficient ambiguous parsing of mathematical formulae. In Andrzej Trybulec Andrea Asperti, Grzegorz Bancerek, editor, *Proceedings of Mathematical Knowledge Management 2004*, volume 3119 of *LNCS*, pages 347–362. Springer-Verlag, 2004.
14. The OpenMath Society. The OpenMath Standard 2.0. `http://www.openmath.org/standard/om20/omstd20html-0.xml`, June 2004.
15. Benjamin Werner. *Une Théorie des Constructions Inductives*. PhD thesis, Université Paris VII, May 1994.
16. XSL Transformations (XSLT). Version 1.0. W3C Recommendation, 16 November 1999, `http://www.w3.org/TR/xslt`.

Managing Informal Mathematical Knowledge: Techniques from Informal Logic

Andrew Aberdein

Humanities and Communication,
Florida Institute of Technology,
150 West University Blvd,
Melbourne, Florida 32901-6975, U.S.A.
aberdein@fit.edu

Abstract. Much work in MKM depends on the application of formal logic to mathematics. However, much mathematical knowledge is informal. Luckily, formal logic only represents one tradition in logic, specifically the modeling of inference in terms of logical form. Many inferences cannot be captured in this manner. The study of such inferences is still within the domain of logic, and is sometimes called informal logic. This paper explores some of the benefits informal logic may have for the management of informal mathematical knowledge.

1 Informal Mathematical Knowledge

What sort of mathematical knowledge does mathematical knowledge management manage? A distinction between *knowledge that* and *knowledge how* is frequently deployed in epistemology. In mathematics this corresponds to the distinction between knowing mathematical propositions and knowing how to conduct mathematical proofs, that is being acquainted with mathematical practice. Each of these two sorts of knowledge may be related to a problem for MKM. In the first case, the problem may be expressed as 'How can a computer represent the truths of mathematics?'. In recent years this problem has been tackled with increasing success. In the second case, the problem may be expressed as 'How can a computer represent the proofs of mathematics?'. If this question is understood as 'How can a computer *perform* the proofs of mathematics?', then the progress in automated theorem proving provides a ready answer. However, this would be to misunderstand the original question, which did not ask how mathematics could be done by a machine, but how it is and has been done by mathematicians.

The traditions of formalization and automated theorem proving upon which much work in MKM has been based are heavily indebted to the methods of formal logic. However, formal logic is a poor guide to mathematical practice, as mathematicians seldom use it to write proofs. Although most mathematical proofs may in principle be formalized, the process is often arduous and can dramatically reduce intelligibility. For this reason such formalization is rarely attempted, and most mathematicians regard formal logic as of little relevance to their work. Moreover, a great deal of important mathematical communication

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 208–221, 2006.

does not even aspire to be formalizable in principle. Why? Because it contains substantial gaps, or even contradictions.

At first glance, it may seem that such mathematics would be no great loss. However, there are several areas of mathematical practice to which it is indispensable. The first of these is the history of mathematics. Little if any mathematical work conducted before the twentieth century meets modern standards of rigour. An historian engaged in a diachronic study of a mathematical theory needs to marshal a considerable body of mathematical inference, much of it unsound. Salvaging the sound parts and restating them with modern rigour may be good practice for textbook writers, but is not acceptable for historians.

Contemporary mathematics can also give rise to similar problems. The refereed journal article is not the only form of mathematical communication. Mathematicians with shared interests can often communicate complex ideas with considerable brevity and absence of formal rigour. Moreover, collaborators often profit from sharing their work in an unpolished, and perhaps mistaken, form. The famous English mathematicians G. H. Hardy and J. E. Littlewood apparently ran their long-standing and successful collaboration in accordance with a set of 'axioms'. The first of these 'said that when one wrote to the other (they often preferred to exchange thoughts in writing instead of orally), it was completely indifferent whether what they said was right or wrong. As Hardy put it, otherwise they could not write completely as they pleased, but would have to feel a certain responsibility thereby.' (from a lecture by Harald Bohr, cited in Littlewood, 1986, p. 10).

That sort of communication may be inadvertently contradictory, but some mathematicians have gone further, to find heuristic insight in 'the idea ... that a proof can be respectable without being flawless' (Lakatos, 1976, p. 139). Consider, for example, the following remarks of Fields medallist Vaughan Jones:

> I used to dislike intensely, *but have come to appreciate and even search for* ... the situation where one has two, watertight well-designed arguments that lead inexorably to opposite conclusions. ... Remember that research in mathematics involves a foray into the unknown. We may not know which of the two conclusions is correct or even have any feeling or guess. Proof at this point is our only arbiter. And it seems to have let us down. I have known myself to be in this situation for months on end. It induces obsessive and anti-social behaviour. Perhaps we have found an inconsistency in mathematics. But no, eventually a crack is found in one of the arguments and it begins to look more and more shaky. Eventually we kick ourselves for being so utterly stupid and life goes on. But it was no tool of logic that saved us. The search for a chink in the armour often involved many tricks including elaborate thought experiments and perhaps computer calculations. Much structural understanding is created, which is why I now so value this process. One's feeling of having obtained truth at the end is approaching the absolute. Though I should add that I have been forced to reverse the conclusion on occasions (Jones, 1998, pp. 208 f., emphasis added).

The situation which Jones describes, a deeper mathematical understanding derived from the analysis of apparently rebutted proofs, is central to the mathematical methodology espoused by the Hungarian philosopher of mathematics Imre Lakatos. His 'method of proofs and refutations' was inspired by the attempts of several generations of mathematicians to rescue the Descartes-Euler Conjecture from numerous apparent rebuttals. He summarizes the method as follows:

> **Rule 1.** If you have a conjecture, set out to prove it and to refute it. Inspect the proof carefully to prepare a list of non-trivial lemmas (proof-analysis); find counterexamples both to the conjecture (global counterexamples) and to the suspect lemmas (local counterexamples).
>
> **Rule 2.** If you have a global counterexample discard your conjecture, add to your proof-analysis a suitable lemma that will be refuted by the counterexample, and replace the discarded conjecture by an improved one that incorporates the lemma as a condition. Do not allow a refutation to be dismissed as a monster. Try to make all 'hidden lemmas' explicit.
>
> **Rule 3.** If you have a local counterexample, check to see whether it is also a global counterexample. If it is you can easily apply Rule 2. (Lakatos, 1976, p. 50).
>
> **Rule 4.** If you have a counterexample which is local but not global, try to improve your proof analysis by replacing the refuted lemma by an unfalsified one. (Lakatos, 1976, p. 58).
>
> **Rule 5.** If you have counterexamples of any type, try to find, by deductive guessing, a deeper theorem to which they are counterexamples no longer. (Lakatos, 1976, p. 76).

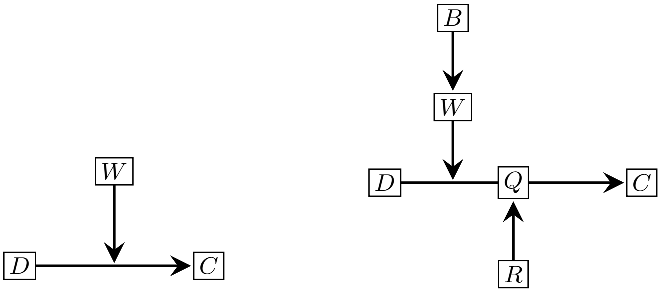
The mathematician described by Lakatos's method accrues mathematical beliefs, but many of them are tentative, changeable, and quite possibly wrong. But if Lakatos is right, and Jones's experience suggests that he is, this is nevertheless one of the most successful strategies for acquiring mathematical knowledge.

2 Informal Logic

2.1 What Is Informal Logic?

> To speak of informal logic is not to contradict oneself but to acknowledge what should be obvious: that the understanding of natural arguments requires substantive knowledge and insights not captured in the axiomatized rules of formal logic. (Govier, 1987, p. 204).

The distinction between formal and informal logic has been stated in a variety of different ways. Much of the confusion arises from the ambiguity of 'formal' (Johnson, 1996, p. 45). Informal logic does not exclude the pursuit of precise and



(a) Basic Layout

(b) Enhanced Layout

Fig. 1. Toulmin layouts

normative principles by which arguments may be analyzed and evaluated. Rather it concerns itself with arguments which cannot reliably be represented purely in terms of the logical form of the component propositions. Characteristically, these are arguments expressed in natural language. However, as the last section suggests, some arguments in mathematics may also have this quality.[1]

2.2 The Toulmin Layout

One of the most influential attempts to analyze the structure of arguments without appealing to the logical form of their propositions was developed in the 1950s by Stephen Toulmin. His 'layout' can represent deductive inference, but encompasses many other species of argument besides. The arguments it analyzes may vary considerably in strength: in particular, they may be defeasible—Toulmin was one of the first philosophers to use this term. Toulmin's layout continues to be an important focus for contemporary work in informal logic.

In its simplest form, shown in Fig. 1(a), the layout represents the derivation of a Claim (C), from Data (D), in accordance with a Warrant (W). This DWC pattern may appear to resemble a deductive inference rule, such as *modus ponens*, but it can be used to represent many other, looser inferential steps. The differences between these types of inference are made explicit by the additional elements of the full layout shown in Fig. 1(b). The warrant is justified by its dependence on Backing (B), possible exceptions or Rebuttals (R) are allowed for, and the resultant force of the argument is stated in the Qualifier (Q). Hence the full layout may be understood as 'Given that D, we can Q claim that C, since W (on account of B), unless R'. For example: 'Given that *Harry was born in Bermuda,* we can *presumably* claim that *he is British,* since *anyone born in Bermuda will generally be British (*on account of *various statutes ...),* unless *his parents were aliens, say*'.[2]

Toulmin's focus is on argumentation in natural language, not mathematics, although he is satisfied that the layout applies there as well. However, his only

[1] For a more substantial defence of the applicability of informal logic to mathematics, see Aberdein (2006).

[2] A frequently used example, derived from (Toulmin, 1958, p. 104).

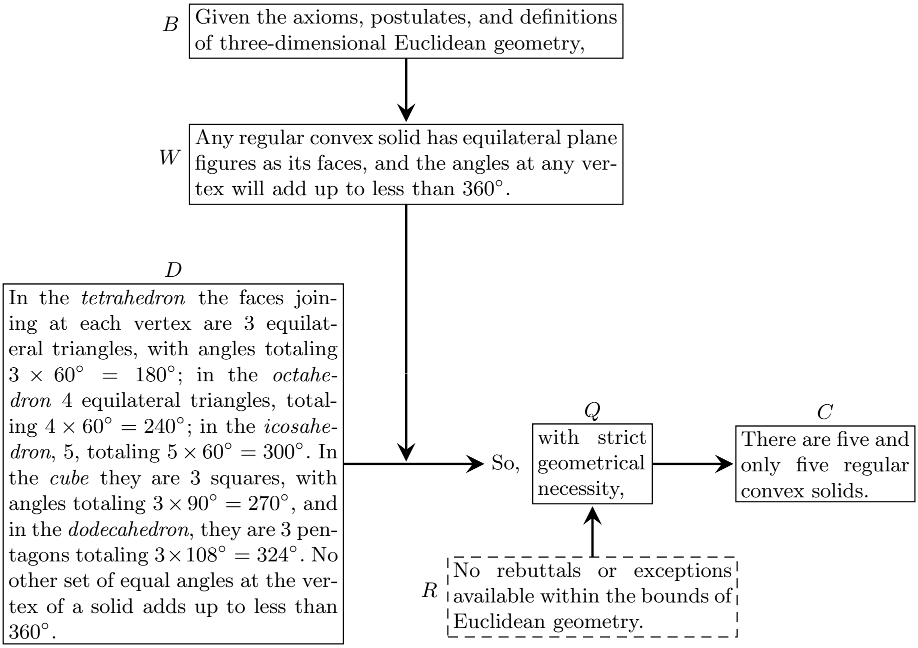

B
Given the axioms, postulates, and definitions of three-dimensional Euclidean geometry,
W
Any regular convex solid has equilateral plane figures as its faces, and the angles at any vertex will add up to less than 360°.
D
In the *tetrahedron* the faces joining at each vertex are 3 equilateral triangles, with angles totaling 3 × 60° = 180°; in the *octahedron* 4 equilateral triangles, totaling 4 × 60° = 240°; in the *icosahedron*, 5, totaling 5 × 60° = 300°. In the *cube* they are 3 squares, with angles totaling 3 × 90° = 270°, and in the *dodecahedron*, they are 3 pentagons totaling 3 × 108° = 324°. No other set of equal angles at the vertex of a solid adds up to less than 360°.
So,
Q
with strict geometrical necessity,
C
There are five and only five regular convex solids.
R
No rebuttals or exceptions available within the bounds of Euclidean geometry.

Fig. 2. Toulmin's analysis of Theaetetus's proof that the platonic solids are exactly five in number (Toulmin & al., 1979, Fig. 7.4, p. 89)

developed mathematical example is the proof from Euclid reproduced here as Fig. 2. One substantial shortcoming that this proof has as a model for how Toulmin's layout may be applied more generally is that it has only one step. Most mathematical proofs have many. Moreover, the number of steps a proof possesses is a function of the detail with which it is presented. In the next section I develop an extension of Toulmin's system which permits the presentation of multi-step proofs and also exhibits the relationship between presentations of differing depths of detail. Before doing so, I shall pause briefly to discuss an idiosyncrasy of Toulmin's approach: its highly visual nature.

2.3 The Pros and Cons of Visual Presentation

There are clear benefits to be gained from the visual presentation of argument, as a growing body of research acknowledges (see Kirschner & al., 2003, for example). A shared visual presentation can significantly facilitate communication of complex ideas, whether collaborative or pedagogic. The Toulmin layout, which is usually represented graphically, is a good example of this. However, visual presentation also has its drawbacks. Diagrams can be time-consuming to produce and frustrating to update, comment upon or integrate with other systems. As the diagrams grow in complexity these problems escalate.

At least two responses may be made to these difficulties. Firstly, these problems can largely be eliminated through the use of suitable software. Several programs are now available which, to a greater or lesser extent, automate the process of argument diagraming. Some of these programs (such as *Araucaria*: see Reed & Rowe, 2005) can be adapted to represent Toulmin layouts. Secondly, the convention of representing Toulmin layouts diagrammatically aids understanding, but is not essential. The basic layout may be thought of as a triple, $\langle D, W, C \rangle$. The greater generality of the enhanced layout may be brought to bear by associating a further triple $\langle B, Q, R \rangle$ with the warrant. We can therefore represent a full Toulmin layout as $\langle D, W\langle B, Q, R \rangle, C \rangle$, where each component represents a set of propositions, except for Q, a single term.

3 Combining Basic Layouts

3.1 Four Principles for Combining Layouts

Various proposals have been made to extend Toulmin's layout (for example, Newman & Marshall, 1992, pp. 15 f.). Many of these are concerned with applications of the layout to types of argument unlikely to occur in mathematical proof. I propose the following principles for combining mathematical layouts:

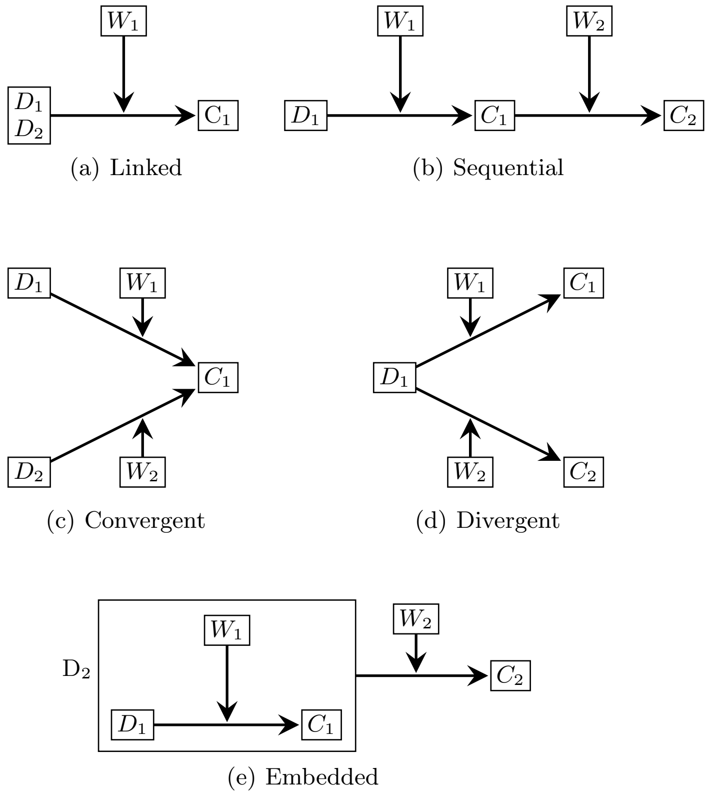
I Treat data and claim as the nodes in a graph or network.
II Allow nodes to contain multiple propositions.
III Any node may function as the data or claim of a new layout.
IV The whole network may be treated as data in a new layout.

Principle I: I shall not consider ways of extending the layout by adding links to components other than the data or claim. This often seems to duplicate existing features of the layout. For example, making the warrant of one layout the claim of a second duplicates the role of backing, albeit with more structure. Where such extensions are original, as in Newman & Marshall's (1992, p. 24) attachment of data and warrant to a rebuttal, they frequently seem more appropriate for other contexts, such as the legal argumentation for which this extension was devised.

Further to the argument in Sect. 2.3, we may observe that although graphs are frequently set out diagrammatically, this is not essential. Strictly speaking, a directed graph comprises a finite set of vertices, or nodes, and a finite multiset of edges, or ordered pairs of vertices. Each simple layout within a compound layout will correspond to such an ordered pair. Hence, in a network of layouts $\langle D_i, W_i\langle B_i, Q_i, R_i \rangle, C_i \rangle$, D_i and C_i label the vertices, and W_i and the other components label the edges.

Principle II: Toulmin already permits multiple data: consider the data in Fig. 2. This allows him to capture the linked argument structure represented as Fig.3(a). We shall go beyond Toulmin in permitting multiple propositions within a node to be distinguished as separate nodes (represented graphically by nested boxes). However, this is unnecessary unless the propositions are individually attached to other nodes.

In practice, we will still treat claims as singular. There may be some economy of exposition to be gained in permitting multiple claims to function as implicit



(a) Linked

(b) Sequential

(c) Convergent

(d) Divergent

(e) Embedded

Fig. 3. Five ways of combining layouts

disjunctions, in the manner of multiple conclusion sequent calculi. However, since nodes may function as both data and claim, this would risk unnecessary confusion.

Principle III: This principle allows us to construct sequential, convergent and divergent arguments, represented in Fig. 3(b)–(d). Sequential layouts are briefly considered in (Toulmin & al., 1979, p. 79); convergent and divergent arguments do not seem to be addressed. Strict adherence to Principle III prevents the occurrence of circuits, that is nodes which may be reached by two separate paths. (So the graphs we produce are actually trees.) Circuits are inferentially benign, provided they are acyclic—that is non-question-begging. However, they represent a redundancy of derivation which is seldom found in mathematical proofs. We could modify the principle to include acyclic circuits, but at the cost of complicating the folding rules introduced in Section 3.2.

Principle IV: This is our most radical departure from Toulmin, but we shall see that it is essential in order to capture some of the most pervasive forms of mathematical argument. Indeed, the reification of proofs as objects within larger proofs was a fundamental step in the development of mathematics. It is rather less common in ordinary discourse, making this situation dual to those we set aside in discussion of Principle I.

Principle IV does not allow embedding of a network into the *claim* of a new layout. The effect of such a step would be that the data and warrant of the new layout justified the derivation contained in the embedded network, placing them more naturally within the backing of (the individual steps of) the latter network. Principle III does permit data containing an embedded network to be treated as the claim of a new layout. However, in practice we shall avoid this move, restricting embedding to initial data.

Figure 4 exhibits how these principles may be used to reproduce some of the most common techniques in mathematical proof. Adjunction, Fig. 4(a), is just an instance of the linked layout, Fig. 3(a). Proof by contradiction, Fig. 4(b), combines divergent, Fig. 3(d), and embedded, Fig. 3(e), layouts. Representing each leg of the divergent layout separately would produce a logically equivalent presentation of this argument as a combination of linked and embedded layouts. This is the combination employed in both proof by cases, Fig. 4(c), and induction, Fig. 4(d).

3.2 Folding Compound Layouts into a Single Layout

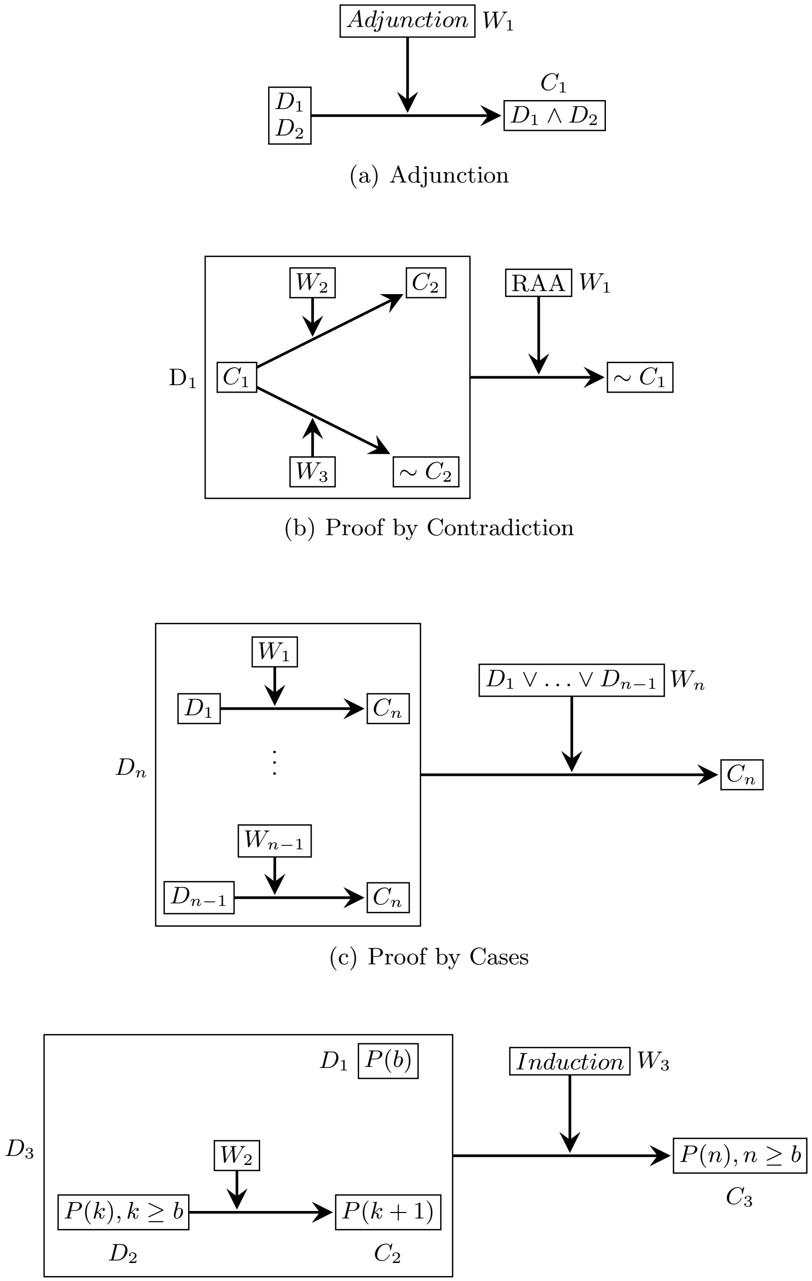
Is it possible to 'fold' the steps of a compound layout into a single layout? This process may obscure much of the detail of a proof, but should preserve soundness. That is, the folded proof should be no *less* sound than the unfolded proof: depending on its qualifiers, this may not itself be sound.

To see how this may be done, we will first observe that any network satisfying Principles I–IV must have at least one node of in-degree zero, which is not derived from anything, and at least one node of out-degree zero, from which nothing is derived. We shall call the former nodes *initial*, the latter *final*, and all other nodes *intermediate*. The folded layout should exhibit the dependency of the final nodes on the initial nodes; the intermediate nodes may be ignored. To preserve soundness, the warrant of the folded layout must be sufficient to justify each step of the unfolded proof. For networks following Principles I–III only, that is without embedding, a folded layout which meets these requirements may be defined as follows:

$$\left\langle \bigcup_{\mathrm{in}(D_i)=0} D_i,\ \bigwedge_i W_i,\ \bigwedge_{\mathrm{out}(C_i)=0} C_i \right\rangle \tag{1}$$

The warrant, W, of the folded layout is thus defined as the conjunction of every warrant, W_i, in the unfolded layout. This guarantees the inferential resources necessary to carry out the proof, although in practice a more concise warrant may suffice. The minimal requirement is that $W \Rightarrow W_i$ for all W_i, where '$\Rightarrow$' represents an appropriate account of derivation, which at this point we are assuming is used indifferently throughout the proof.

To deal with Principle IV, start with the most deeply embedded network(s) and reduce each to a simple layout, as in (1). There may be multiple layouts



(a) Adjunction

(b) Proof by Contradiction

(c) Proof by Cases

(d) (Weak) Induction

Fig. 4. Some common proof methods

embedded in the same box, as in Fig. 4(c): see the next section for a worked example. Then replace each embedded layout with a new data node $D_j^{\#}$ defined as $D_j \Rightarrow C_j$, where D_j and C_j are the data and claim of the embedded layout(s). Finally, conjoin the warrant(s), W_j, of each embedded layout to that of the layout in which it was embedded, W_k. The following simple layout will result:

$$\left\langle \bigcup_j (D_j \Rightarrow C_j), W_k \wedge \bigwedge_j W_j, C_k \right\rangle \tag{2}$$

In representing the embedded layout propositionally, we are again assuming that a single account of derivation is in use throughout the proof.

By applying these rules recursively, it is possible to reduce a compound layout of arbitrary complexity to a simple layout. Of course, such a reduction will omit much detail, but it will preserve soundness. As a simple example, consider the representation of proof by contradiction in Fig. 4(b). Applying (1) to the embedded argument produces a simple layout $\langle C_1, W_2 \wedge W_3, C_2 \wedge \sim C_2 \rangle$. We may then use (2) to produce a single layout representing the whole argument, $\langle C_1 \Rightarrow (C_2 \wedge \sim C_2), W_1 \wedge W_2 \wedge W_3, \sim C_1 \rangle$.

3.3 An Extended Example

Figure 5 shows how the techniques introduced above may be applied to a real example, in this case the proof that every natural number greater than one has a prime factorization. The proof is by induction, in this case strong induction, rather than the weak induction exhibited in Fig. 4(d), since all the preceding cases are included in the antecedent of the inductive step. Moreover, the inductive step is itself a proof by cases, so we have more than one level of embedding. I have indicated the nested boxes in the network as (*i*)–(*iii*). However, box (*i*) is not a case of embedding, but rather a sequential pair of layouts in which new data is added at the intermediate stage. This network, and the simple layout above it are the two cases which constitute the proof of the inductive step. As such they are embedded in the data of that step, here indicated as (*ii*). Together

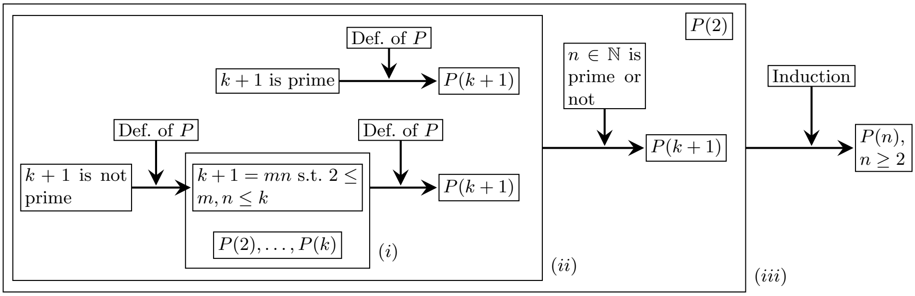


Fig. 5. Proof that every natural number greater than one has a prime factorization ($P(n)$ abbreviates 'n has a prime factorization'.)

with the base step, $P(2)$, the inductive step is itself embedded in the data (*iii*) of the outermost layout.

By applying the folding rules from Section 3.2, we may fold the whole proof into a single layout,

$$\begin{aligned}\langle\{P(2), \{k+1 \text{ is prime} \Rightarrow P(k+1),\\ \{k+1 \text{ is not prime}, P(2), \ldots, P(k)\} \Rightarrow P(k+1)\} \Rightarrow P(k+1)\},\\ \text{Def. of } P \;\wedge\; n \in \mathbb{N} \text{ is prime or not } \;\wedge\; \text{Induction},\\ P(n) \text{ for } n \geq 2\rangle.\end{aligned} \tag{3}$$

This just says that the desired result follows from the base case and the inductive step, in accordance with induction and the warrants for the inductive step. Clearly, if the unfolded proof is sound, the folded proof must be too.

4 Enhanced Layouts

For the sake of simplicity I have so far only considered combinations of layouts which lack backing, qualifiers or rebuttals. It is reasonable to omit these from analyses of the steps of a proof when they are identical in every step. This is true of many mathematical proofs. However, in some of the most interesting cases it is not. The folding rules from Sect. 3.2 may be augmented to include these additional components as follows:

$$\left\langle \bigcup_{\mathrm{in}(D_i)=0} D_i,\ \bigwedge_i W_i \left\langle \bigwedge_i B_i, \mathrm{lub}\bigcup_i Q_i, \bigvee_i R_i \right\rangle, \bigwedge_{\mathrm{out}(C_i)=0} C_i \right\rangle \tag{4}$$

$$\left\langle \bigcup_j (D_j \Rightarrow_{Q_j} C_j), W_k \wedge \bigwedge_j W_j \left\langle B_k \wedge \bigwedge_j B_j, \mathrm{lub}\{Q_k \cup \bigcup_j Q_j\}, R_k \vee \bigvee_j R_j \right\rangle, C_k \right\rangle \tag{5}$$

These rules are conservative of those for the more primitive components, with one exception. Once we permit differently qualified steps in the same proof we may no longer assume that all derivations have equal force. The univocal concept of derivation used in (2) is therefore indexed to the prevailing qualifier in (5).

The justification of the rules for backing and rebuttals is straightforward. We treat backing identically to warrants, by conjoining all the individual backings to ensure a common backing sufficient for the folded layout. Since the rebuttal of a single step is enough to rebut the entire proof, the compound rebuttal is just the disjunction of all the individual rebuttals. Combining qualifiers takes a little more care. To do so we need to make some preliminary assumptions. Firstly, that all qualifiers occurring more than once in a derivation are *non-cumulative*. We shall describe a qualifier Q as non-cumulative iff it holds of a compound layout of n steps, each with qualifier Q itself. Typically qualifiers indicating dependence on some axiom or assumption are non-cumulative, whereas qualifiers indicating likelihood are cumulative. However, for *small* values of n, qualifiers indicating high, but not absolute, levels of confidence may be treated as non-cumulative, since multiplying the possibility of error by the number of steps would still yield a very low number.[3] Secondly, we must assume that the different qualifiers Q_i may be given a partial ordering, such that $Q_j \leq Q_k$ iff every Q_j-qualified step is a Q_k-qualified step. For example, every constructively valid step is also classically valid, so 'constructively' ≤ 'classically'. Finally, we shall assume that every pair of qualifiers has a least upper bound (lub). Note that this qualifier need not itself be attached to any step of the proof.

The qualifier of the compound layout may then be defined as the least upper bound of the qualifiers of the individual steps. In some pathological cases the different steps of a proof may appeal to mutually inconsistent standards of rigour, for instance classical and Brouwerian intuitionistic mathematics. Here the least upper bound of the two qualifiers will be something falling far short of mathematical rigour (of either kind), such as 'perhaps', since the proof must be invalid.

The use of qualifiers (and backing) in a layout can make explicit the different assumptions and standards of rigour underlying different steps of the proof. We can see this in the classically but not constructively valid proof of the Intermediate Value Theorem laid out in Fig. 6. This proof has several steps which are constructively (and therefore also classically) valid, here folded together into the first step. However, there is at least one step with the qualifier 'classically', hence 'classically' must also be the compound qualifier, since it is the least upper bound of 'constructively' and 'classically'.

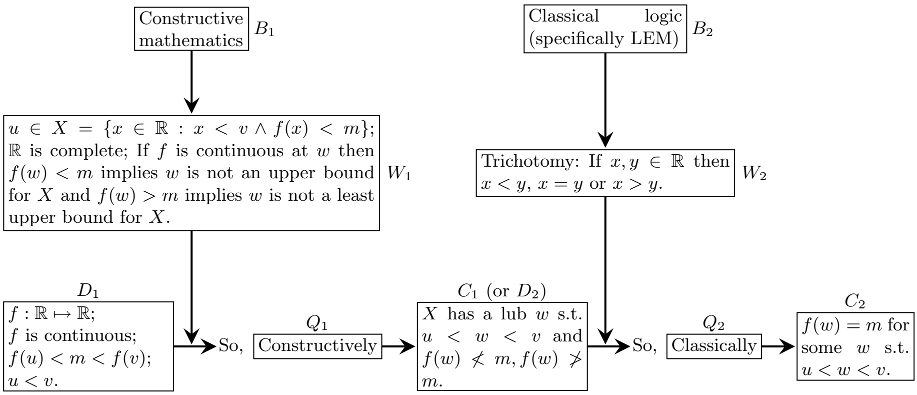

Constructive mathematics
B_1
Classical logic (specifically LEM)
B_2
$u \in X = \{x \in \mathbb{R} : x < v \wedge f(x) < m\}$; $\mathbb{R}$ is complete; If f is continuous at w then $f(w) < m$ implies w is not an upper bound for X and $f(w) > m$ implies w is not a least upper bound for X.
W_1
Trichotomy: If $x, y \in \mathbb{R}$ then $x < y$, $x = y$ or $x > y$.
W_2
D_1
$f : \mathbb{R} \mapsto \mathbb{R}$; f is continuous; $f(u) < m < f(v)$; $u < v$.
So,
Q_1
Constructively
C_1 (or D_2)
X has a lub w s.t. $u < w < v$ and $f(w) \not< m, f(w) \not> m$.
So,
Q_2
Classically
C_2
$f(w) = m$ for some w s.t. $u < w < v$.

Fig. 6. Classical proof of the Intermediate Value Theorem

5 Rebuttals

The system developed above is a potentially powerful tool for the rational reconstruction of mathematical proofs. It bears some similarity to projects such

[3] A number representing an upper bound on the possibility of error of the whole layout, since the sources of error associated with each step need not be independent.

as Leslie Lamport's 'structured proofs' (1995) or Michael Kohlhase's OMDoc (2000). In particular, the derive steps which comprise the OMDoc proof environment correspond fairly closely to basic Toulmin layouts. However, these steps lack any analogue to either the qualifier or the rebuttal component of a full layout. In this respect OMDoc (and structured proofs) are fit to their primary purpose: to facilitate greater formalization of proofs. Conversely, my approach is primarily intended to respect the level of (in)formality with which the proof was originally framed. Central to this pursuit is the rebuttal component. In this final section I explore how it may enrich our understanding of mathematical proof.

Ostensibly, Toulmin denies that mathematical arguments can ever be rebutted: observe the place holder for rebuttal in Fig. 2. He does accept that mathematical arguments are open to criticism, but only by challenging their 'standards of rational adequacy' (Toulmin & al., 1979, p. 133). Individual proofs may be undermined by wide-ranging shifts in mathematical rigour, but they are not subject to more specific rebuttal. But this is just to say that Toulmin's focus is on formal, not informal, mathematics. His is an entirely reasonable attitude to take to settled and formalized mathematical results. However, in the context of informal mathematics, it is equally reasonable to admit the possibility of rebuttal.

The ease with which Lakatos's rules lend themselves to translation into the idiom developed in this paper helps to confirm its usefulness in the analysis of informal mathematics. In the first rule, the conjecture corresponds to the (final) claim of a network of Toulmin layouts representing the proof. The lemmas are the initial data, the global counterexamples rebuttals of the final step in the proof, and the local counterexamples rebuttals to earlier steps. The second rule, Lakatos's technique of lemma incorporation, may then be understood as a transformation on a Toulmin layout, as represented in Fig. 7. Here we would generally expect $Q' \leq Q$, since removing the rebuttal should have strengthened the argument. It is also likely that $W' = W$ and $B' = B$, since the only additional support required for the new layout is a valid inference in most systems of logic ($\sim L \models L \Rightarrow C$). With Lakatos's Rule 4 we see a genuine dividend in the

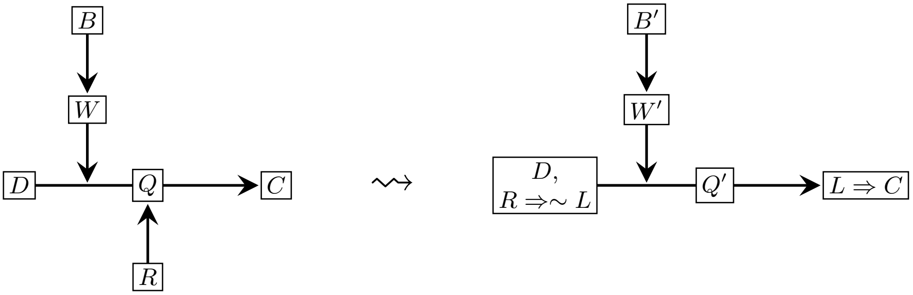


Fig. 7. Lemma incorporation

new idiom: whereas Lakatos's focus is propositional, Toulmin's is on the process of argument. Lakatos's account of this move only permits the replacement of one proposition, 'the refuted lemma', with another; our model permits the replacement of a rebutted section of a proof network, including not just data, but also warrants, backing, qualifiers, and rebuttals.

Bibliography

Aberdein, A.: The informal logic of mathematical proof. In R. Hersh, ed., *18 Unconventional Essays about the Nature of Mathematics*, Springer-Verlag, New York, NY, 2006.

Govier, T.: *Problems in Argument Analysis and Evaluation*, Foris, Dordrecht, 1987.

Johnson, R.: *The Rise of Informal Logic*, Vale, Newport News, VA, 1996.

Jones, V.: A credo of sorts. In H. G. Dales, G. Oliveri, eds., *Truth in Mathematics*, Clarendon, Oxford, 1998.

Kirschner, P. A., Buckingham Shum, S. J., Carr, C. S., eds.: *Visualizing Argumentation: Software Tools for Collaborative and Educational Sense-Making*, Springer-Verlag, London, 2003.

Kohlhase, M.: OMDoc: An open markup format for mathematical documents. *Seki Report SR-00-02*, Fachbereich Informatik, Universität des Saarlandes, 2000, www.mathweb.org/omdoc.

Lakatos, I.: *Proofs and Refutations: The Logic of Mathematical Discovery*, J. Worrall, E. Zahar, eds., Cambridge University Press, Cambridge, 1976.

Lamport, L.: How to write a proof. *American Mathematical Monthly*, **102(7)** (1995) 600–608.

Littlewood, J. E.: *Littlewood's Miscellany*, B. Bollobás, ed., Cambridge University Press, Cambridge, 1986.

Newman, S., Marshall, C.: Pushing Toulmin too far: Learning from an argument representation scheme, *Xerox PARC technical report no. SSL-92-45*, 1992, www.csdl.tamu.edu/˜marshall/toulmin.pdf.

Reed, C., Rowe, G.: Toulmin diagrams in theory & practice: Theory neutrality in argument representation. In D. Hitchcock, D. Farr, eds., *The Uses of Argument: Proceedings of a Conference at McMaster University, 18–21 May 2005*, OSSA, Hamilton, ON, 2005.

Toulmin, S.: *The Uses of Argument*, Cambridge University Press, Cambridge, 1958.

Toulmin, S., Rieke, R., Janik, A.: *An Introduction to Reasoning*, Macmillan, New York, NY, 1979.

From Untyped to Polymorphically Typed Objects in Mathematical Web Services

William Naylor and Julian Padget

Department of Computer Science, University of Bath, UK
{wn, jap}@cs.bath.ac.uk

Abstract. OpenMath is a widely recognised approach to the semantic markup of mathematics that is often used for communication between OpenMath compliant systems. The Aldor language has a sophisticated category-based type system that was specifically developed for the purpose of modelling mathematical structures, while the system itself supports the creation of small-footprint applications suitable for deployment as web services. In this paper we present our first results of how one may perform translations from generic OpenMath objects into values in specific Aldor domains, describing how the Aldor *interface* domain `ExpressionTree` is used to achieve this. We outline our Aldor implementation of an OpenMath translator, and describe an efficient extension of this to the Parser category. In addition, the Aldor service creation and invocation mechanism are explained. Thus we are in a position to develop and deploy mathematical web services whose descriptions may be directly derived from Aldor's rich type language.

1 Introduction

Mathematical web services are becoming an important feature in the web of today and it will likely be more so in the future. Computer algebra systems are one of the major technologies that can be used to support a certain class of mathematical web services. Here we show how the Aldor computer algebra system with its sophisticated type system, can be used both as a mathematical web service constructor and as a back end for mathematical web services. Several requirements fed the design and development of Aldor, amongst them being:

- **Interoperability**, so that integrating programs written in different languages is more straightforward and in particular, this makes it more amenable than most for writing mathematical web services due to the relative ease of integration with the inevitable Java
- **Strong typing** Aldor has a sophisticated two-level type structure based on *domains of computation*, that we will refer to as *domains* and *categories*. The type system is a direct descendant of that developed over many years in the line from Scratchpad through to Axiom, where the objective was to have a type language that was sufficiently tractable for checking—but not inference—yet rich enough to be able to capture the structure of mathematics. In Aldor, a domain is an environment providing a collection of exported constants including functions—analogous to a *class* in Java—while a category is used to specify information about a domain, in terms of a

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 222–236, 2006.

collection of exports that the domain in question is required to provide—analogous to an *interface* in Java. Domains and categories in Aldor may be dependant on other Aldor objects that may be members of domains, domains themselves or categories; that is both domains and categories may be parameterized by other Aldor objects.

- **Efficiency**—both in speed and space—which is why Aldor provides a good basis for mathematical services, because Aldor programs may be compiled to provide executables with execution speeds comparable to that of C++. This allows services to be compiled prior to deployment and invoked with little overhead. Since the inception of Aldor—it was originally developed as a compiler language for the Axiom [2] computer algebra system in the early 1990s, however it has developed separately since—a number of stand-alone libraries have been developed, and specifically the *algebra* library [4] which provides a number of the domains and categories utilised by the work detailed in this paper.

A web service is not unlike a (remote) procedure in that the user must supply some inputs (arguments) and in return should receive some outputs (results). These input and output values will be represented in some communication language that it is desirable should not be system specific, because web services, so it is implied, should make no assumption about the context of the clients of the service. OpenMath[1] is our chosen language for the representation of mathematical objects for input to and output from the mathematical web services. OpenMath adopts a novel, but also historically enforced, solution to the unambiguous identification of objects, in that rather than using namespaces, which did not exist when OpenMath was first conceived, attributes indicate the referenced content dictionary and element. Thus `<OMS cd = "linalg2" name = "matrix"/>` in the example below identifies the `matrix` object in the `linalg2` content dictionary. The definitions are organised using Content Dictionaries (CDs) which may be stored in standard libraries, for example those maintained by the OpenMath society [12], or shared between applications. OpenMath may be represented in a number of ways but the accepted representation, especially as far as communication over the Internet is concerned, is in an XML (eXtensible Markup Language) [16] format.

Example 1. The matrix $\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$ *may be represented in OpenMath markup as:*

```
<OMA>
  <OMS cd = "linalg2" name = "matrix"/>
  <OMA>
    <OMS cd = "linalg2" name = "matrixrow"/>
    <OMI>1</OMI> <OMI>3</OMI>
  </OMA>
  <OMA>
    <OMS cd = "linalg2" name = "matrixrow"/>
    <OMI>2</OMI> <OMI>4</OMI>
  </OMA>
</OMA>
```

where `linalg2` *is the second linear algebra content dictionary, defining the* `matrix` *and* `matrixrow` *concepts,* `OMA` *identifies an application (of a constructor in each of*

[1] We refer to OpenMath 1.0 in this paper as it is forward compatible and we do not require any of the improved features of the newer versions.

the three cases here), `OMS` *identifies a symbol with attribute* `name` *in the given CD and* `OMI` *identifies an integer.*

The purpose of this paper is to demonstrate how the categories and domains of Aldor's type system can be used to advantage in the creation of mathematical web services and specifically, how it may be used to capture accurately the semantics of OpenMath input and subsequently how it assists in turning outputs back into OpenMath for communication to other OpenMath-aware applications.

We now outline the structure of the rest of the paper. In section 2 we describe the architecture of our service manager, discussing details of the problems that arose from the philosophical mismatch of Aldor's strong-typing and OpenMath's type agnosticism, and how we have overcome this by means of the `ExpressionTree` domain. In section 3 we discuss both the theoretical—from a type system point of view—and practical limitations of the approach we describe. In section 4 we give specific details of the service manager, how the services must be wrapped in order that they can communicate over the Internet and demonstrate how a service is invoked *via* Axis. Finally in section 5 we give an example of our service manager, demonstrating deployment of a service right through to service invocation and receiving OpenMath results over axis, which may then be converted *via* style-sheets to presentation MathML, which may subsequently be displayed by the browser.

2 Design of the Service Constructor Service

If a service is to be usefully deployed on the Internet, it must receive parameters, or input from the external world. It must also return the results of the computation to the external world. We utilise the conventional solution of a *wrapper* around the code that implements a service to provide the function of processing OpenMath received over an *Input Stream*. Consequently the OpenMath is converted into the internal Aldor representation that may then be processed by the service. On successful completion the service will produce results which are then converted to OpenMath and fed to an *Output Stream*. The input stream and output stream referred to in this description originate from a java class listening to and writing to an axis-created SOAP connection.

The first step in making Aldor OpenMath aware was the definition of an OpenMath domain. Fortunately, since there are already a set of domains implementing XML-DOM [17,10] it was relatively straightforward to adapt these for the purpose. These domains take a domain-valued parameter that must have a *Category* of character, such as `UTF8Char`. This expresses the constraint that there are a number of functions specified in the Category that the domains must export. `UTF8Char` is the character domain which we use in this presentation, it being a full implementation of the Unicode UTF-8 encoding [15]. It is written to supply the full functionality of the character encoding and utilises the lowest level operations available in Aldor to ensure high efficiency.

2.1 Interface Typing Problems and Our Solution

One of the major problems encountered when performing translations between OpenMath and Aldor is that to a large extent OpenMath is type agnostic. That is to say that the objects appearing in OpenMath markup have no type information attributed to them; it is possible to give type information by means of `OMATTR` elements, however their use is not mandated by the standard and therefore we can not assume that received OpenMath markup will contain such attributes. Aldor values however are strongly typed. This means that all values, including those supplied as parameters to a function, must belong to a specific domain (or type), additionally the return value from a function must be of a specific type.

The service code supplied by its author specifies that its parameters and its return value are of specific types, and the parameters received over the input stream and the return value to be sent to the output stream have to be expressed in OpenMath. This requires that there is some translation mechanism between the heterogeneous type structure of Aldor and the single OpenMath domain. Furthermore, in order that the wrapper generator is well-structured and extensible, it is necessary to present the functionality via a carefully-defined API. We propose a solution to this problem in section 2.4 and have built a service generator based on it, which is detailed in section 4. There are a number of Categories and domains which are central to our solution which we shall detail below.

2.2 ExpressionTree

The purpose of this section is to explain how to translate between Aldor internal objects and other (external) representations. The key to the translation process is the ExpressionTree domain, which acts as a gateway to a number of external representations, *e.g.* TeX, C, Fortran, lisp or maple, and specific Aldor domains. Most of the domains in Aldor satisfy the `ExpressionType` Category[2], that is they export a function with the following signature:

```
extree : % -> ExpressionTree;
```

This should be read as meaning that the domain exports a function with name `extree`, which takes a parameter of type `%` (the domain in question) and returns a value of type `ExpressionTree`.

Example 2. The Aldor domain `DenseMatrix(R)` *represents matrices of elements of type* `R` *in a dense format,* `R` *is a type which has both the Categories* `ArithmeticType` *(meaning one can perform arithmetic operations on its elements), and* `ExpressionType` *(one can make* `ExpressionTree` *objects from its elements). The domain* `DenseMatrix` *is of the* `ExpressionType` *Category, this means that it has an* `extree` *function which can construct* `ExpressionTree` *objects from its objects, as seen in the following Aldor interpreter snippet:*

```
%1 >> mat:DenseMatrix(Integer) := [[1,2],[3,4]]
matrix [[1,3],[2,4]] @ DenseMatrix(AldorInteger)

%2 >> exmat := extree(mat)
(matrix 2 2 1 3 2 4) @ ExpressionTree
```

[2] The categories `ExpressionType`, `Parser` and `Parsable` actually require exportation of additional functions, however we are only concerned with the ones mentioned.

the presentation given after the second interpreter statement should be read as: (i) The first value indicates the type of object represented, (ii) The second two values are the dimensions of the matrix and (iii) the rest are the values in the matrix. One also notes that the type `AldorInteger` *is also of Category* `ExpressionType`. *Now that we have an* `ExpressionTree` *version of our* `DenseMatrix` *objects, we are in a position where we can transform them into a number of external formats, as follows:*

```
%3 >> axiom(stdout,exmat)    -- Axiom format
matrix [[1,3],[2,4]]   () @ TextWriter

%4 >> maple(stdout,exmat)    -- maple format
linalg[matrix](2,2,[1,3,2,4])   () @ TextWriter

%5 >> tex(stdout,exmat)      -- tex format
\pmatrix{
1 & 3 \cr
2 & 4\cr }    () @ TextWriter
```

The above deals with converting values from a specific Aldor type into a generic Aldor type, which may then be converted into a number of different external types. For our needs we must convert to the `OpenMath` type. We may achieve this by extending the `ExpressionTree` domain with a function which has signature:

```
openmath: (TextWriter, %) -> TextWriter
```

Which performs the task of converting `ExpressionTree` objects into OpenMath. This enables Aldor to communicate Aldor values to the external world in an unambiguous machine processable manner.

To communicate in the opposite direction, it is necessary to: (i) convert from the external markup to `ExpressionTree`, in our case from OpenMath to `ExpressionTree`, then (ii) from `ExpressionTree` to the specific Aldor domain. Implementation of step i) involves extending the OpenMath domain with the `Parser` category that is described in more detail in section 2.3. Step ii) requires that the target domain must be of category `Parsable`, which means that the domain must export the function with signature:

```
eval: ExpressionTree -> Partial(%)
```

This should be read as meaning that the domain exports a function `eval` which takes an `ExpressionTree` argument and returns a value of type `Partial(%)`, which is Aldor's name for lifted domains, that is domains extended by $\perp$. Aldor's name for $\perp$ is `failed`, indicating that the expression tree does not represent an object from the domain referred to by `%` (*i.e.* the domain in question), or it may be a value from the domain referred to by `%`. In the later case a value of type `%` may be obtained using the function `retract` exported by the `Partial(%)` domain.

Example 3. We shall continue example 2, and show how we can reconstruct the matrix from the `ExpressionTree` *representation. (N.B. during execution of a service the* `ExpressionTree` *objects to be evaluated will originate from* `OpenMath` *objects.)*

First we translate from `ExpressionTree` *to* `Partial(DenseMatrix(Integer))` *using the* `eval` *function, available from the* `DenseMatrix` *domain since this domain is of Category* `Parsable`.

```
%6 >> pmat:Partial(DenseMatrix(AldorInteger)) := eval(exmat)
[F matrix [[1,3],[2,4]]] @ Partial(DenseMatrix(AldorInteger))
```

Finally we translate from the `Partial(···)` *domain to the specific domain:*

```
%7 > retract(pmat)
matrix [[1,3],[2,4]] @ DenseMatrix(AldorInteger)
```

2.3 Parsing OpenMath in Aldor

In this section we describe the approach we have taken in extending the OpenMath domain with the `Parser` category. In order that the OpenMath domain satisfies the `Parser` category, it is necessary that the OpenMath domain exports a function with signature:

```
parse! : % -> ExpressionTree
```

that is, export the function `parse!` which takes an OpenMath value (the specialisation of `%` in this case) and returns the `ExpressionTree` equivalent. The naïve approach might simply traverse a table of OpenMath classes, associating one class of OpenMath objects with its `ExpressionTree` equivalent. Although, this approach would certainly work, the complexity would depend on the size of the table which would be large. The complexity would become worse as the number of OpenMath objects handled became large (*i.e.* it would not scale well). The algorithmic complexity of this process would be $O(nm)$ where n is the number of elements in the document and m is the number of different classes of OpenMath objects handled.

The approach that we have taken is to build up a hash table associating strings characterising OpenMath objects with functions taking an OpenMath object as parameter and returning `Partial(ExpressionTree)` objects. The functions may be extracted from the hash table dynamically and applied during a recursive descent of the OpenMath XML tree. The algorithmic complexity of this operation will be $O(n)$ where n is the number of child elements in the document.

Example 4. If we are translating the OpenMath element: `<OMI>10</OMI>` *into Aldor, the tag-name "OMI", is used as the key to the hash table; this is associated with a function which takes an OpenMath object as parameter and returns an* `ExpressionTree`*. This particular function takes the content of the* `OMI` *element (*10 *in this case), and returns its* `ExpressionTree` *representation.*

Example 5. If we are translating the OpenMath application element:

```
<OMA>
  <OMS cd="set" name="set1"/>
  ...
</OMA>
```

elements are members of the set being constructed. To characterise this element we concatenate the values of the `cd` *and* `name` *attributes with an "@" separator, to obtain "set@set1"*[3] *in this case. The value obtained from the hash table will be a function which we apply to the OpenMath object. The body of this function recursively performs a* `parse!` *application on all the children, bar the first, of its argument to obtain their*

[3] The characterisation used is implementation specific, we report the one that we have used.

values as ExpressionTree *objects. Consequently, we are in a position to build an* ExpressionTree *set and return this as the return value of the function.*

2.4 Solving the Type Translation Problem

Summarising our solution to the type translation problem, our method performs the following steps:

1. Read characters from the input stream (this is assumed to be OpenMath XML, if not an error will be returned to the client),
2. Convert the input stream to internal Aldor `OpenMath` objects,
3. Convert the `OpenMath` objects to `ExpressionTree` objects *via* the technique detailed in section 2.3,
4. Parse the `ExpressionTree` objects to be of type `Partial(S)` where `S` is the specific type of the particular parameter, for this step to be possible the parameter type must be of the category `Parsable`,
5. The `Partial(S)` object so obtained must now be retracted to the type `S`, in which format it may be processed by the functions provided by the service author,

After the service code has been executed, a return value will be generated with a single return type, this must be converted to OpenMath, in order that it may be sent back to the client over axis.

vi) The return type must be of category `ExpressionType`; this implies that the relevant domain exports the `extree` function which implements the transformation to `ExpressionTree`,
vii) Call the `openmath` function with which the `ExpressionTree` domain has been extended, in order to convert the `ExpressionTree` to OpenMath format and write it to the standard output.

3 Theoretical and Practical Limitations

Both the Aldor system and the OpenMath markup language are extensible. This means that one may define new domains in the former allowing one to construct novel objects to interact with the rest of the system. We may then write new OpenMath Content Dictionaries, which define new symbols allowing one to represent these objects. This extensibility has its advantages and drawbacks. On the one hand this means that if a service is utilising objects that are not handled by the system, there is a possibility that the system may be extended to deal with them, however fundamental limitations surely exist and we must determine the limitations on the scope of the system. In this section we consider the limitations on the system.

3.1 Theoretical Limitations

The basic limitations on the types of objects which may be accepted by the services are that it must be possible to perform the required translations. These limitations naturally fall into the following partitions:

Category Considerations: The type of the parameters must have the Category `Parsable`, in order that the parameter objects may be obtained from their `Expression Tree` formats. This is in order that step 4) in section 2.4 may be performed. The return type must be of Category `ExpressionType` in order that an `ExpressionTree` object may be obtained from the return value (step vi), section 2.4). If any of these domains do not have the required Category this may be rectified using Aldor's `extend` facility, which allows extra functions to be exported by a domain.

Translation from `OpenMath` to `ExpressionTree`: To effect the translation from `OpenMath` objects to `ExpressionTree` objects, we use the technique detailed in section 2.3. The `HashTable` which it utilises must be loaded with the correct functions, otherwise there is no information to specify how the transformation is to take place. In our current prototype, the set of translation methods is fixed. Clearly this is not practical for the longer term, due to the extensible nature of OpenMath which means that new CDs will be written and new translation methods will be required. We are currently considering how this may be made more flexible within the constraints and capabilities of Aldor.

Function Objects: Greater problems arise with translation to and from function objects and the remainder of this section is on-going work. In Aldor function objects are treated as first class objects which may be assigned to variables, passed as parameters and returned from functions. In OpenMath abstract functions may be represented using a λ-binding notation:

Example 6. The function which a mathematician might write as: $\lambda\ x \cdot x^2$ could be represented in OpenMath as

```
<OMBIND>
  <OMS cd="fns1" name="lambda"/>
  <OMBVAR><OMV name="x"/></OMBVAR>
  <OMA>
    <OMS cd="arith1" name="power"/>
    <OMV name="x"/>
    <OMI>2</OMI>
  </OMA>
</OMBIND>
```

Here are some of the issues this matter raises:

- The first problem is that function objects are not members of domains, and so the extend mechanism referred to earlier can not be used. It is envisaged that a special purpose *package* would be written to translate from objects of the `OpenMath` domain to objects of `ExpressionTree`, and thence to the primitive function objects.
- A second problem becomes apparent with this approach: it appears that the `ExpressionTree` framework does not supply a rich enough descriptive mechanism to describe function objects. It is hoped that this framework could be extended in some way. A different approach might be to translate directly from `OpenMath` to the primitive function objects.
- A third and apparently intractable problem is arises from the translation from Aldor to OpenMath, since this would imply decomposing the function objects into their

constituent parts. Currently no tools exist in Aldor to do this and we must leave this as future work.
- However, there is some hope: if the OpenMath input were to contain fully annotated types—this is not the normal practice—this information could be propagated through the process and potentially re-exported when needed. Indeed, annotation is probably the only way forward, given the computational intractability of type inference for type systems such as that in Aldor.

A number of the objects which exist in the algebra library have function objects as part of their representation, *e.g.* `DenseUnivariateTaylorSeries`, it is not possible to represent these objects without using functions. For the above reasons we are not currently able to translate between these objects and OpenMath.

4 Building and Using Aldor Web Services

We have built a web service manager which supports the construction and deployment of Aldor-based web services and is achieved via a set of JSP pages. One of the functionalities of the manager is to assist users in the deployment of services by writing the generic web services code automatically. The user simply supplies the code that implements the service and the web service manager sends this code as a string through Axis to a wrapper service. We built a similar system earlier as part of the MONET project [9,6] for the deployment of Maple functions as web services.

4.1 The Wrapper Service

The purpose of this wrapper service is to convert the function or functions which implement a service into a set of functions which take their parameters as OpenMath objects from an input stream and convert them into the required type for the function, similarly it will translate the return value of the function into OpenMath and write that to the output stream. To do this we must translate the code submitted by the service implementer into code to perform the actions outlined in Algorithm 1.

Construction of wrapped services is performed dynamically as the service code is received, since the specific details of the wrapped code will depend on the type of the arguments and return values of the code. We perform this service wrapper creation using a java program which implements the actions detailed in Algorithm 2.

4.2 Service Invocation

The service manager also allows invocation of the wrapped web services as follows:

1. The client selects which service they wish to invoke,
2. The client is prompted for the requisite number of parameters of the required types,
3. The client supplies these parameters in OpenMath format. It would be possible to perform type checking at this stage,
4. The request is made *via* a SOAP [14] engine, for example Axis,
5. The wrapped service, as described in section 4, is invoked and supplied with the OpenMath parameters,

Algorithm 1. Service Wrapper

```
serviceWrap():() == {
        $\{x_1, \cdots, x_n\}$ ← read OpenMath arguments from the default Input Stream
        $\{E_1, \cdots, E_n\}$ ← convert $\{x_1, \cdots, x_n\}$ to ExpressionTree
        ret:R ← service_code($E_1, \cdots, E_n$) —- R is the return type of the service code
        return openmath(stdout,extree(ret))
}

service_code($E_1$ : ExpressionTree, $\cdots$, $E_n$ : ExpressionTree):R == {
        import from Partial(D$_1$) $\cdots$ Partial(D$_n$) and D$_1$ $\cdots$ D$_n$,where D$_1$ $\cdots$ D$_n$ are
        the arguments to the service code.
        $\{e_1, \cdots, e_n\}$ ← convert the ExpressionTree objects into objects of the specific
                                types.
        the rest of the service code
```

Algorithm 2. Wrapper Creation

```
input: prog – The service code
Extract the arguments and their types from the interface function of prog.
if The argument types are not of category Parsable then
  throw a TypeNonParsable exception, whose detail records the types which are not of category Parsable(see note below).
end if
Extract the return type of the interface function of prog.
if The return type is not of category ExpressionType then
  throw a TypeNonParsable exception
end if
Build the program detailed in Algorithm 1 where the arguments are those given as the parameters of prog
Compile the constructed program, and store the executable in the service database.

Note: The type information recorded by the exception may be useful to the implementers of the
service manager, as they will then be given information about extensions required by service
implementer clients.
```

6. The service does the required processing and transmits OpenMath results which are returned (over Axis),
7. The client receives the OpenMath results, it may do further processing, *e.g.*, translation to Presentation MathML using XSLT style-sheets.

4.3 Automatic MSDL Generation

After a service has been created, it is necessary to construct its advertisement as a web service that may be stored in a UDDI-like repository and subsequently found by service brokers. A straight Web Service Description Language description is relatively unhelpful since it only contains the information necessary to invoke the service. Extended UDDI registries contain textual descriptions, but these too are of little use for software clients. These deficiencies were essentially the motivations behind the MONET

```
<monet:definitions
  targetNamespace= "http://monet.nag.co.uk/problems/">
<monet:problem name ="AntiTranspose">
  <monet:header>
    <monet:taxonomy taxonomy= "http://gams.nist.gov" code="GamsD1b"/>
  </monet:header>
  <monet:body>
    <monet:input name ="M">
      <monet:signature>
        <om:OMOBJ>
          <om:OMA>
            <om:OMS cd ="sts2" name ="matrix"/>
            <om:OMS cd="setname1" name="Z"/>
          </om:OMA>
        </om:OMOBJ>
      </monet:signature>
    </monet:input>
    <monet:output name ="A">
      <monet:signature>
        <om:OMOBJ>
          <om:OMA>
            <om:OMS cd ="sts2" name ="matrix"/>
            <om:OMS cd="setname1" name="Z"/>
          </om:OMA>
        </om:OMOBJ>
      </monet:signature>
    </monet:output>
    <monet:pre-condition>
      <om:OMOBJ>
        OpenMath for the number of columns in A = the number of rows in A
      </om:OMOBJ>
    </monet:pre-condition>
    <monet:post-condition>
      <om:OMOBJ>
        OpenMath for A_{rc} = M_{len-c+1,len-r+1} where len is the size of the matrix
      </om:OMOBJ>
    </monet:post-condition>
  </monet:body>
</monet:problem>
</monet:definitions>
```

Fig. 1. MSDL generated from the antiTranspose example (see Section 5)

project's development of Mathematical Service Description Language (MSDL) [5] that takes some inspiration from its contemporary DAML and DAML-S by describing a service in terms of pre-conditions and post-conditions. In the XML markup used to represent the MSDL document these are represented by the following elements:

- `input` elements, the signatures of the input parameters,
- `output` elements, the signatures of the return values,
- `pre-condition` elements, conditions which must hold prior to service execution and
- `post-condition` elements, conditions which must hold after the service has executed

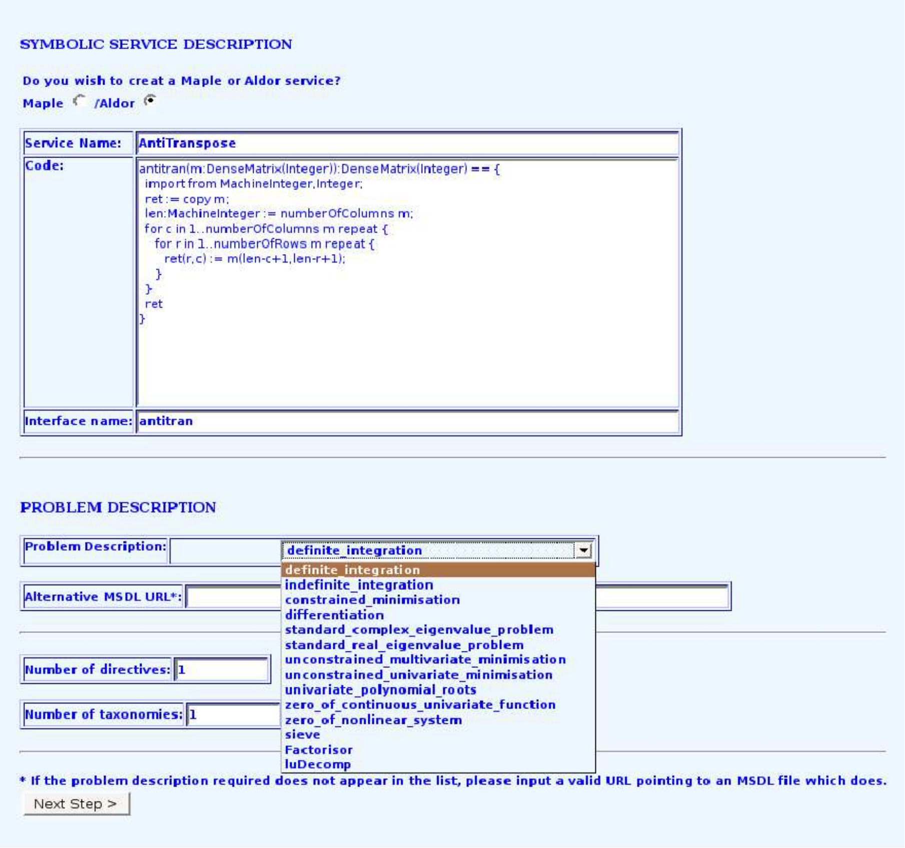
In both MONET and GENSS[4], it has been necessary to construct the MSDL descriptions mostly by hand, which is an arduous and error-prone task. A particular benefit of the capacity to translate between OpenMath and Aldor's type system is that it is possible to generate, as a side-effect of Algorithm 2, the types of the arguments to the service and the type of the return value. These values may be used to create automatically the `input` and `output` elements respectively and also some of the basic constraints for the pre- and post-conditions. The association of this information with a service is important not only for service advertisement [11], but also in checking the correctness of parameters during a service invocation.

[4] GENSS is a follow-on project to MONET; see `http://genss.cs.bath.ac.uk`

5 Example

We assume that an Aldor service has been deployed through Axis. The service we demonstrate is one that calculates the *Anti-diagonal* of a matrix *viz.* the matrix obtained by reflection about the anti-diagonal of the matrix. The code to implement calculation of the anti-diagonal:

```
antitran(m:DenseMatrix(Integer)):DenseMatrix(Integer) == {
  import from MachineInteger,Integer;
  ret := copy m;
  len:MachineInteger := numberOfColumns m;
  for c in 1..numberOfColumns m repeat {
     for r in 1..numberOfRows m repeat {
        ret(r,c) := m(len-c+1,len-r+1);
     }
  }
  ret
```



SYMBOLIC SERVICE DESCRIPTION

Do you wish to creat a Maple or Aldor service?

Maple ○ /Aldor ●

Service Name:	AntiTranspose

Code:

```
antitran(m:DenseMatrix(Integer)):DenseMatrix(Integer) == {
 import from MachineInteger,Integer;
 ret := copy m;
 len:MachineInteger := numberOfColumns m;
 for c in 1..numberOfColumns m repeat {
   for r in 1..numberOfRows m repeat {
     ret(r,c) := m(len-c+1,len-r+1);
   }
 }
 ret
}
```

Interface name: antitran

PROBLEM DESCRIPTION

Problem Description: definite_integration

definite_integration
indefinite_integration
constrained_minimisation
differentiation
standard_complex_eigenvalue_problem
standard_real_eigenvalue_problem
unconstrained_multivariate_minimisation
unconstrained_univariate_minimisation
univariate_polynomial_roots
zero_of_continuous_univariate_function
zero_of_nonlinear_system
sieve
Factorisor
luDecomp

Alternative MSDL URL*:

Number of directives: 1

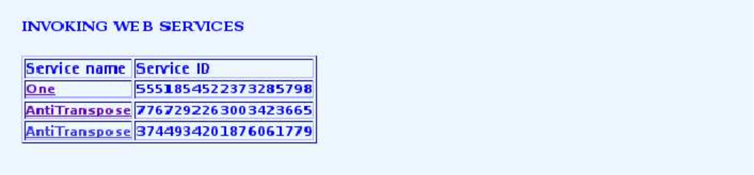
Number of taxonomies: 1

* If the problem description required does not appear in the list, please input a valid URL pointing to an MSDL file which does.

Fig. 2. Submission of Code to the service manager

INTERACTING WITH AN ALDOR WEB SERVICE

When invoking a service, the interface first invites the user to select a service from the list of services which the manager has in its database. In order to deal with the case where different instances of a service have been deployed, with or without the same implementation, the service manager associates a unique identification number with each service.

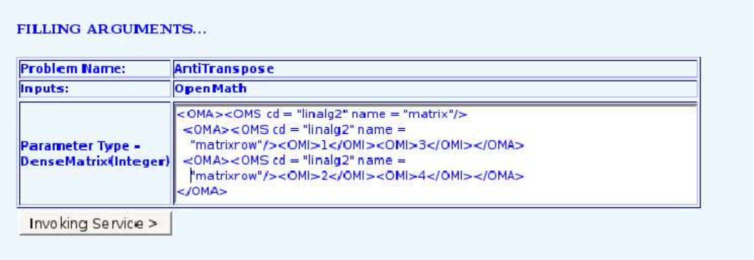


INVOKING WEB SERVICES

Service name	Service ID
One	5551854522373285798
AntiTranspose	7767292263003423665
AntiTranspose	3744934201876061779

For this naïve implementation of the manager, we use a string representation of the OpenMath parameters. A more user friendly approach would be to offer several alternative representations, *e.g.* Aldor, Mathematica, Maple format then issue a call to a translation service which would translate the parameters to OpenMath, This is a trivial extension, but is outside the scope of the current work. Returning to the example, the parameter supplied is the matrix

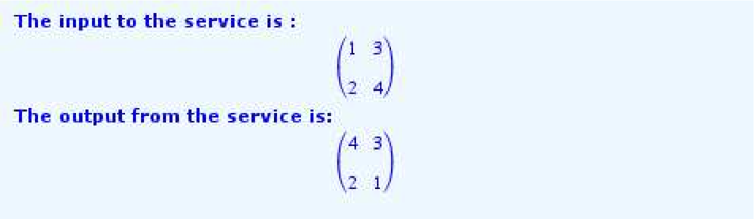
$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$



FILLING ARGUMENTS...

Problem Name:	AntiTranspose
Inputs:	OpenMath
Parameter Type - DenseMatrix(Integer)	<OMA><OMS cd = "linalg2" name = "matrix"/> <OMA><OMS cd = "linalg2" name = "matrixrow"/><OMI>1</OMI><OMI>3</OMI></OMA> <OMA><OMS cd = "linalg2" name = "matrixrow"/><OMI>2</OMI><OMI>4</OMI></OMA> </OMA>

The service is then invoked over Axis. This in turn invokes the Aldor executable created as outlined in section 4.1. The OpenMath parameters are sent to the service by a Java method over and Output Stream. This same method receives the input from the executable on an Input Streams as OpenMath, which is transmitted by axis back to the client, where the JSP page converts the OpenMath to Presentation MathML, and the following page results:



The input to the service is :

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$

The output from the service is:

$$\begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix}$$

Fig. 3. Phases of interaction with the Aldor-based web service

may be submitted to the service manager *via* the JSP page shown in figure 2. Additional information, apart from the code, which must be associated with the service are its description for the purpose of service advertisement, discovery *etc.*. The details of the interaction is shown in Fig 3.

6 Related Work

The work detailed in this paper utilises a similar architecture as that used by the service manager used in the GENSS project [8]. The services created as part of GENSS were based on the Maple computer algebra system. Conversion from OpenMath to Maple is less problematic then conversion between OpenMath and Aldor, due to the fact that Maple, being a latent-typed language, can better accommodate OpenMath's type agnosticism. This conversion however is performed by procedures written in the Maple interpreted language [13] with the relatively high overhead involved in constructing a new Maple instance every time a service is invoked. Indeed, it was the time taken in launching Maple and loading all the necessary library code that drove us to seek an alternative solution, resulting in the application of Aldor reported here. The GENSS project was a follow-on project from the EU funded MONET project under which the MSDL [5] and OpenMath based ontologies which are the underlying communication languages for these projects where developed. The Maple based service manager created in the GENSS project followed a similar method to that described by Dewar et al. [7] where the service manager developed at the University of Western Ontario is described, along with associated technologies. The MathBroker I and MathBroker II projects have focused on mathematical service brokerage [3], apparently using taxonomic information to perform service categorization and selection. It would be interesting to investigate how the MathBroker software might be used as advertisement agencies for Aldor-based services.

7 Conclusions

We have described how the Aldor language, and in particular its type system, can be used to advantage in the creation and publication of mathematical web services. By creating translators between OpenMath and Aldor objects for (a subset of) the Aldor type system, and *vice-versa*, we are able to deploy Aldor-based web services that accept OpenMath as input and generate OpenMath as output, through the use of wrapper service code. Both the wrapper code and the service code may be compiled and stored in a repository. A particular challenge of the work has been accommodation of the tension between OpenMath's absence of need for type information and Aldor's sophisticated strong typing scheme, but by using the `ExpressionTree` domain, and an efficient, linear time algorithm for the translation of OpenMath objects to Aldor objects, it is possible to handle the different Aldor objects that may have very different procedures for providing this translation. Furthermore the correspondence between Aldor types and OpenMath that has thus been established can now be applied to the signature of the service procedure and used to generate some of the necessary information to go in the MSDL description that might subsequently enable a broker to identify the service as useful.

Acknowledgements. The work reported here was partially supported by the Engineering and Physical Sciences Research Council under the Semantic Grids call of the e-Science program (grant reference GR/S44723/01).

References

1. Web Services - Axis, Apache Project. Available *via* `http://ws.apache.org/axis/` (March 2006).
2. The Axiom Computer Algebra System. Available *via* Wiki at `http://wiki.axiom-developer.org/FrontPage` (March 2006).
3. Rebhi Baraka, Olga Caprotti, and Wolfgang Schreiner. A Web Registry for Publishing and Discovering Mathematical Services. In *EEE*, pages 190–193. IEEE Computer Society, 2005.
4. Manuel Bronstein and Marc Moreno Maza. The Standard Aldor Library, Version 1.0.2. Available *via* `http://www-sop.inria.fr/cafe/Manuel.Bronstein/libaldor/html`.
5. Stephen Buswell, Olga Caprotti, and Mike Dewar. Mathematical Service Description Language. Technical report, 2003. Available from the MONET website: `http://monet.nag.co.uk/cocoon/monet/publicdocs/monet-msdl-final.pdf`.
6. Olga Caprotti, Michael Dewar, James Davenport, and Julian Padget. Mathematics on the (Semantic) Net. In Christoph Bussler, John Davies, Dieter Fensel, and Rudi Studer, editors, *Proceedings of the European Symposium on the Semantic Web*, volume 3053 of *LNCS*, pages 213–224. Springer Verlag, 2004.
7. Mike Dewar, Elena Smirnova, and Stephen Watt. XML in Mathematical Web Services. In *XML Conference proceedings*, 2005.
8. GENSS Home Page. Available from `http://genss.cs.bath.ac.uk` (March 2006).
9. MONET Home Page. Available from `http://monet.nag.co.uk` (March 2006), 2002.
10. William Naylor. The XML-DOM domain for the Aldor computer algebra system. Available *via* `http://www.cs.bath.ac.uk/~wn/Papers/usersGuide.ps`, (March 2006).
11. William Naylor and Julian Padget. Semantic Matching for Mathematical Services. In *Proceedings of the Conference on Mathematical Knowledge Management 2005*, volume 3863 of *LNAI*. Springer Verlag, 2005. In press.
12. OpenMath website. `http://www.openmath.org`, February 2002.
13. Clare So, Zheng Wang, Sandy Huerter, and Stephen Watt. An Extensible OpenMath-Maple Translator. In *East Coast Computer Algebra Day (ECCAD) 2004*. Wilfred Laurier University, Waterloo, Ontario, 2004.
14. SOAP – Simple Object Access Protocol. Available *via* `http://www.w3.org/TR/soap/` (March 2006).
15. The Unicode Standard. Available *via* `http://www.unicode.org/standard/standard.html` (March 2006).
16. Extensible Markup Language (XML), W3C. Available *via* `http://www.w3.org/XML/` (March 2006).
17. Document Object Model (DOM) Level 2 Core Specification, W3C. Available *via* `http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/` (March 2006).

Managing Automatically Formed Mathematical Theories

Simon Colton[1], Pedro Torres[1,4], Paul Cairns[2], and Volker Sorge[3]

[1] Department of Computing, Imperial College, London, UK
{sgc, ptorres}@doc.ic.ac.uk
[2] UCL Interaction Centre, University College, London, UK
p.cairns@ucl.ac.uk
[3] School of Computing, University of Birmingham, UK
V.Sorge@cs.bham.ac.uk
[4] Supported by Fundação para a Ciência e a Tecnologia, (SFRH/BD/12437/2003)

Abstract. The HR system forms scientific theories, and has found particularly successful application in domains of pure mathematics. Starting with only the axioms of an algebraic system, HR can generate dozens of example algebras, hundreds of concepts and thousands of conjectures, many of which have first order proofs. Given the overwhelming amount of knowledge produced, we have provided HR with sophisticated tools for handling this data. We present here the first full description of these management tools. Moreover, we describe how careful analysis of the theories produced by HR – which is enabled by the management tools – has led us to make interesting discoveries in algebraic domains. We demonstrate this with some illustrative results from HR's theories about an algebra of one axiom. The results fueled further developments, and led us to discover and prove a fundamental theorem about this domain.

1 Introduction

Automating the formation of mathematical theories has occupied Artificial Intelligence researchers for nearly 40 years. This fascination began with Lenat's inspirational – but ultimately flawed[1] – approach to mathematical concept formation via the AM and Eurisko programs [15], which formed concepts in set and number theory. Following these early attempts, methods for theory formation in particular domains were implemented, e.g., plane geometry [2], number systems (such as Conway numbers) [21] and non-associative algebras [12]. Particular attention has been paid to graph theory, with Epstein's GT program [8] providing a generic model for theory formation, and Fajtlowicz's Graffiti program [9] producing many conjectures, the proofs/disproofs of which have led to publication in the mathematical literature. More recent approaches to theory formation include further application to graph theory [19] and the approach by Pease et. al [18] to modeling the philosophy of mathematics championed by Lakatos [14].

[1] See [1], [20] and chapter 13 of [3] for criticisms of this work.

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 237–250, 2006.

Our contribution to mathematical theory formation has been to develop a novel descriptive machine learning algorithm, known as *automated theory formation* [3], and to apply this to discovery tasks in domains of pure mathematics, such as group theory, graph theory and number theory. Our first implementation of this technique (in the HR1 program) was written in Prolog and allowed us to investigate various concept formation and conjecture making mechanisms at a fundamental level. Our second implementation (in the HR2 program) was written in Java and addressed various disadvantages of the Prolog model. One drawback to the Prolog implementation was the lack of functionality to properly manage and present the large amounts of mathematical knowledge that was produced. In particular, we found it very difficult to extract the most interesting and important aspects of the theories produced. Hence, we have paid particular attention to implementing knowledge management tools, and HR2 boasts some fairly sophisticated ways of extracting and presenting information requested by the user. While these methods have been developed organically to meet demands from new applications, we have followed four basic principles:

- The tools implemented should be modular, hence developers should be able to augment them easily.
- The tools should enable identification of the most interesting material, even if the user's interests only come to light during or after a theory has been formed.
- Extraction of information should enable us to produce results in markup languages such as HTML, XML and LaTeX, in order to use appropriate viewers.
- The theory produced should be available for query while it is being formed.

We present here the first full description of the knowledge management techniques implemented in HR2 (hereafter referred to as simply HR). In §2, we describe the nature of the material produced by HR. We then describe the tools in HR's user interface which enable quick access of information (§3), and the more sophisticated report generation methods HR possesses (§4). We propose the hypothesis that careful use of HR's knowledge management tools enables us to extract relevant material from the theory which can lead to mathematical discovery. To add support to this hypothesis, in §5 we present the results of a series of recent sessions with HR which have led us to make some discoveries about a (relatively) little-studied algebraic domain. In §6, we summarise the management tools available in HR, and we propose improvements for future implementations of automated theory formation tools.

We will use group theory – a well known algebraic system with one operator [11] – as a running example. The operator in groups is usually denoted $*$, and this multiplies pairs of elements, x and y of a set, G, to produce a third element, $x*y$ in such a way that: (i) $*$ is associative, i.e., $\forall\, x, y \in G\,(x*(y*z) = (x*y)*z)$, (ii) there is an identity element id, such that $\forall\, x \in G\,(x*id = id*x = x)$, and (iii) each element has an inverse with respect to the identity, i.e., $\forall\, x \in G, \exists\, x^{-1} \in G$ s.t. $x * x^{-1} = x^{-1} * x = id$.

2 Theory Constituents

The HR system has many modes of operation, depending on the requirements of the user and the background information available. We focus here on two modes commonly used for forming theories about algebraic domains: (m1) the user provides only the axioms of an algebra, and (m2) the user extracts certain results from previous sessions formed using mode (m1), most importantly the example algebras produced (which we call the *objects of interest*), and provides these as background information in new sessions. HR has two main activities which add material to the theory being produced. Firstly, HR introduces new concepts by using one of 17 production rules to take a single old concept – or two old concepts – and produce a new one. In mode m2, the initial set of concepts is supplied by the user, whereas in mode m1, the concepts are extracted from the axioms supplied. The production rules are described in detail in [5] and chapter 6 of [3]. To give a flavour of how they operate and the concepts they produce, we describe how HR can construct the concept of commutative groups. It begins by constructing the concept of commutative pairs of elements by composing the background concept of group multiplication with itself. That is, given this background concept:

1.$[a, b, c, d] : b, c, d \in a \wedge b * c = d$
HR produces the following new concept using the *compose* production rule:
2.$[a, b, c, d] : b, c, d \in a \wedge b * c = d \wedge c * b = d$
HR then uses the *exists* production rule to generalise concept 2 further:
3.$[a, b, c] : b, c \in a \wedge \exists\, d$ s.t. $(b * c = d \wedge c * b = d)$
Finally, HR uses its *forall* rule to produce the concept of commutative groups:
4.$[a] : \forall\, b, c\, (b, c \in a \rightarrow \exists\, d$ s.t. $(b * c = d \wedge c * b = d))$

This construction highlights the fundamental information that HR records about concepts, namely the concept identification number, the scope of the concept, i.e., the tuple in square brackets which represents objects to which the concept applies, and the definition of the concept which tuples must satisfy. HR also records a plethora of other information about each concept produced. This includes (a) the ground instances of tuples which satisfy the concept definition, i.e., the success set of the definition (b) the construction history of the concept (c) the way in which the concept categorises the examples, and (d) various numerical values which measure how interesting the concept is, e.g., the *applicability* of a concept calculates the proportion of objects of interest which appear in the concept's success set, and the *novelty* of a concept calculates the reciprocal of the number of times the categorisation of examples afforded by that concept is also the categorisation for another concept. As discussed in [6], while the measures of interestingness drive HR's heuristic search, they are also useful for sorting and pruning the concepts it produces. In particular, we often find that concepts which score highly for novelty are the most interesting in a theory.

HR's second main activity is to make conjectures about the concepts. It does this empirically, by noticing patterns in the success sets of the concepts. For instance, if HR invents a new concept with exactly the same examples as a previous one, it makes the conjecture that the definitions of the two concepts are equivalent. Moreover, whenever a new concept is added to the theory, HR checks to see whether the new concept's examples are a subset or superset of the examples of a previous concept, and makes implication conjectures accordingly. Given an empirically formed conjecture, HR extracts simpler conjectures from it, i.e., from an equivalence conjecture, it extracts two implication conjectures, from an implication conjecture, it extracts Horn clause implicates (where a conjunction of premises implies a single goal), and from Horn clause implicates, it extracts prime implicates, where no proper subset of the premises implies the goal. In addition, HR uses the Otter theorem prover [16] and the Mace model generator [17] to prove/disprove conjectures. Otter therefore introduces a third type of theory constituent, namely proofs, and Mace introduces a fourth, namely new objects of interest (e.g., groups) which act as a counterexample to a non-theorem.

HR's conjecture making functionality is described in detail in [4]. As an illustration, in mode m1, HR is given no example groups, so it first forms the conjecture that there are actually no groups. Mace disproves this by supplying the trivial group as a counterexample. HR then conjectures that the only element in a group is the identity element and Mace supplies the cyclic group of order two as a counterexample. Later on, HR invents the concept of idempotent elements, i.e., elements, x, for which $x * x = x$. Given that the success set of this concept is exactly the same as that for the background concept of identity elements, HR makes this conjecture: $\forall G, \forall b \in G \, (b * b = b \leftrightarrow b = id)$. Otter proves this easily, and HR extracts the two obvious implication conjectures, which are already prime implicates, so no further processing is performed.

In summary, theories produced by HR include (a) objects of interest, which are introduced by the user or as counterexamples produced by a model generator (b) concepts which categorise the objects of interest (c) conjectures which relate the concepts, and (d) proofs of the conjectures. We see that there is a high degree of cross referencing between the different types of theory constituent. In addition, the volume of this material is quite large. For instance, in a mode m1 session, S, of 5000 theory formation steps lasting 7163 seconds in group theory, HR produced 9 objects of interest (groups), 306 concepts and 3017 conjectures, of which 2941 were supplied with Otter proofs. In the next two sections, we describe how HR manages this information.

3 Front-End Management Utilities

The graphical user interface to HR is extensive, consisting of more than 300 widgets (check boxes, buttons, lists, text fields, etc.,) spread over 20 screens. Nine of these screens enable the user to set up HR at the start of a theory formation session, and ten screens enable the user to probe the theory during (as HR is multi-threaded) and after theory formation. The final screen enables the user to record changes to any of the other screens into text (macro) files. Running one of these macro files enables the widget changes and button clicks to be repeated at the start of a new session, which is a useful tool. Of the ten theory-probing screens, three are devoted to directly presenting information about the theory constituents, i.e., there is a separate screen for the objects of interest, the concepts and the conjectures in the theory. There is no separate screen for the proofs, as these are attached to the conjectures which they prove.

In each of the three theory constituents screens, there is a main text area where information about a chosen theory constituent is displayed. There are also four lists which enable the user to be specific about the information presented. These lists are: (a) the constituent list, which enables the user to choose which constituent to look at (b) the details list, which enables the user to turn on and off various pieces of information about the chosen constituent (c) the pruning list, which enables the user to temporarily hide constituents which do not satisfy certain criteria, and (d) the sorting list, which enables the user to order the constituents according to various measures of interestingness. In combination, these lists enable quite specific pinpointing of aspects of the knowledge HR generates. For instance, in the concept-probing screen, there are 59 details to choose from, 23 ways to prune the concepts and 19 ways to sort the concepts. HR also provides many ways to cross reference the theory constituents, for instance by providing the list of conjectures which involve a concept in the details for that concept. HR also provides a number of other tools for probing the theory constituents directly, e.g., a text search facility in the concepts screen. HR can present concept definitions and conjecture statements in plain text, in Otter format, in Prolog format, and in TPTP format [22]. It also has an ability to simplify concept definitions, e.g., it would simplify the Otter-style definition of commutative groups from §2 above to: $4.[a] : \forall\, b, c \in G\,(b*c = c*d)$. In addition, HR uses Dot [13] to present graphical construction histories of concepts and conjectures.

To illustrate usage of these screens, after the theory formation in session S described in §2 above, we looked at the conjectures, narrowed down our view to just proved equivalences, and sorted them by the length of the proof produced by Otter. We identified this conjecture as having the longest proof (68 steps):

$$\forall bcd\,((b*c = d \wedge c*d = b \wedge b^{-1} = d) \leftrightarrow (b*b = c \wedge c*b = d \wedge c*d = b \wedge (\exists\, e\,(e*c = d))))$$

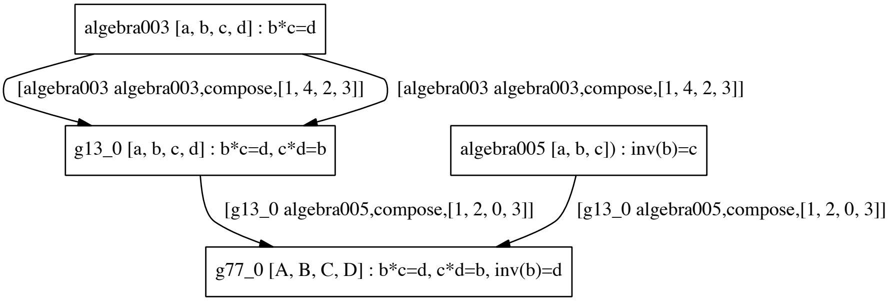

algebra003 [a, b, c, d] : b*c=d
[algebra003 algebra003,compose,[1, 4, 2, 3]]
[algebra003 algebra003,compose,[1, 4, 2, 3]]
g13_0 [a, b, c, d] : b*c=d, c*d=b
algebra005 [a, b, c]) : inv(b)=c
[g13_0 algebra005,compose,[1, 2, 0, 3]]
[g13_0 algebra005,compose,[1, 2, 0, 3]]
g77_0 [A, B, C, D] : b*c=d, c*d=b, inv(b)=d

Fig. 1. Construction history for the group theory concept appearing in the theorem with the longest Otter proof

We cross referenced the left hand concept of this equivalence conjecture and found that it had a normalised novelty value of 0.83 (hence it was quite novel). We also generated the construction history graph for the concept, as shown in figure 1. We noted that it was constructed using only the compose production rule.

4 Report Generation Techniques

An advantage of the HR1 Prolog implementation was being in an interpreted environment where bespoke queries could be constructed. This functionality is crucial, because theory formation can take many hours, and often the exact criteria which we need to pinpoint the theory constituents of interest only become clear on consideration of the formed theory. One possibility to enable bespoke queries at run-time would be to output the theory to a database, and use SQL to query it. Another possibility would be to output a Prolog representation of the theory (which HR can do) and query it using Prolog. Another possibility would be to write and compile bespoke Java classes and use class-loading at run-time.

Our solution offers, we believe, more flexibility than any of these alternatives. We have implemented a Java interpreter which uses the Java reflection mechanism to execute scripts supplied by the user at run-time. Each script is supplied with pointers to certain objects, in particular to the theory object that HR has produced. In effect, this enables the user to write a script to query in any way, any aspect of the theory. At present, the interpreter is able to handle simple arithmetic and string manipulation, for-loops, if-statements, and to call any public method or access any public field of any class, including those imported from the core Java API (which enables, amongst many other things, file input/output). As an example, the script in figure 2 prints to the standard output an XML file with the definition of each concept along with the sum of the concept's applicability and novelty values.

We take full advantage of the ability to execute scripts at run-time. In addition to a screen where the user can write scripts with the output directed to a text

```
Vector concepts = theory.concepts;
for (int c_num = 0; c_num < concepts.size(); c_num++){
  Concept concept = concepts.elementAt(c_num);
  System.out.println("<concept>");
  System.out.println("   <definition>");
  System.out.println(concept.writeDefinition("otter");
  System.out.println("   </definition>");
  System.out.println(concept.write("   <sum>"));
  System.out.println(concept.applicability + concept.novelty);
  System.out.println(concept.write("   </sum>"));
  System.out.println("</concept>")
};
```

Fig. 2. Example script for producing XML output

area, there are also text boxes on the concepts, conjectures and objects of interest screens described above, which enable the user to type in Java code to act on the theory constituent being examined. Also, the macros described in section 3 above can be embellished with Java code. Moreover, we use the interpreter inside the reports, statistics and react screens, as described below.

The report screen provides more support to the user in writing report generating scripts such as those in figure 2 above. In particular, the screen allows the user to write and save report scripts, to choose one or more scripts to run simultaneously, to run reports during/after theory formation, and to dictate after how many new theory formation steps the reports should be run. Each script is passed the theory object, and the user must specify which theory constituents (concepts, equivalences, implications, implicates, objects of interest, etc.) the script will report on. Looping through the constituents is organised internally, so the user only needs to supply Java code for outputting information either to the screen or to a file. The user can also dictate how many times the loop is performed, and can specify code to be run at the start/middle/end of each loop. This allows cumulative values to be calculated – e.g., the average applicability of concepts – and reported. We have written fairly sophisticated scripts which produce hyper-linked web pages. For example, one script lists the proved prime implicates present in the theory, with each one hyper-linked to further information, including the proof of the prime implicate.

In the statistics screen, the user is able to tabulate numerical and textual information about the theory constituents. In particular, there are three lists, from which the user can choose: the set of theory constituents to tabulate information for; the public fields for those constituents; and the public methods to be run for the constituents. There is also a text box into which the user can type Java code to apply to each of the constituents. Moreover, HR can interface with the GnuPlot program to plot information on a graph. For instance, in figure 3, we plot the average novelty of concepts as the theory progresses. Note that the graph exhibits a saw-tooth behaviour, with peaks whenever a highly novel concept is formed which decline as the concept loses its novelty. We have found such analyses to be very useful during theory formation. The user can also choose to see run-time statistics, in order to identify bottlenecks in the theory formation process.

In the react screen, the user can write scripts to be run at one of seven key points during a theory formation step (e.g., after a new concept has been added to the theory). Often, this functionality is used to alter the search according to the results of the step. However, we have often inserted report generation scripts which react to certain events, e.g., to notify us of certain highly interesting concepts as soon as they are formed.

5 Enabling Mathematical Discovery - Illustrative Results

We demonstrate here how study of the three major theory constituents (objects of interest, concepts and conjectures/theorems) that HR produces can lead to

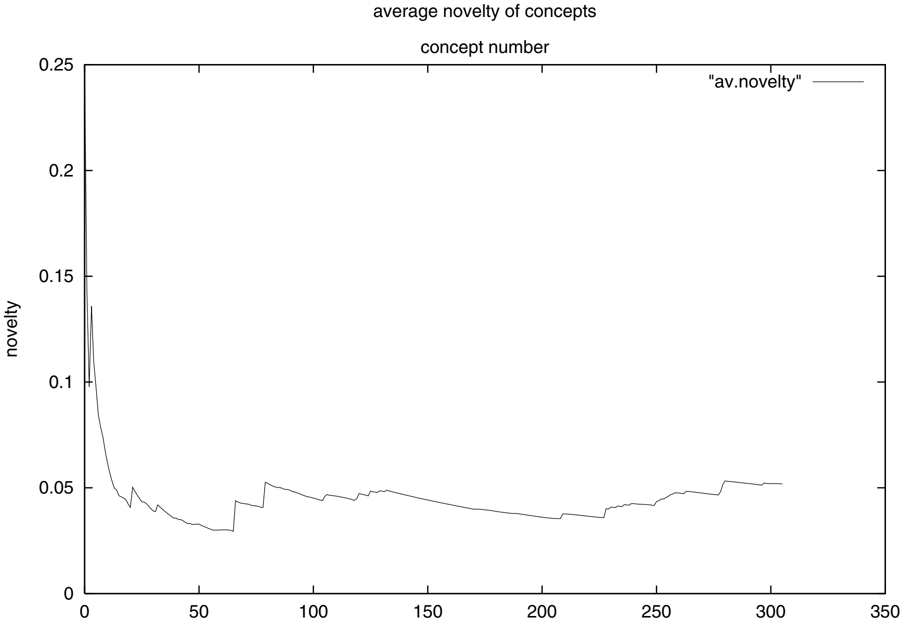


Fig. 3. Average Novelty of Concepts during Theory Formation

novel discoveries in pure mathematics. We look at a (relatively) little-studied algebraic structure which we refer to as star algebras.[2] These algebras have a single axiom, which resembles associativity: $\forall\, x, y, z\, ((x * y) * z = y * (z * x))$.

5.1 Identity Results

We first formed a theory in mode m1 (from the axioms alone) which enabled HR to generate 39 star algebras. These were then extracted and used in later theory formation sessions (mode m2). We then looked at the proved prime implicates generated by HR and we discovered some results about left and right identities. In particular, HR highlighted the fact that: $\forall\, a, b\, (b * a = a \rightarrow a * b = a)$. Paraphrased, this states that, in star algebras, any element which is a left identity for another element is also a right identity for that element. We also used the first order generic production rule [23] to specify closure under multiplication as interesting to study. Using this, HR conjectured – and Otter proved – that the elements which have a left identity are closed under multiplication, i.e., the product of two elements which both have a left identity is also an element which has a left identity.

This result also holds for elements with a right identity. However, on looking at the sub-star algebras produced by taking all elements with a right identity, we noticed that they were always the trivial algebra. Note that HR's embed-algebra

[2] Initial conversations with J.D. Phillips inspired our study of this domain. Unfortunately, we have not been able to ascertain why they are called star algebras.

production rule enabled us to look at the sub-algebras. This led us to form the incorrect hypothesis that right-identity sub-algebras are always trivial. Given that we had evidence of non-trivial left-identity sub-algebras, our hypothesis is false, as, if we take any star-algebra S and produce S' by writing $b * a$ in place of $a * b$ in the multiplication table, then S' is still a star algebra (proof omitted). Clearly, left identities in S are right identities in S', hence there are non-trivial right-identity sub-algebras.

5.2 Idempotency Results

HR also used the first order generic production rule to discover and prove that idempotent elements (such that $x * x = x$) are closed under multiplication. We cross-referenced concepts from this conjecture, and narrowed our attention to star algebras which specialise the domain. In particular, we examined the concept of idempotent star algebras (i.e., star algebras for which $\forall\, x\, (x * x = x)$). We cross-referenced this further by looking at some of the examples which satisfied the specialisation. Their multiplication tables were as follows:

	0	1	2
0	0	0	0
1	0	1	0
2	0	0	2

	0	1	2	3
0	0	0	0	0
1	0	1	0	0
2	0	0	2	0
3	0	0	0	3

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	0	0	0
2	0	0	2	0	0
3	0	0	0	3	0
4	0	0	0	0	4

We found that, in all the examples of the idempotent concept, all the non-diagonal entries on the multiplication table were the same element (zero). Given this, we hypothesised and proved that all similar constructions (of any size) produce star algebras (proof omitted). Note that we used Mace to disprove the hypothesis that such star algebras are the only idempotent ones. We used the classification system described in [7] to prove that idempotent star algebras for which all the non-diagonal entries are the same element (not necessarily zero) form an isomorphism class for sizes 6, 7 and 8. This gave us good empirical evidence and the confidence to prove that these specialisations characterise a *family* of star algebras, i.e., for every order n, these star algebras form an isomorphism class (c.f., cyclic groups in group theory, etc.).

5.3 Canonical Examples

In a separate set of experiments, we concerned ourselves with finding canonical examples of star algebras. Broadly speaking, we were looking for star algebras with certain properties which were not bestowed upon them by the fact that they satisfied another axiom set. For instance, many properties of certain star algebras are true purely because they are commutative and/or associative. Hence we were more interested in looking at non-associative, non-commutative star-algebras. We generalised this notion of canonical examples, and wrote a new measure of interestingness with respect to the objects of interest, called ImpOut (shorthand for implication outlier), as described below.

Given concepts $p(a_1, \ldots, a_n)$ and $q(a_1, \ldots, a_n)$, HR makes an implication conjecture $p \rightarrow q$ whenever the set of examples for concept p is a subset of the set of examples for q. Hence, any example satisfying the definition of p will also satisfy the definition of q. The ImpOut measure was designed to capture the idea of objects of interest which which are examples of concept q but not by virtue of being examples of concept p. More rigorously, ImpOut is a function associating a real number in the interval $[0, 1]$ to each object of interest in the theory. The corresponding ImpOut value for object of interest E is computed by considering an enumeration of all HR implication conjectures $\{p_i \rightarrow q_i\}_i$ and dividing the number of those implications for which E satisfies q_i but not p_i by the number of implications for which E satisfies q_i.

When using the ImpOut measure to sort the 39 star algebras in the theory, we noticed that the least interesting with respect to this measure were those which had a repeated row or column, such as examples A and B below, and the most interesting had no repetition, such as examples C and D below:

A	0	1
0	0	0
1	0	0

B	0	1	2	3
0	3	3	1	3
1	3	3	3	3
2	3	3	1	3
3	3	3	3	3

C	0	1	2
0	1	2	0
1	2	0	1
2	0	1	2

D	0	1	2	3
0	3	2	1	0
1	2	0	3	1
2	1	3	0	2
3	0	1	2	3

In addition, we ran another theory formation session in mode m2, where we specialised the axioms to non-commutative, non-associative star algebras. HR generated 5 examples, and in each one, we noticed a repeated row. Analysis of these and other results led us to the following hypothesis: the only non-associative, non-commutative star algebras are those with a repeated row or column in their multiplication table. We tried to prove this using the Vampire, Otter and E provers, but each failed to prove it within 72 hours of processing. We also used the Mace and Finder model generators to exhaust the search for counterexamples up to size 10, but we found none. This fueled our interest, and we eventually proved a more general result by hand, as shown below.

Definition
An algebra is said to be *k-nice* if the product of any k elements is the same regardless of bracketing or the order of the elements in the product. (This definition is taken from [10]). For instance, commutative, associative algebras are 3-nice, because multiplying a triple of elements in any way always produces the same product.

Definition
An algebra is said to be *redundant* if there exist two distinct elements x and y in S, such that, for every e in S, we have: $x * e = y * e$ and $e * x = e * y$.

Theorem
For $k > 3$, every k-nice non-commutative algebra S is redundant.

Proof
Given a pair of elements (α, β) from S, we define the following recursive algorithm: if it is possible to find an element x of S such that $x * \alpha \neq x * \beta$, then recurse with the pair $(x * \alpha, x * \beta)$. Otherwise, if it is possible to find an element y of S such that $\alpha * y \neq \beta * y$, then recurse with the pair $(\alpha * y, \beta * y)$. Otherwise, output the pair (α, β) and stop.

The first point to notice is that, if we let $\alpha = a * b$ and $\beta = b * a$ for some $a, b \in S$, then the algorithm will terminate. To see this, we note that the algorithm simply multiplies both elements of the pair by some other element at each recursion. Therefore, if the algorithm hasn't already terminated by recursion number $k - 1$, then both elements in the pair will be representable as the same set of $k - 1$ elements multiplied together, albeit in a different order (because of the reversal of the a and b in α and β). By definition of S being k-nice, if we multiply both elements in the pair by the same element (either on the right or left), then the product – of k elements – will be the same. Hence, we will not be able to find an element which multiplies on the left or right of the pair to give distinct elements, and the algorithm will terminate.

The second point to notice is that, if $\alpha \neq \beta$, then the output of the algorithm will likewise be a pair of distinct elements. This is by definition of the algorithm: at each recursion, the next pair is constructed specifically to be distinct, and the output of the algorithm is just the input to the final recursion.

Now, given that S is non-commutative, we can find $x, y \in S$ such that $x * y \neq y * x$. As shown above, if we input $(x * y, y * x)$ to the algorithm, it will terminate and output a distinct pair of elements (x', y'). These will be such that $\forall e \in S, (e * x' = e * y' \wedge x' * e = y' * e)$. Hence, S is redundant. □

Corollary
The set of non-redundant star algebras is exactly the set of non-redundant commutative and associative algebras.

Proof
In [10], Hentzel et. al. prove that star algebras are 5-nice. Therefore, the theorem above implies that non-commutative star algebras are redundant. Taking the converse, we conclude that any non-redundant star algebra, S, is commutative. S will also be associative because $\forall\, a, b \in S((a*b)*c \overset{\text{com}}{=} c*(a*b) \overset{\text{star}}{=} (b*c)*a \overset{\text{com}}{=} a * (b * c))$. In addition, as commutative, associative algebras are 3-nice, they clearly satisfy the star-algebra axiom. □

Returning to the question of canonical examples of star algebras, we can now conclude that the only non-associative, non-commutative star-algebras are those with repeated rows and columns, as the data from HR suggested. Indeed, the set of non-redundant star-algebras is the set of associative, commutative algebras, hence non-redundant star-algebras are not worthy of further study. Moreover, given that the theorem above states that non-commutative star algebras have the same pair of rows and columns repeated, we can define the following reduction algorithm which will transform a redundant star-algebra into a non-redundant one.

Reduction Algorithm
Suppose $(S, *)$ is a redundant algebra with distinct α and β satisfying the redundancy condition $\forall e\, (\alpha e = \beta e \wedge e\alpha = e\beta)$. We define the *reduction of* $(S, *)$ *with respect to* (α, β), denoted $\rho_{(\alpha,\beta)}(S, *)$, as the algebra $(S\backslash\{\beta\}, *')$, where $*'$ is defined for all $a, b \in S\backslash\{\beta\}$ as

$$a *' b = \begin{cases} \alpha & \text{if } a * b = \beta \\ a * b & \text{otherwise} \end{cases} .$$

```
while( $(S, *)$ is redundant )
    find $(\alpha, \beta)$ such that $\alpha \neq \beta \wedge \forall e\, (\alpha e = \beta e \wedge e\alpha = e\beta)$
    $(S, *) \leftarrow \rho_{(\alpha,\beta)}(S, *)$
end
```

Hence, we see that the only canonical examples worth studying are trivially reducible to commutative, associative algebras, which largely draws a line under this avenue of investigation.

6 Conclusions and Future Work

We have presented the knowledge management tools implemented in the HR system which enable us to cherry pick and cross-reference the most interesting information from the theories it forms. These tools have been developed according to four principles. Firstly, as HR is multi-threaded, the user is able to query the theory as it is being produced, and can stop or alter the search accordingly. Secondly, due to the Java interpreter in HR, the user can construct bespoke queries and generate novel types of report during and after a theory has been formed. Thirdly, the tools are modular, as demonstrated by our addition of the ImpOut measure of interestingness. Fourthly, thanks to the embedding of the interpreter in various screens, and HR's ability to write concept definitions and conjecture statements in a variety of syntaxes, the tools are able to output information in LaTeX, XML, HTML, Prolog, TPTP etc., format, enabling the usage of appropriate viewers, in addition to the output of graphical summaries of the theory and of individual theory constituents. In future implementations, we intend to build on the Java interpreter, report generator and cross-referencing of material between screens, as we have found these the most useful of HR's knowledge management tools.

We have demonstrated the flexibility and utility of the knowledge management tools during an investigation which led to some interesting mathematical results. In one sense, the investigation of star algebras from the perspective of canonical examples led to a negative result. That is, we have proved that any star algebra which is non-associative and non-commutative (properties which a canonical example should have), has these properties purely by virtue of having a redundant row and column. However, the investigation led to a more general – and surprising – result: that any k-nice non-commutative algebraic structure has redundancy. While HR and it's knowledge management tools cannot take credit for the discovery and proof of this result, it is clear that they played their part in this investigation. We believe that such interactive use of mathematical theory formation systems can enhance mathematical research, and we plan to make our techniques more powerful, and our interfaces and knowledge management tools more flexible to appeal to research mathematicians.

Acknowledgments

We would like to thank the anonymous reviewers for their interesting and instructive comments.

References

1. M Anderson. A critical evaluation of Lenat's AM program. Technical Report TR 89-09-19, Department of Computer Science, University of Washington, 1989.
2. R Bagai, V Shanbhogue, J Żytkow, and S Chou. Automatic theorem generation in plane geometry. In *LNAI 689*. Springer, 1993.
3. S Colton. *Automated Theory Formation in Pure Mathematics*. Springer, 2002.
4. S Colton. The HR program for theorem generation. In *Proceedings of CADE*, 2002.
5. S Colton, A Bundy, and T Walsh. Automatic identification of mathematical concepts. In *Proceedings of ICML*, 2000.
6. S Colton, A Bundy, and T Walsh. On the notion of interestingness in automated mathematical discovery. *International Journal of Human Computer Studies*, 53(3):351–375, 2000.
7. S Colton, A Meier, V Sorge, and R McCasland. Automatic generation of classification theorems for finite algebras. In *Proceedings of the IJCAR*, 2004.
8. S Epstein. On the discovery of mathematical theorems. In *Proceedings of the International Joint Conference on Artificial Intellignce*, 1987.
9. S Fajtlowicz. On conjectures of Graffiti. *Discrete Mathematics 72*, 23, 1988.
10. I Hentzel, D Jacobs, and S Muddana. Experimenting with the identity (xy)z = y(zx). Technical report, Clemson University, 1993.
11. J Humphreys. *A Course in Group Theory*. OUP, 1996.
12. D Jacobs. The Albert non-associative algebra system: a progress report. Technical report, Clemson University, 1994.
13. E Koutsofios and C North. Dot user's guide. Technical report, AT+T Bell Labs, Murray Hill, NJ, 1998.
14. I Lakatos. *Proofs and Refutations: The logic of mathematical discovery*. Cambridge University Press, 1976.
15. D Lenat. AM: Discovery in mathematics as heuristic search. In *Knowledge-Based Systems in Artificial Intelligence*. McGraw-Hill, 1982.
16. W McCune. The OTTER user's guide. Technical Report ANL/90/9, Argonne National Laboratories, 1990.
17. W McCune. A Davis-Putnam program and its application to finite first-order model search. Technical Report MCS-TM-194, Argonne National Labs, 1994.
18. A Pease and S Colton. Modelling Lakatos's philosophy of mathematics. In *Proceedings of the Second European Conference on Computing and Philosophy*, 2004.
19. H Pistori and J Wainer. Automatic theory formation in graph theory. In *Argentine Symposium on Artificial Intelligence*, pages 131 – 140, 1999.

20. G Ritchie and F Hanna. AM: A case study in methodology. *Artificial Intelligence*, 23, 1984.
21. M Sims. *IL: An Artificial Intelligence approach to theory formation in mathematics.* PhD thesis, Rutgers University, 1990.
22. G Sutcliffe and C Suttner. The TPTP problem library: CNF release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.
23. P Torres and S Colton. Applying model generation to concept formation. In *Proceedings of the Automated Reasoning Workshop*, 2006.

Authoring LeActiveMath Calculus Content

Paul Libbrecht and Christian Gross

[1] DFKI GmbH, Saarbrücken, Germany
[2] Ludwigs Maximilans Universtität, München, Germany

Abstract. Within the LeActiveMath project, a collection of OMDoc files and supporting material has been realized. This content covers the derivative side of calculus and is being used by students in the LeActiveMath learning environment. LeAM-calculus is the first collection trying to make use of most of the features of the learning environment including advanced usages of OpenMath and OMDoc. It has been written in OQMath, a readable xml-syntax.

This paper describes the tools to produce it, how they were used and combined, the resulting content and the experience gained. It argues that the declaration of new OpenMath symbols is a requirement and explains challenges of authoring semantic mathematical content. Finally, it presents the management activities to support the authoring process.

Introduction

This paper is an experience report on a field yet little experimented with: the creation of semantic mathematical documents for use in an integrated learning environment. It is organized as follows: first we describe the expected deliverable of the authoring activity, and its ingredients. Third-party tools that could answer these missions are covered. We provide a description of the tools used to fulfill the mission. The collection is, then, depicted in numbers and characteristics. Challenges met during the authoring activity are then described with a description of tools realized to (partially) answer them. A special accent is put on the management of mathematical knowledge. Finally, open challenges are described with a hint to future work.

1 The Mission

Among the goals of the LeActiveMath project is to prove the learning-efficiency and acceptance of the tools developed within the research project based on usage within real educational settings. For this to happen, a large content collection was to be written. This content should showcase the usage of the advanced features of the learning environment:

- It should support the semantic mathematical approach used in the OpenMath [3] encoding which makes it possible to export formulæ to such tools as computer-algebra systems for exercise evaluation [4] or to search for them.

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 251–265, 2006.

- It should support the macro-level semantics of mathematical documents as proposed by OMDoc and extended in the ActiveMath [11] and LeActiveMath projects where the domain knowledge is represented using competencies inspired by the Pisa study [13]
- It should provide a sufficient amount of learning content for the adaptivity to be realized. LeActiveMath's adaptivity is realized, mostly, using the tutorial component [15] which selects content elements from a wide choice using founded pedagogical strategies.
- It should make proper use of the tools developed in the LeActiveMath project such as the concept-mapping exercise and cognitive-support tool [10], the copy-and-paste facility, ...
- It should be available at least in German, English, and Spanish to demonstrate the language adaptivity of the learning environment.
- The content should be presented with a high-quality formula rendering within web-browsers using the presentation architecture and notation system [9].

The authors of the content should be experts in both pedagogy and mathematics and will be trained in the features of the learning environment and in the usage of the authoring tools to write such content. The authors are to stay in close contact with tool developers in order to ensure that their expectations, as educationalists and mathematicians, are met and in order for them to take advantage of the evolving features.

1.1 Elements of OMDoc Authoring

The content in LeActiveMath is encoded with the OMDoc language [7]: this language extends the OpenMath standard [3] by a formalism for mathematical documents. In turn this language has been enriched to support more pedagogical metadata [11]. The content to be input is made of textual or conceptual items which have a mathematical *role*, e.g. a proof, a definition, or an exercise. Each item has an identifier and can be served individually.

The items are annotated with knowledge to present it (e.g., titles, authorship), and for it to be inspected by the tutorial component or learner model (such as the educational context it is aimed at, its difficulty, or the competencies and concepts it is intended to). These annotations, grouped in an element called *metadata*, also contain relations between the items, e.g., the relation between an exercise and the concepts that are trained in this exercise, or the relation between a theorem and its proof(s) and corollaries. The input of these relations requires an author to know well the content or be able to browse through it easily.

The items unless purely conceptual will be presented to learners. Their content is written using text mixed with mathematical formulæ in OpenMath. This plain text is enriched with a small amount of markup such as the links to items or the embedding of resources such as pictures or applets. Each textual fragment is flagged with a language which makes it possible for the content's organization in items to be multilingual.

Many references are expected to be input by the author. They include explicit links on a piece of text, relations in the metadata, transitions in response to exercise evaluations, inclusion of an item in a *book*, as well as OpenMath symbol references in the form of OMS elements.

2 Other Approaches

In this section we describe existing approaches in the direction of authoring semantic mathematical content or content for adaptive hypermedia.

2.1 Authoring of Mathematical Semantic Content

Semantic mathematical documents to be played on the web are not, yet, commonly authored, especially with the requirements of a *macro-structure* slicing items with mathematical roles and metadata as well as that of an extensible set of mathematical symbols. To our knowledge the following tools work with the same requirements and they all work on OMDoc: STeX [8] a LaTeX-macro package which can produce OMDoc, CPoint [6] an extension to PowerPoint to support annotations of slides with OMDoc metadata, and QMath which we shall describe later. To all of these approaches we can at least state two missing wishes: error reporting about the produced OMDoc seems not possible or considered and support to suggest possible insertions while editing may be missing.

The Connexions project [5] aims at the development of an open-source commons for scholarly content. The content with formulæ is expected to be authored in XML-source form. Connexions relies on MathML-content for mathematical formulæ with little for structuring parts of the content-*modules*. Because of this, the project lacks features such as authorable notations or the context needed to provide definitions and supporting material around the introduction of a mathematical symbol as a conceptual entity.

Many other approaches exist to help in the publication of mathematical documents on the web. For most of them the ability to define items and metadata and the support for semantic mathematical objects is less important. They include LaTeX exporters, extended word-processors, mathematical assessment systems, or computer algebra systems. Only the two latter categories do export semantic mathematics and, in their case, this export is not easily extensible to support the addition of new symbols. Mathematical assessment systems, e.g. as presented at the WebALT conference[1], are still mostly centered on fixed computer algebra system whose input-syntax and formula rendering is used. An export to semantically encoded documents is rarely available.

We insist on the fact that, as has been shown in [11], the effort of authoring semantic mathematical formulæ pays on the long run. An example is the international character of the expressions which becomes language-specific when

[1] The WebALT conference took place at the beginning of the year 2006, see `http://webalt.math.helsinki.fi/webalt2006` for the programme and access to most papers.

presented (e.g. presents “ggT” or “gcd”). Such an expression authored in a non-semantic way would require different expressions for each language. Semantic formulæ are also the basis of most added value to a formula display such as the possibility to click to see definitions of a symbol or to copy-and-paste an expression.

2.2 Authoring Content for Adaptive Environments

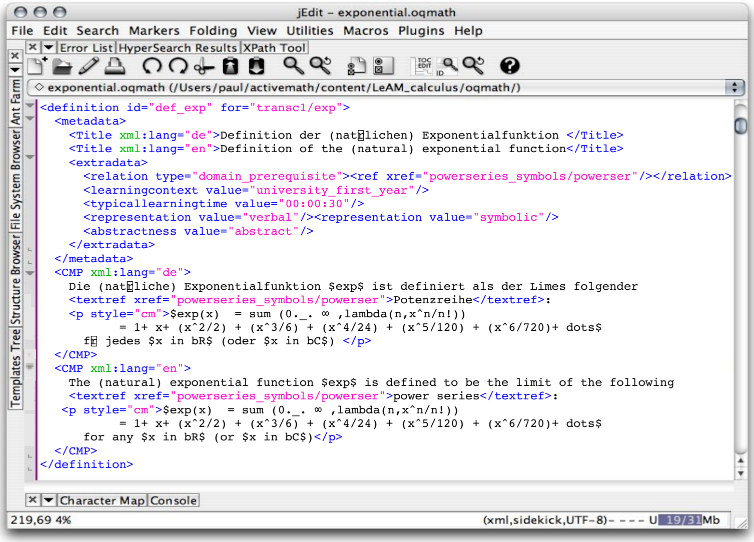
The field of authoring tools for adaptive environments, or more generally for *Advanced Technology Learning Environments* is rich and diverse. It aims at solving mostly the input of the knowledge that provides the adaptivity or pedagogical intelligence along with the more static content. A summary and reference model has been proposed in [12] which also provides general recommendations to designers of such authoring tools. Our approach differs in that jEditOQMath is based on source editing whereas the usage the WYSIWYG (*what you see is what you get*) paradigm is most recommended for micro-level content: the content of LeActiveMath being semantic, its result is multi-faceted and only relying on a browser-like view would neglect the several other perspectives under which to look at the content (in other words *to get it*). In exchange, fast test-run cycles are provided which allow to proof (or *get*) the content under any perspective the learners may be encountering. Relying on the source-and-build paradigm, jEditOQMath does also follow a classical authoring paradigm of mathematical content. Finally, as also recommended by [12], the ontology of mathematical conceptual and content items of OMDoc by their name and XML element names provides a reification that mathematicians can understand well.

Under the view of [2] providing an overview of authoring tools for adaptive hypermedia systems, LeActiveMath is among the multiple indexing systems. Adaptivity in LeActiveMath is mainly within the process of selecting items, within the tutorial component. LeActiveMath is more tight to the mathematical semantic of the macro-structure of items as opposed to the free fragment slicing described in [2]. This can be viewed as a limitation but, at the same time, provides a structure vocabulary that is easy to work with and is more realistic.

3 The Authoring Environment

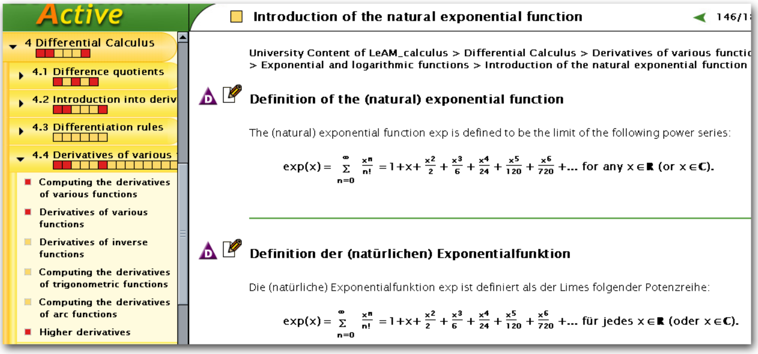
The content collection is made of source OMDoc files and static resources. In order to create the OMDoc files the authors are provided with a plain-text editor and they write and read very readable XML sources. The formulæ are input with a more compact format. After input authors can verify resulting views to the content in short edit-and-test cycles.

The current authoring environment of ActiveMath has grown out of the practice of manual editing OMDoc files by developers. The first experience was to edit complete OMDoc files by hand. One realized quickly, though, that OpenMath expressions are much too verbose to be manually input. The approach taken by the current environment, called jEditOQMath is thus to process the


jEdit – exponential.oqmath
File Edit Search Markers Folding View Utilities Macros Plugins Help
Error List HyperSearch Results XPath Tool
Templates Tree Structure Browser File System Browser Ant Farm
exponential.oqmath (/Users/paul/activemath/content/LeAM_calculus/oqmath/)
Character Map Console
219,69 4%
(xml,sidekick,UTF-8)- - - - U 19/31Mb

```
<definition id="def_exp" for="transc1/exp">
  <metadata>
    <Title xml:lang="de">Definition der (nat□lichen) Exponentialfunktion </Title>
    <Title xml:lang="en">Definition of the (natural) exponential function</Title>
    <extradata>
      <relation type="domain_prerequisite"><ref xref="powerseries_symbols/powerser"/></relation>
      <learningcontext value="university_first_year"/>
      <typicallearningtime value="00:00:30"/>
      <representation value="verbal"/><representation value="symbolic"/>
      <abstractness value="abstract"/>
    </extradata>
  </metadata>
  <CMP xml:lang="de">
    Die (nat□liche) Exponentialfunktion $exp$ ist definiert als der Limes folgender
    <textref xref="powerseries_symbols/powerser">Potenzreihe</textref>:
    <p style="cm">$exp(x)  = sum (0._. ∞ ,lambda(n,x^n/n!))
           = 1+ x+ (x^2/2) + (x^3/6) + (x^4/24) + (x^5/120) + (x^6/720)+ dots$
      f□ jedes $x in bR$ (oder $x in bC$) </p>
  </CMP>
  <CMP xml:lang="en">
    The (natural) exponential function $exp$ is defined to be the limit of the following
    <textref xref="powerseries_symbols/powerser">power series</textref>:
   <p style="cm">$exp(x)  = sum (0._. ∞ ,lambda(n,x^n/n!))
           = 1+ x+ (x^2/2) + (x^3/6) + (x^4/24) + (x^5/120) + (x^6/720)+ dots$
      for any $x in bR$ (or $x in bC$)</p>
  </CMP>
</definition>
```

Fig. 1. A OQMath source being edited, the rendering of which is in Figure 2


Active
4 Differential Calculus
4.1 Difference quotients
4.2 Introduction into deriv
4.3 Differentiation rules
4.4 Derivatives of various
Computing the derivatives of various functions
Derivatives of various functions
Derivatives of inverse functions
Computing the derivatives of trigonometric functions
Computing the derivatives of arc functions
Higher derivatives
146/1

Introduction of the natural exponential function

University Content of LeAM_calculus > Differential Calculus > Derivatives of various functi
> Exponential and logarithmic functions > Introduction of the natural exponential function

Definition of the (natural) exponential function

The (natural) exponential function exp is defined to be the limit of the following power series:

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1+x+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}+\frac{x^5}{120}+\frac{x^6}{720}+\ldots \text{ for any } x \in \mathbb{R} \text{ (or } x \in \mathbb{C}\text{).}$$

Definition der (natürlichen) Exponentialfunktion

Die (natürliche) Exponentialfunktion exp ist definiert als der Limes folgender Potenzreihe:

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1+x+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}+\frac{x^5}{120}+\frac{x^6}{720}+\ldots \text{ für jedes } x \in \mathbb{R} \text{ (oder } x \in \mathbb{C}\text{).}$$

Fig. 2. LeActiveMath with the example content from Figure 1, presented in English, resp., German (manually added)

formulæ using QMath and let the authors continue editing OMDoc files except for the formulæ.

3.1 QMath and OQMath

Using QMath[2] allows a highly readable syntax to be used for the formulæ. The syntax of QMath can even be more readable than that of TeX since QMath supports the whole range of Unicode characters. An example of input used

[2] QMath is an open-source project written in C++ and Bison by Alberto Gonzáles Palomo. More information from `http://www.matracas.org/`.

in the project is depicted in the formula in figure 1. The input of such Unicode characters is non-trivial on most keyboards but is made possible by the definition of *abbreviations* which expand to these characters; because of the common-usage of the TEX language, the abbreviations where taken from it. The mathematical input syntax can be extended by authors by the definition of new notations wrapped in files called *contexts*. Since QMath was written for complete OMDoc documents, a wrapper has been developed around it called OQMath, which extracts the context-declarations and the formulæ and replaces them by their OpenMath translation thus creating complete OMDoc documents.

The OQMath file format is using this processor and the OMDoc DTD. This allows it to be very readable and still be valid XML files. Many editors exist to edit such files, with ample support provided by grammars such as a document type definition (DTD, see [1]).

3.2 jEdit and jEditOQMath

The jEdit text editor is such an editor.[3] Its facilities for XML-editing include on-the-fly validation, supported input of child elements and their attributes, and visual folding of sub-trees. jEdit also supports well the Unicode character set, its encodings, display, and input.

As most text-editors, find-and-replace commands are supported, locally and globally. This feature has proven quite important in an ongoing development process where syntax and child elements are changed from time to time.

jEdit was extended to become jEditOQMath by allowing the start-up at the click of a link and the rapid opening and linking of an item through the usage of the drag-and-drop paradigm: the link to an item dropped in jEdit creates, depending on where it is dropped, an OMDoc `ref` element, a link, or opens the source-document of this item.

Content Packaging and Publication. In LeActiveMath, the content is organized as a series of content-collections, each with a set of OMDoc files, a set of static resources such as pictures, and a content-descriptor. LeActiveMath uses the descriptor to load the OMDoc file in content-store and identify the books. Once loaded in the content-store, items' content can be accessed individually.

jEditOQMath complements the content-collections with an OQMath directory and *build-file*. The latter is a script to perform the publishing tasks to a running server: the build first applies the OQMath process which outputs OMDoc-files, then sends a reload command to the content-store, finally, invalidates the cache in the author's LeActiveMath.

The OQMath process may complain about parsing errors in the formulæ and the content-store reload process is responsible to resolve all links and check them. Each of these steps may report errors: being part of a build script, error-reporting is done in a similar way as compilers with a quick access to the line of the document where the error is suspected. These errors are added to the

[3] jEdit is an open-source Java-based text-editor running on contemporary platforms. More information from `http://www.jedit.org/`.

XML-validity errors. They are presented with a link allowing an author to go the location to fix the error in one click. This error-reporting activity forms an important management tool which allows permanent control of the consistency of the overall content loaded in the current author's LeActiveMath installation.

3.3 Other Tools Used for LeAM-calculus

The realization of the collection required a few other tools which we briefly sketch here:

Multimedia Elements were created by third-party using classical tools such as Adobe Illustrator for graphics or the NetBeans IDE for graphical design of applets. Their embedding is done in a similar way as the embedding of a picture in an HTML-page.

Because of the text nature of the biggest part of the authored material, a versioning system with text-difference abilities, the classical CVS system, was put to use and has enabled the five persons working on the content to keep in synchronization and to publish their content in demo-servers of the LeActiveMath project. Conflicts sometimes produced by CVS merge operations were not an issue to deal with since the source files were manipulated by authors.

4 Learning to Author

In order to become familiar with the LeActiveMath learning environment as well as to start authoring, a training session was organized. Four authors took part and members of the developers' group held the session.

The first challenges came with the installation requirements for LeActiveMath (Mozilla, a recent Java, Jess, ...) then the configuration of the environment with playable content. The editor was then introduced. The primary experience that authors had with source editing approaches was the TEX typesetting system: this is often done with very simple editors, in comparison to general purpose editors with their numerous functions. For most authors, this session is also where they first wrote an XML-document. The support of jEdit for it turned out to be helpful. Once the usage of the editor, XML editing and the usage of the publication scripts were acquired, the reference mechanism in theories and imports of OMDoc was explained and the error report experimented with.

The input of expressions in QMath syntax that produce OpenMath is then taught: such a lesson starts with the discovery of OpenMath along with examples of the OpenMath content-dictionaries directly from its web-site and continues with an explanation of the QMath notation-*context* files. It ends with an experience of the symbol-reference resolution and rendering of OpenMath expressions in LeActiveMath.

Discovery of the elements possible to input at each point is done, in jEditOQMath, using the XML-grammar described by the DTD, and with the usage of templates which provide pre-filled elements with *blanks* that authors are expected to fill resembling a from.

The training was followed by an introduction into multi-steps exercises [4]. The exercise system was, then, in a very young condition and updates of the grammar were needed to allow correct checking of the syntax. This could easily be done as it only required a few files to be copied.

4.1 Other Authoring Experiences

After the training, the jEditOQMath tutorial was written as an easy introduction for starter authors. Other authoring experiences have been made. In 2004, a mathematics teacher with no experience TEX, or HTML was introduced using the tutorial. After a day, he could do all basic steps. After a week, his input of semantic mathematical formulæ was mature enough to define new symbols. This author has, since then, created large quantities of material in OQMath.

In 2006, the jEditOQMath tutorial was followed by a classroom of computer science students. It took less than 3 hours to get the installations running and their first content be written.

5 The Content Collection

Since this training, the second author of this article has produced 86 percent of the content for the LeAM-calculus collection. Within the last two years (and approximately 2000 hours) he has authored 185 symbol declarations, 138 definitions, 241 assertions (i.e., theorems, lemmas, propositions, corollaries), 207 proofs, 148 texts (introductions, motivations, notes, etc.), 308 examples, and 268 exercises, where the latter comprise fill-in-blank exercises, multiple-choice-questions, and open exercises, also some search and concept-mapping exercises, as well as multiple-steps exercises consisting of up to 2500 lines of source code. He has enriched the content with more than 23.000 textual links. When converted to a PDF form using LATEX, the collection is about 450 pages long in German, English, and Spanish. The content covers (nearly) the whole calculus material on differentiation in one variable. As demo material for the adaptivity of LeActiveMath, it is intended for various users in multiple learning contexts, ranging from collective work on pre-recorded books in a grade 11 classroom (about 16 years old) followed by individual homework repetitions up to first year University students who want to rehearse their knowledge in calculus.

Meanwhile we received feedback on the content from university students and lecturers, as well as from high school students and their teachers. In general, this feedback was very positive, most of the users were impressed by the extent of the content. Suggestions for improvement, e.g., demands for additional content, have been incorporated by the authors. In a first field study, approx. 80 grade 11 students have been working with the content in their classrooms and homework repetitions, a picture of which is at Fig 3. According to the online questionnaires they filled in, they were most impressed by the pictures and exercises, as well as by the presentation of the mathematical formulae. On the other hand, they also stated a need for improvement of the exercise hints, which is only natural since of all the scaffolded hints authored for each exercise — ranging from a first, general

Fig. 3. LEAM-CALCULUS content being used in the evaluations

hint over more detailed tips and partial results up to the complete solution — the software only presented the first one, a bug that is meanwhile fixed.

6 Management Activities of the Collection

As the content was being written it was tested, organized, adapted, retested, discussed, analysed, criticized, searched, ... we consider all these activities as *management* activities since they often require a holistic view at the content collection and how it is used.

Quality Checks. One of the first impression authors may have is that the authoring activity has nothing to do with an artistic creation but is much closer to programming. The source editing and build-process both contribute to this impression but the main reason is probably that authors are considered *final recipients of the software* which means that only when their content is integrated can a complete test be done.

The following facets of the content, used in the software, should be tested by authors using both their mathematicians' and pedagogues' eyes:

- The presentation of the textual content interleaved with formulæ, graphics, and other multimedia elements. The correct presentation of formulæ on the web is particularly challenging. The checks must be done for all languages and in the three supported output media and may be stained with platform and browser dependencies.
- The links that are clickable in the presented content, including authored links and symbol references.
- The disposition of books for the planned learning scenarios.
- The correct interpretation of the knowledge about each item encoded in its metadata. Ideally, checking this interpretation should include tests with the learner-model where the knowledge is used to build a domain-map and propagate the beliefs in the learner's mastery or the tutorial component where the knowledge is used to select the items.
- The correct behaviour of interactive exercises including the evaluation of users' input and chosen feedback elements.

This testing activity is typically done in small write-publish-test cycles which are greatly helped by a fast publication mechanism. The fact that this process is enriched with elementary reference and grammar validation is an important safeguard.

In such a research project as LeActiveMath, the authors are among the first users of the software. As a result, quality monitoring is even more important and it has not been rare that authors contribute to the design of the tools by providing early feedback to features.

Exchanging About Content Pieces. In order for members of the project to exchange about available content-pieces, care had to be made in the design of the learning environment so that individual items can be addressed by a direct URL. The identifiers of each content fragment can be inserted in such communication forms as e-mails or electronic forums.

The display's possible dependency on browsers and platforms did not prevent exchanges of screen-shots to show the current situation an author was experiencing. The reproduction of such situations was, sometimes a difficult task.

One of the types of interactions where no good solution was found for its description, and where even screenshots have sometimes proved useless, are paths of interactive exercises: one could only describe sequences of inputs which were not includable in e-mails if done in OpenMath using the input-editor.

Since developers are, all, working with mathematical objects in OpenMath, advice to authors was made in OpenMath. Very often, however, authors did communicate with QMath expressions. Such messages took much longer to be interpreted.

Keeping an Overview. The content collection is large already but still reflects only a part of the normal calculus curriculum (e.g., topics about integration were completely left out). Tools to help authors and others assessing the content in obtaining an overview of the content are needed:

- The first of these is the automated production of a LeActiveMath book where each page contains the content of each file. This allows quick access to the content being created. The maintenance of other books is done as XML-source editing, relying, among others, on the ability to drag-and-drop a content item within a book's table-of-contents' source.
- Moreover, in order to have an eye on the coverage of the possible target learners and competencies, a catalogue of exercises accessible along their characteristics was done.
- Finally, when writing new notations for the tools of [9], the set of prototype expressions and their associated presentations for all the symbols stored in a LeActiveMath installation are presented. The comparative overview of the symbols, their notations, and argument priorities is presented.

Overall the tools provided to manage content collections of content are blended within the learning environment. This empowers the authors, and probably other expert user, with methods to verify the quality of the content played within the learning environment. As principles of these tools are the fast test-and-edit cycles as well as the one click edition of content under the eyes.

7 Challenges of Authoring Semantic Mathematics

In this section we cover the difficult task of inputting semantic mathematical formulæ. This input is critical to guarantee interoperability with other systems, long term preservation, as well as advanced functions of the learning environment. The tools that we have described above apply to most mathematical content that can easily be input as simple text, which is the case of the majority formulæ in classical mathematics. Let us recall that we expect authors to produce all their formulæ in OpenMath in a conformant fashion.

Authoring Formulæ. When first starting to input formulæ, an author needs to find typical patterns of input. This is often found in OpenMath content dictionaries. From these patterns, he can find ways and possibly define input-notations so as to enter such expressions in QMath syntax. Finding these patterns is done by crawling through the content dictionaries and finding the input-notations is done by crawling through QMath context-files. This browsing activity is not comfortable and an integrated search would certainly be helpful. Built-in support for the input of elementary symbols of the formulæ within the editor is also wished. Contemporary integrated development environments provide examples of such support. The process-oriented nature of QMath and OQMath does not offer suitable introspection mechanism for such a support; other parsing facilities have to be provided.

Normal mathematicians conceive their mathematical objects with a good deal of semantics but they are mostly used to write them as presentation. The need to write it semantically is challenging and is made more difficult by the great wealth of mathematical notations.

A typical example that surprises an author during his first authoring experience is the need to use the lambda construct: in the case of the sin function used in a limit, for example: the notation $\lambda.x \sin(x)$ is needed, or, as better understood by mathematicians, $x \mapsto \sin(x)$.

A classical example where semantics encoding is almost at the limit is the usage of ellipsis patterns. Partial results are incorporated in the collection such as the notation $i = 1, \ldots, k$ but a general formalism and interpretation of ellipses is ongoing research as proved in [14].

Declaring new symbols. Fortunately, OpenMath has been designed to be extensible and the declaration of new symbols is possible and even well supported by the OMDoc structure. The declaration of a new symbol goes, as follows:

- First the author defines a `symbol` element, with a name.
- This symbol can then be defined in a single or in several definitions.

- The QMath context files are enriched with notations for this.
- Using them in a QMath expression yields `OMS` elements which are now rendered with default (prefix) notation. One has, then, to define presentation notations.

Refining this presentation is done by the definition of a few `notation` elements associating patterns of OpenMath terms with MathML presentation. These notations are used by the symbol-presentation engine presented in [9]. Being based on OpenMath patterns, they are simple to author and to manage and allow to approach the wealth of notations of the mathematicians. Being based on MathML, they can approach the quality of layout of TEX which is often expected.

Finally, this symbol may be either a mild extension or a completely new concept. If a mild extension, rephrase rules can be authored which can translate expressions using these symbols to expressions in more widespread content-dictionaries-groups.

Why new symbols. The introduction of a new symbol can be seen as breaking interoperability since the semantic of this symbol may not be shared by external recipients of them. The introduction of the rephrase-rules allows at least symbols to be introduced for purely presentational reasons or with the intent of a semantic refinement (e.g. to clarify the ambiguity of the symbol `times` of `arith1` CD). An example of such is the declaration, in LeAM-calculus, of the unary-plus sign. This sign actually has no real semantic and was not introduced in the `arith1` content-dictionary. It is, however, important for the presentation as well as for pedagogical reasons. Similarly, $\mathbf{R}_+$ the set of positive real numbers is missing. Experience has proved that many of these symbols are actually translatable in expressions using symbols of the MathML CD-group as defined in [3] which is expected to be supported by many applications..

Publishing new symbols for others to use. New symbols may also take a completely new semantic, for example the ray $[PQ[$ has been introduced. Often a publication of this symbol is wished so that others use it. The publication on the web of OMDoc files might be considered sufficient. The OpenMath society has, however, maintained a growing catalogue of content dictionaries on their web-site and this is a recognized central place to discover content dictionaries.

A tool to produce content dictionaries out of OMDoc `symbol` and `example` elements is being realized. This conversion is, however, impossible in full generality. Several features of OMDoc are incompatible with the OpenMath content-dictionary format. This includes the multilingual textual fragments as well as the ability to mix text, links, and formulæ in them. The usage of a renderer of formulæ to text may make the creation of OCD files possible in the future. However, a revision of the OCD format to allow links and multilingual texts might be needed.

8 Other Challenges Encountered

Grammar Specification and Documentation. A long requested feature by authors is a complete reference documentation of the set of elements that can be input and their use. Because of the evolving knowledge representation, especially within research projects where software, knowledge representation, and content evolve together, we believe it is needed to have such a documentation bound to the knowledge representation.

The XML DTD of OMDoc can be directly read for this, or can be read with a helper. DTDs are not, however, designed to be enriched with documentation and hence DTD-documentations are very poor.

Using an XML-schema instead of a DTD may be a good solution as this standard has good support and tools for embedded documentation. The migration to such a technology may, however, make the XML-syntax much less readable bringing to the surface, for example, the various namespaces used in OMDoc documents for LeActiveMath. Such an impact may turn out intolerable for the readability of the source files which is a basis of our work.

Input and Testing of Complex Exercises. One of the challenges of realizing LeAM-calculus is the input of content for multiple-steps exercises. Multiple-steps exercises form, conceptually a large set of nodes connected by transitions triggered by inputs of the learner. The amount of such nodes, including the diagnostic oriented metadata of each node, has yielded exercises that can be larger than 2000 lines of code where the sole large-scale structure is the interaction graph. Keeping an overview through it is difficult, testing all paths of such is even more difficult especially since, thus far, little indication is provided to a tester as to which condition has been evaluated. For this reason, a graph-based authoring tool is being developed based on the spirit of *authoring by doing.*

Scalability and Distribution of Content. The current LeAM-calculus collection is already challenging the content store of LeActiveMath and more content is actually expected. A more elaborate distribution strategy is needed which should allow content collections to be distributed anywhere on the web and be used by the drop of a URL.

9 Conclusion

The realization of the LeAM-calculus collection has proven that a large content collection using tools around jEditOQMath can be written and that a readable XML-syntax has provided the right language level for authors to edit and as basis of the discussion between authors and developers in an evolving software like the one developed in the EU project LeActiveMath.

Other experiences of authoring content for the learning environment using the same tools have proved that even persons with no TEX or HTML capabilities can, within a week or two, be authoring significant content.

The source form of editing may appear primitive to many and, indeed, we received many recommendations to provide a visual authoring tool. The usage of a visual software to edit content would have needed the software to be much more engineered so as to guarantee the trust of authors, an unconditional requirement. Moreover, the evolutionary needs of the project would have required such a visual tool to be permanently adapted to reflect the changing knowledge organization. Moreover, observation of the mathematicians' practices reveals a strong bias, at University level, towards the TEX composition system hence a broad acceptance of the source and build paradigm.

Finally, this paradigm proved very useful as the requirement to track errors raised more and more important. It has been experienced in this project that ignorance of errors reported by the content store yielded easily erroneous behaviour of the latter which triggered buggy error reports of producers or consumers of the content.

The error reporting paradigm proved itself useful and more automated checks are being investigated in order to approach, for example, the completeness in the educational targets' coverage, or the consistency of mathematical macro-structure.

Acknowledgements

The realization of this content collection has been done within the EU FP6 project LeActiveMath.[4] The realization of authoring tools have been also partially supported by the BMB+F projects MISS and In2Math.

The authors of the whole set of content of LeActiveMath are Christian Gross, Marianne Moormann, Manolis Mavrikis, and Iain Gibson.

References

1. Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML). W3C Recommendation PR-xml-971208, World Wide Web Consortium (1997) Available at `http://www.w3.org/TR/PR-xml.html`.
2. Brusilovsky, P.: Developing adaptive educational hypermedia systems: From design models to authoring tools. In Murray, T., Blessing, S., Ainsworth, S., eds.: Authoring Tools for Advanced Technology Learning Environment, Kluwer Academic Publishers, Dordrecht (2003) See `http://www2.sis.pitt.edu/ peterb/papers/KluwerAuthBook.pdf`.
3. Buswell, S., Caprotti, O., Carlisle, D., Dewar, M., Gaëtano, M., Kohlhase, M.: The OpenMath standard, version 2.0 (2004) Available at `http://www.openmath.org/`.
4. G.Goguadze, Palomo, A., E.Melis: Interactivity of Exercises in ActiveMath. In: In Proceedings of the 13th International Conference on Computers in Education (ICCE 2005), Singapore (2005) 107–113

[4] This publication is partly a result of work in the context of the LeActiveMath project, funded under the 6th Framework Program of the European Community – (Contract IST-2003-507826). The author is solely responsible for its content. More information about the LeActiveMath project can be read from `http://www.leactivemath.org/`.

5. Henry, G.: Connexions: An alternative approach to publishing. In: Proceedings of ECDL 2004 European Conference on Digital Library, University of Bath (2004) See also `http://cnx.org/`.
6. Kohlhase, A.: CPoints mathematical user interface. In: Proceedings of the MathUI Workshop. (2004) Online at `http://www.activemath.org/~paul/MathUI04/`.
7. Kohlhase, M.: OMDoc: Towards an openmath representation of mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes (2000) See also `http://www.mathweb.org/omdoc`.
8. Kohlhase, M.: Semantic markup for TEX/LATEX. In: Proceedings of the MathUI Workshop. (2004) See `http://www.activemath.org/~paul/MathUI04/`.
9. Manzoor, S., Libbrecht, P., Ullrich, C., Melis, E.: Authoring presentation for openmath. In Kohlhase, M., ed.: Mathematical Knowledge Management:MKM 2005, Bremen, Germany. Volume 3863 of LNCS., Heidelberg, Springer (2006) 33–48
10. Melis, E., Kärger, P., Homik, M.: Interactive Concept Mapping in ActiveMath (iCMap). In Haake, J.M., Lucke, U., Tavangarian, D., eds.: Delfi 2005. Volume 66 of LNI., Rostock, Germany, Gesellschaft für Informatik e.V. (GI) (2005) 247–258
11. Melis, E., Büdenbender, J., Andrès, E., Frischauf, A., Goguadze, G., Libbrecht, P., Pollet, M., Ullrich, C.: Knowledge Representation and Management in ActiveMath. Annals of Mathematics and Artificial Intelligence, Special Issue on Management of Mathematical Knowledge **38**(1-3) (2003) 47–64 Volume is accessible from `http://monet.nag.co.uk/mkm/amai/index.html`.
12. Murray, T.: Principles for pedagogy-oriented knowledge based tutor authoring systems: Lessons learned and a design meta-model. In Murray, T., Blessing, S., Ainsworth, S., eds.: Authoring Tools for Advanced Technology Learning Environment, Kluwer Academic Publishers, Dordrecht (2003)
13. Niss, M.: Mathematical competencies and the learning of mathematics: the danish KOM project. Technical report (2002) See `http://www7.nationalacademies.org/mseb/mathematical_competencies_and_the_learning_of_mathematics.pdf`.
14. Pollet, M., Sorge, V., Kerber, M.: Intuitive and Formal Representations: The Case of Matrices. In: Mathematical Knowledge Management, MKM 2004. Number 3119 in LNAI, (Springer Verlag) Available from `http://www.cs.bham.ac.uk/~vxs`.
15. Ullrich, C.: Tutorial Planning: Adapting Course Generation to Today's Needs. In Grandbastien, M., ed.: Young Researcher Track Proceedings of 12th International Conference on Artificial Intelligence in Education, Amsterdam, The Netherlands (2005) 155–160

Information Retrieval and Rendering with MML Query

Grzegorz Bancerek*

Faculty of Computer Science
Białystok Technical University, Poland
bancerek@mizar.org

Abstract. Mizar, a proof-checking system, is used to build the Mizar Mathematical Library (MML). MML Query is a semantics-based tool for managing the mathematical knowledge in Mizar including searching, browsing and presentation of the evolving MML content. The tool is becoming widely used as an aid for Mizar authors and plays an essential role in the ongoing reorganization of MML. We present new features of MML Query implemented in the third release and describe the possibilities offered by them.

1 Introduction

The Mizar language is a language used for such a formalization of mathematics that is close to the vernacular used in mathematical publications. An implemented Mizar verifier is available for checking correctness of Mizar texts. The perpetual development of the Mizar system (see [11]) has resulted in the Mizar Mathematical Library (MML)—a centrally maintained library of formalized mathematics. Contributions to MML have been the main activity of the Mizar project since the late 1980's. MML is organized as an interrelated collection of Mizar articles. At this moment—February 2006—there are 937 articles in MML, occupying 66354 kB, containing 42150 theorems and 7926 definitions. The most important facts included in MML are

- Jordan Curve Theorem (JCT), [10],
- Gödel Completeness Theorem (GCT), [6],
- Fundamental Theorem of Algebra (FTA), [12]
- Reflection Theorem, [2]

JCT is a substantial achievement of the project and is the result of a long lasting cooperation between Shinshu University and the University of Białystok which was initiated by Yatsuka Nakamura in 1992 and has involved 16 people. About 70 articles[1] from the MML are devoted, directly or indirectly, to the JCT project.

Another large project within MML, called the CCL project [1,3], is aimed at formalization of the theory of continuous lattices as presented in [8]. 58 Mizar

* Partially supported by COST Action 282.

[1] These articles are not devoted only to JCT but also to Brouwer Theorem, Urysohn Theorem, Tietze Theorem, etc.

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 266–279, 2006.

articles written by 16 authors cover about 65% of the main course of the book at the moment.

Notwithstanding the above, MML's coverage of mathematical knowledge is still minuscule. Even so, information retrieval in MML became a burning issue a long time ago. The lack of searching tools, which would be more advanced than some grep-based utilities, had delayed work in the CCL project when several authors formalized interrelated parts of the theory using various and barely compatible formalizations from the MML. This prompted in 2000 efforts aiming at development of a semantics-based searching tool for MML [1] and it was the origin of the MML Query system [4]. The first release was completed in 2001 and included basic queries enabling semantic searching of library items only. The second release completed in 2002 is described in [4]. It introduced a number of searchable resources and a variety of queries enabling more advanced searching. Additionally, beginnings of semantic presentation of MML were available in this release.

The third release developed in 2004-2005 was inspired by the works aimed at presentation of the content of MML for different purposes:

- an application of MML Query in the Trial-Solution project [7] to generate semantically linked slicing of Mizar articles,
- translation of MML into the OMDoc format [9],
- semantic browsing in Emacs [5].

These investigations as well as continual development of the web interface to the MML Query system resulted in a text transformation processor MMLQT[2] which is able to interpret MML Query language. The language in third release was improved itself to satisfy requirements of MMLQT (ordered queries, version queries, and metadata queries) and to make searching with MML Query somewhat easier (non-expert searching, rough queries).

Currently, MML Query provides the following functionalities coinciding with the MKM's objective: semantic searching, semantic browsing, semantic presentation, collection of MML statistics,[3] and assistance for authoring Mizar articles with Josef Urban's Mizar mode for Emacs [14]. In consequence, the tool facilitates individual authoring as well as collaborative work in larger projects by enabling adequate serching and uniform and unambiguous presentation. Especially, MML Query provides possibility to make monographs–the uniform ordered semantic presentation of a specified piece of a theory which may be spread over the MML. These features of MML Query are also used in ongoing reorganization of MML into the Encyclopedia of Mathematics in Mizar.

2 Background

A Mizar article accepted into MML gets a unique identifier which is later used to identify MML Query elements extracted from the article, see [4]. The first

[2] MML Query Templates or MML Query Transformation.

[3] Accessible via the World Wide Web at http://mmlquery.mizar.org/

Table 1. Homonyms

Symbol	Numbers	Examples
*	110/184/185	multiplication, composition
strict	105/105/105	strict
+	79/113/117	addition
-	79/117/119	subtraction
@	49/50/50	different castings
Sum	32/34/35	sum of numbers, vectors
"	29/44/45	reverse image
*'	28/35/36	limited multiplication
.	26/176/176	application
.:	26/54/56	image

Numbers: meanings/constructors/notations

step of the extraction process is to recognize MML Query elements in the article and to generate their unambiguous representation as dli items (dli stands for *decoded library item*). Dli item is a tree and is stored in dli format which is just the tree written in prefix order with commas and brackets '(' and ')'. We hope that pretty soon this format will be changed to XML thanks to Josef Urban's great job [16] in making Mizar processes XML-based.

Dli items are expressed in terms of *constructors*. Constructors may be understood as representations (variants) of the meaning that symbols of operation, predicate, adjective, or of type have in the context their occurrence. In other words, a constructor is a pointer (hyperlink) to an appropriate definition or redefinition. Definitions introduce meanings and redefinitions introduce variants of a meaning. For example, the symbol '*' is used in the definition of the multiplication of complex mumbers ($x * y$) which is also a complex number. This meaning of the symbol '*' has 9 another variants (constructors) introduced by redefinitions like

```
definition
  let x,y be Real;
  redefine func x * y -> Real;
  coherence ....
end;
```

The symbol '*' is also used to denote the composition of relations ($R * Q$), the set of all finite sequences of a given set ($X*$), etc.

Constructors in dli items resolve overloading of symbols (and formats) which is heavily used in Mizar. Overloading of symbols means that one symbol may have more than one meaning. Even the number of left and right arguments can be the same (overloading of formats). Homonyms, symbols (and formats) with multiple meanings, are the first hindrance in text based searching in MML; Table 1 presents the most homonymous symbols (in MML there are 13 symbols with at least 20 meanings, 54 symbols with at least 10 meanings, and 196 symbols with at least 5 meanings). The second hindrance concerns the opposite situation—synonyms, some presented in Table 2. Mizar allows one to introduce synonymous notations for one constructor and the use of constructors in dli items

Table 2. Synonyms

Constructor	Numbers	Symbols (synonyms and antonyms)
$\leq$	9 / 12	’<’, <, <=, <R, <R=, >, >=, >R, >R=
$\sqsubseteq$	6 / 7	<<=, <=, >=, >>=, ~<=, ~>=
$\subseteq$	5 / 19	<=, <=‘, c=, is_a_prefix_of, is_preposition_of
non-empty	5 / 9	being_not_0, is_not_0, non-empty, with_zero, without_zero

Numbers: symbols / meanings

glues these synonymous notations together. The reconstruction of lexical context of a fragment of Mizar text which affects tokenization is the third hindrance. The tokenization does not concerns dli items as they are made after tokenization.

In dli items, all information hidden from the surface of a Mizar text is made explicit, i.e.

- all hidden arguments are reconstructed,
- all variables are explicitly qualified,
- all constructors are identified,
- all adjectives in types are listed,
- all quantifiers and their order are presented.

MML Query uses its own data base which in essence is a set of named binary relations called *basic relations*. The relations express dependences between MML Query elements. The names of basic relations are single words (abbreviations) or two word phrases which are intended to be meaningful to Mizar users. The fundamental one is relation `ref` of pairs (i, c) such that element i refers to constructor c. In other words, c occurs in dli item i and the relation opposite to `ref` is denoted by `occur`. Other examples of basic relations are given in Table 3. The full list of basic relation is available on the web[4].

The second step in extraction of MML Query from Mizar articles is the computation of several basic relations. The input to this process is: dli items, Mizar

Table 3. Basic relations

Relation	**Description** of (x, y) in the relation
`by ref`	x refers to y in the proof (in the proof of x there is a step justified by y)
`positive ref`	x refers positively to y (y occurs in dli item of x in positive context[5])
`negative occur`	x occurs negatively in y (x occurs in dli item of y in negative context)
`definition`	x is defined by y (y is the definitional theorem of x)
`notation`	constructor x is denoted by y or symbol x is used in notation y (y uses x)
`constructor`	x denotes y (y is denoted by x)
`author`	x is written by y (y is an author (a coauthor) of article x)

[4] `http://mmlquery.mizar.org/syntax.html`

[5] *Positive context* means a place in a formula under even number of negations.

articles, bibliographic files, and data base of translation patterns from the journal *Formalized Mathematics* (ISSN 1426-2630). The journal publishes positively reviewed articles from MML. The postscript rendition of articles is obtained through mechanical translation based on systematically developed translation patterns for expressions introduced in MML. Reuse of these patterns in MML Query gives a possibility for querying with (FM) keywords which often carry more appeal than MML symbols (at least to a casual user), e.g., MML symbol `VectSp` is translated into *vector space*.

3 Basic Syntax and Semantics of the MML Query Language

(The full syntax and semantics of MML Query language is available on the web[4].)

The MML Query name for an MML item is built as follows:

Article-name:*Kind-name Number*

where *Article-name* is the unique MML identifier of the source article, *Kind-name* is the abbreviation of the item kind (th - theorem, def - definitional theorem, etc.), and *Number* is the serial number of the item. In the case of theorems, *Kind-name* may be omitted (JTC is named `JORDAN:107` and `JORDAN:th 107`). A MML Query name for a symbol includes the qualifier `symbol` and the symbol itself which may be taken in single or double quotes, e.g., `symbol '+'`. Examples of basic queries are given below.

```
JORDAN:107 ref                                          (1)
{JORDAN:107, GOEDELCP:35, POLYNOM5:75, ZF_REFLE:29}     (2)
list of th from WAYBEL26                                (3)
JORDAN:107 ref butnot list of constr from JORDAN        (4)
list of th from YELLOW19 | ref                          (5)
YELLOW19:def 5 ref & occur                              (6)
list of th where [ref | filter struct]                  (7)
list of constr where notation > 1                       (8)
list of th where positive ref <= negative ref           (9)
```

The queries (4)–(9) might be written with additional brackets (which do not change the meaning):

```
(JORDAN:107 ref) butnot (list of constr from JORDAN)
(list of th from YELLOW19) | ref
(YELLOW19:def 5 ref) & occur
(list of th) where [ref | filter struct]
(list of constr) where notation > 1
(list of th) where positive ref <= negative ref
```

The query (2) consists of four important theorems: JCT, GCT, FTA, and Reflection Theorem. Other queries listed above may be read as follows: (1) all constructors appearing in JCT, (3) all theorems from article `WAYBEL26`, (4) all constructors from JCT defined in articles other than `JORDAN`, (5) all constructors appearing in any theorem from article `YELLOW19`, (6) all items which refer to all

Table 4. Semantics of basic queries

$$\llbracket \{x_1, \ldots, x_n\} \rrbracket_v = \{x_1, \ldots, x_n\} \tag{10}$$

$$\llbracket A \ \texttt{and} \ B \rrbracket_v = \llbracket A \rrbracket_v \cap \llbracket B \rrbracket_v \tag{11}$$

$$\llbracket A \ \texttt{or} \ B \rrbracket_v = \llbracket A \rrbracket_v \cup \llbracket B \rrbracket_v \tag{12}$$

$$\llbracket A \ \texttt{butnot} \ B \rrbracket_v = \llbracket A \rrbracket_v \setminus \llbracket B \rrbracket_v \tag{13}$$

$$\llbracket xR \rrbracket_v = \{y : x \llbracket R \rrbracket_v y\} \tag{14}$$

$$\llbracket x \ \texttt{in} \ R \rrbracket_v = \{y : y \llbracket R \rrbracket_v x\} \tag{15}$$

$$\llbracket A \ \texttt{|} \ R \rrbracket_v = \{y : \exists_{x \in \llbracket A \rrbracket_v} x \llbracket R \rrbracket_v y\} = \bigcup_{x \in \llbracket A \rrbracket_v} \llbracket xR \rrbracket_v \tag{16}$$

$$\llbracket A \ \texttt{\&} \ R \rrbracket_v = \emptyset \quad \textbf{if } \llbracket A \rrbracket_v = \emptyset \quad \textbf{otherwise} \tag{17}$$

$$\llbracket A \ \texttt{\&} \ R \rrbracket_v = \{y : \forall_{x \in \llbracket A \rrbracket_v} x \llbracket R \rrbracket_v y\} = \bigcap_{x \in \llbracket A \rrbracket_v} \llbracket xR \rrbracket_v \tag{18}$$

$$\llbracket A \ \texttt{where} \ R \rrbracket_v = \{x \in \llbracket A \rrbracket_v : \llbracket xR \rrbracket_v \neq \emptyset\} \tag{19}$$

$$\llbracket A \ \texttt{where} \ R \ \texttt{=} \ n \rrbracket_v = \{x \in \llbracket A \rrbracket_v : \mathrm{card}(\llbracket xR \rrbracket_v) = n\} \tag{20}$$

$$(\texttt{>}, \texttt{>=}, \texttt{<=}, \texttt{<}) \qquad\qquad (>, \geq, \leq, <)$$

$$\llbracket A \ \texttt{where} \ R \ \texttt{=} \ Q \rrbracket_v = \{x \in \llbracket A \rrbracket_v : \mathrm{card}(\llbracket xR \rrbracket_v) = \mathrm{card}(\llbracket xQ \rrbracket_v)\} \tag{21}$$

$$\llbracket x \ [\texttt{not} \ R] \rrbracket_v = \emptyset \quad \textbf{if } \llbracket xR \rrbracket_v \neq \emptyset \tag{22}$$

$$\llbracket x \ [\texttt{not} \ R] \rrbracket_v = \{x\} \quad \textbf{if } \llbracket xR \rrbracket_v = \emptyset \tag{23}$$

$$\llbracket x \ [R \ \texttt{and} \ Q] \rrbracket_v = \{y : x \llbracket R \rrbracket_v y \wedge x \llbracket Q \rrbracket_v y\} = \llbracket xR \rrbracket_v \cap \llbracket xQ \rrbracket_v \tag{24}$$

$$\llbracket x \ [R \ \texttt{or} \ Q] \rrbracket_v = \{y : x \llbracket R \rrbracket_v y \vee x \llbracket Q \rrbracket_v y\} = \llbracket xR \rrbracket_v \cup \llbracket xQ \rrbracket_v \tag{25}$$

$$\llbracket x \ [R \ \texttt{butnot} \ Q] \rrbracket_v = \{y : x \llbracket R \rrbracket_v y \wedge \neg (x \llbracket Q \rrbracket_v y)\} = \llbracket xR \rrbracket_v \setminus \llbracket xQ \rrbracket_v \tag{26}$$

$$x \ [R \ \texttt{|} \ Q] = (xR) \ \texttt{|} \ Q \tag{27}$$

$$x \ [R \ \texttt{\&} \ Q] = (xR) \ \texttt{\&} \ Q \tag{28}$$

$$\llbracket x \ \texttt{filter} \ K \rrbracket_v = \{x\} \text{ if } x \text{ is } K \tag{29}$$

$$\llbracket x \ \texttt{filter} \ K \rrbracket_v = \emptyset \text{ if } x \text{ is not } K \tag{30}$$

$$\llbracket x \ \texttt{from}(A) \rrbracket_v = \{x\} \text{ if } x \in \llbracket A \rrbracket_v \tag{31}$$

$$\llbracket x \ \texttt{from}(A) \rrbracket_v = \emptyset \text{ if } x \not\in \llbracket A \rrbracket_v \tag{32}$$

$$\llbracket x \ \texttt{const}(A) \rrbracket_v = \llbracket A \rrbracket_v \tag{33}$$

$$\llbracket x \ \texttt{id} \rrbracket_v = \{x\} \tag{34}$$

$$\llbracket x \ \texttt{=} \ y \rrbracket_v = \{x\} \text{ if } x = y \tag{35}$$

$$\llbracket x \ \texttt{=} \ y \rrbracket_v = \{\} \text{ if } x \neq y \tag{36}$$

constructors appearing in definitional theorem no 5 in article YELLOW19, (7) all theorems concerning structures, (8) all constructors with more than one notation, (9) all theorems that refer positively to a bigger number of constructors than they refer negatively.

Atomic queries are built by application of a relation to an element (1) or by listing elements explicitly (2) or implicitly (3). Compound queries are obtained by the use of binary connectives `and`, `or`, and `butnot` (4) and by use of filters and conditions. A filter query is an application of a relation to a query with ‘`|`’ (5) or with ‘`&`’ filter (6). A conditional query applies also a relation to a query (7) or, in more useful cases, either a relation and a number[6] (8) or two relations (9). Grouping in compound queries is done with the round brackets.

It is natural that in conditional queries more complex relations than the basic relations should be allowed; it is presented in (7). Compound relations are built similarly to queries with `and`, `or`, `butnot`, `|`, `&`, and the square brackets for grouping. Moreover, `not` is used for negation (see (22) in Table 4), `in` for making inverse relations to basic relations (15), and `filter` for making filtering relations from resource codes (29), (30). E.g., `in ref` and `occur` are synonyms and also `in occur` and `ref` are.

The semantics of queries depends on the version v of MML. Table 4 gives formalized semantics of basic queries. In the table x and y denote arbitrary MML Query elements, A and B - arbitrary queries, R and Q - arbitrary relations, and K denotes a MML Query resource. The answer to a query A and the interpretation of a relation R in the version v is written as $\llbracket A \rrbracket_v$ and $\llbracket R \rrbracket_v$, respectively.

4 The MMLQT Processor

MMLQT is a text processor used for rendering MML content when browsing MML Query results, making MML statistics, or generating semantically linked abstracts [5]. For example, Table 1 was produced with MMLQT processor run over the template:

```
\begin{tabular}{|c|c|l|}
  \hline
  Symbol & Numbers & Examples\\
  \hline
<mmlq type="foreach" query="list of symbol
   ordered by number of [notation|constructor|origin] reversed
   select 0-9">
  <mmlq type="value" style="-q"/> &
  <mmlq type="count" relation="notation|constructor|origin"/> /
  <mmlq type="count" relation="notation|constructor"/> /
  <mmlq type="count" relation="notation"/> & .... \\
</mmlq>
  \hline
\end{tabular}
```

[6] (7) is equivalent to ‘`list of th where [ref | filter struct] > 0`’.

When filling in the template the processor is rewriting lines 1-4, processing the loop in lines 5-12, and rewriting lines 13-14. The loop is done for 10 elements (symbols) satisfying the query in lines 5-7. For each symbol the processor writes the symbol itself (line 8) and 3 numbers: the number of meanings of the symbol (line 9), the number of constructors denoted with the symbol (line 10), and the number of notations using the symbol (line 11). Dots '. . . .' in line 11 stand for examples which must be completed by hand.

The MMLQT processor is available with Template Maker at a web page.[7]

4.1 The <mmlq> Element and Its Type Attribute

The key role in MMLQT templates is played by the XML element <mmlq> which indicates tasks to the processor. Actually, the processor uses an XML parser only for occurrences of <mmlq> and does not count any other XML elements. The text outside <mmlq> element is simply rewritten. Other behavior of the processor is controlled by the XML attribute type of <mmlq>. The value of the attribute determines the task to be performed. In Table 5 some tasks controlled by the attribute type are given.

Table 5. MMLQT tasks

type	Other attributes	Description
author		the author(s) of argument is(are) displayed (argument must be an article)
change	subject	argument is changed to subject
changever	version	the focus to MML version is changed to version
count	query or relation	the query query or argument relation is computed and the number of elements in the result is returned
explain		the meaning of argument is rendered
fillin	template	the template template is filled in
foreach	query or relation	the query query or argument relation is computed and the text inside <mmlq> is filled in with new argument for each computed element
if	query or relation	the query query or argument relation is computed and if the result is non empty then the text inside <mmlq> is filled in (without change of argument)
template	name	the template name is defined
title		the title of argument is displayed (for articles)
var	name, value	the variable name is set to the value value
version		focused MML version is displayed

The environment of the MMLQT processor includes the variable argument which keeps the MML Query name of the subject (current argument). By default, when starting to browse the result of a query, the variable argument is set to the element browsed. Its value is changed dynamically according to type's

[7] http://mmlquery.mizar.org/template-maker.php

control when filling in templates. The value may be rendered with `<mmlq type="value"/>` or with curly braces {`mmlq:argument`}. The variable `argument` is used when the `<mmlq>` element includes the attribute `relation`. Then the processor computes the query obtained by application of `relation` to `argument`. User defined variables are also allowed. They may be set with `<mmlq type="var" name="foo" value="..."/>` and the value is rendered with {`mmlq:foo`}.

The meaning of the current argument may be rendered with `<mmlq type="explain"/>` and the rendition may include semantic linking. When rendering, the uniform human-readable presentation independent from the original form as in a Mizar article is used.

4.2 Ordered Queries

The presentation of the result of a query is usually expected to be ordered and shown in reasonable chunks. This may be achieved with *ordered query*:

$$A \texttt{ ordered by } r_1, \ldots, r_k \texttt{ select } s_1, \ldots, s_n$$

where A is a query, $r_1, \ldots, r_k$ are ordering rules and $s_1, \ldots, s_n$ are selections by positions (from a number to a number). If the ordering is omitted, then the lexical order is assumed which is also added by default as rule r_{k+1}. Rules r_1, $\ldots, r_k$ may only partially order MML Query elements and using a total ordering is necessary. To reverse the order the word `reversed` is used at the end of a rule.

Among others, there are the following kinds of ordering rules:

- `lexical order` - strings are ordered lexically and numbers are ordered according to their values,

```
ABCMIZ_0:th 7 < JORDAN:th 107
ABCMIZ_0:th 7 < ABCMIZ_0:th 17
```

- `processing order` - almost chronologically,
- `value of` R - by the results of applications of the relation R,
- `number of` R - by the number of elements in the results of applications of the relation R,
- `expression` e - by the values of the expression e.

In the collection of MML statistics[3] there are the following examples of ordering (and selection):

- latest 30 articles

```
list of article ordered by processing order select 0-29
```

- authors by contribution to MML

```
list of article | author ordered by expression
12*articles1+6*articles2+4*articles3+3*articles4 reversed
```

where `articles`i is the basic relation which gives for an author a number of articles co-authored together with $i-1$ others.

- the 50 most popular theorems

```
list of thdef ordered by number of in by ref reversed
                   select 0-49
```

- 50 items with the hardest proofs

```
list of item ordered by number of by ref reversed select 0-49
```

5 New Features

In the third release of MML Query system new features were introduced to make searching with MML Query easier. The most important among them are rough queries and non-expert searching.

5.1 Rough Queries

Group queries, i.e., queries `at least`, `at most`, and `exactly`, were available in non-rough variants in the first release. The new rough variant of `at least` query

$$\texttt{at least minus}\ n\ (c_1,\ldots,c_k)$$

gives all items which refer to all constructors c_1, …, c_k excluding at most n of them. Similarly, `at most` query

$$\texttt{at most plus}\ n\ (c_1,\ldots,c_k)$$

gives all items which refer to constructors c_1, …, c_k and to at most n more other constructors. `exactly` query combines `at least` and `at most` queries. An extended form of a group query is a *rough query* which takes a number of queries q_1, …, q_k and two numbers n and m standing for minimum and maximum,

Table 6. Semantics of rough queries

$$[\![\texttt{at least minus}\ n\ (c_1,\ldots,c_k)]\!]_v = \bigcup_{I\in\mathbb{I}_k^{k-n}} \bigcap_{i\in I} [\![c_i]\!]_v \tag{37}$$

$$[\![\texttt{at most plus}\ n\ (c_1,\ldots,c_k)]\!]_v = \left\{ x \in \bigcap_{i=1}^{k} [\![c_i]\!]_v : N(x) \le n \right\} \tag{38}$$

$$[\![\texttt{exactly plus}\ n\ \texttt{minus}\ m\ (c_1,\ldots,c_k)]\!]_v = \left\{ x \in \bigcup_{I\in J} \bigcap_{i\in I} [\![c_i]\!]_v : N(x) \le n \right\} \tag{39}$$

$$\begin{aligned} [\![\texttt{at least minus}\ n\ *(A)]\!]_v &= [\![\texttt{at least minus}\ n\ (c_1,\ldots,c_k)]\!]_v \\ \text{where } \{c_1,\ldots,c_k\} &= [\![A\ \texttt{|}\ [\texttt{filter constr or filter number}]]\!]_v \end{aligned} \tag{40}$$

$$[\![\texttt{rough}\ n\text{-}m\ (q_1,\ldots,q_k)]\!]_v = \left\{ x \in \bigcup_{i=1}^{k} [\![q_i]\!]_v : n \le F(x) \le m \right\} \tag{41}$$

respectively. The result of the query is the set of all elements which are in the results of at least n and at most m queries. The formal semantics is given in Table 6.

In Table (6), $[\![c]\!]_v = [\![c\ \texttt{occur}]\!]_v$ is the set of all elements which refer to c,

$$\mathbb{I}_k^n = \{I \in 2^{\{1,\ldots,k\}} : \mathrm{card}(I) = n\}$$

is the set of all n-element subsets of $\{1, \ldots, k\}$, $J = \mathbb{I}_k^{k-m}$,

$$N(x) = \mathrm{card}([\![x\ \texttt{ref}]\!]_v \setminus \{c_1, \ldots, c_k\})$$

is the number of constructors occurring in x and different from c_1, …, c_k, and

$$F(x) = \mathrm{card}\left(\{i : x \in [\![q_i]\!]_v\ \wedge\ i \in \{1, \ldots, k\}\}\right)$$

is the number of queries from q_1, …, q_k for which x is in the result.

If only one number is presented in rough query, e.g., `rough` n $(q_1, \ldots, q_k)$, then n is the minimum and the maximum is equal to k. Instead of numbers it is possible to use words `count` and `max` to denote, respectively, k and the maximal number of queries in conjunction with non-empty result. In particular, the query `rough` $(q_1, \ldots, q_k)$ which is the shortcut for `rough max` $(q_1, \ldots, q_k)$ gives all elements fulfilling the biggest number of queries.

5.2 Non-expert Querying

The simplest queries consist of querying only symbols. It means that a user without a deeper knowledge is able to do some searching by writing only symbols (like in Google with words).

The *Mizar formula query*[8] provides such functionality

$$\texttt{Mizar}\ (s_1\ s_2\ \ldots\ s_n) \tag{42}$$

It is computed as follows. For each recognized symbol s_i a query q_i of all possible occurrences of s_i is generated. Then a query

$$q_1\ \texttt{and}\ \ldots\ \texttt{and}\ q_n$$

is tested and if the result is not empty it is the result of (42). If the result is empty, then a rough query

$$\texttt{rough}\ (q_1, \ldots, q_n)$$

is applied.

Mizar formula query has also some additional functionality which is available by the use of some MIZAR reserved words. For example, with words `theorem`, `scheme`, `definition`, and `cluster` one limits the result of Mizar formula query to the elements of the indicated kind. On the other hand, the query

$$\texttt{Mizar}\ (s_1\ \ldots\ s_n\ \texttt{implies}\ s_1'\ \ldots\ s_m')$$

[8] The query is intended to fully recognize MIZAR formulae (including recognition of formula patterns) and is partially implemented at this time.

is computed for positive occurrences of $s_1, \ldots, s_n$ and negative occurrences of $s'_1, \ldots, s'_m$. This functionality is a step towards recognition of formula patterns (φ `implies` ψ) and turns out to be quite satisfactory. Namely, a few symbols in the query are enough to yield appropriate theorems from current MML. More adequate searching by formula patterns may be realized with a *sequence query* which concerns the tree structure of dli items. This feature is under construction and requires much more experience with using (so, it is not a non-expert query).

5.3 Versions

MML is being continually developed and revised. Revisions cause disappearance or displacement of theorems and definitions. As a result an author of an article which is not yet submitted to MML may run into some problems. It happens when the author using a version of MML makes a reference to a theorem and while still working on the article switches to a newer version of MML, where the theorem has been removed or moved to another place. In both cases author must find a theorem in the newer version with meaning similar to the theorem from the older version. This task can be done with *version queries*. For example, theorem FUNCT_2:74 after a number of revisions became identical to FUNCT_2:23 and disappeared between version 4.50.934 and 4.53.937. The author may try to find a substitute of FUNCT_2:74 using the query

```
version 4.53.937 exactly * (version 4.50.934 FUNCT_2:74 ref)
```

and it returns theorem FUNCT_2:23.

Version queries allow one to list all theorems from version v_1 which are absent in version v_2:

$$(\texttt{version}\ v_1\ \texttt{list of th})\ \texttt{butnot}\ (\texttt{version}\ v_2\ \texttt{list of th}) \qquad (43)$$

and list all versions in which theorem, e.g., JORDAN:107, existed:

$$\texttt{list of version where interpretation}(\texttt{JORDAN}:107) \qquad (44)$$

6 Conclusions and Further Work

We have presented new features of MML Query which improve the capabilities of the system and allow for easier start of querying for non-experts (as well as for experts). The web interface with a semantical browser helps with performing more advanced and adequate queries, starting from MML symbols or FM keywords. As a result, MML Query is becoming widely used as an aid for Mizar authors.

Further improvement of the system should concern the XML format of Mizar articles. Dli items can include exportable data only when the XML format includes the semantic form of a full article, e.g., when proofs are included. Moreover, Josef Urban's experiments [16] with the XQuery language on XML-ized MML show new convenient functionalities which are not accessible or hardly accessible in MML Query (advanced searching concerning the tree structure is already available, but it is not flexible enough and it is too slow).

Another direction may concern the use of theorem provers and data mining techniques developed by Josef Urban in MoMM [15] and Mizar Proof Adviser.[9] Internal lemmas extracted from MML and hints for a proof of an arbitrary MIZAR formula are obtained with them and should be made available to users of MML Query.

References

1. G. Bancerek. Development of the theory of continuous lattices in MIZAR, in M. Kerber and M. Kohlhase (eds), *Symbolic Computation and Automated Reasoning*, 65-80, A. K. Peters, 2001.
2. G. Bancerek. The Reflection Theorem, *Formalized Mathematics*, 1(5):963-972, 1990.
3. G. Bancerek and P. Rudnicki. A Compendium of Continuous Lattices in MIZAR, *Journal of Automated Reasoning*, 29(3-4): 189-224, 2002.
4. G. Bancerek and P. Rudnicki. Information retrieval in MML, in A. Asperti, B. Buchberger and J. H. Davenport (eds), Proceedings of MKM 2003, Bertinoro, *LNCS*, 2594: 119-131, 2003.
5. G. Bancerek and J. Urban. Integrated Semantic Browsing of the Mizar Mathematical Library for Authoring Mizar Articles, in A. Asperti, G. Bancerek and A. Trybulec (eds), Proceedings of MKM 2004, Białowieża, *LNCS*, 3119: 44-57, 2004.
6. P. Braselmann, P. Koepke. Gödel's Completeness Theorem, *Formalized Mathematics*, 13(1):49-53, 2005.
7. I. Dahn. Management of Informal Mathematics Knowledge–Lessons Learned from the Trial-Solution Project, in F. Bai, B. Wegner, Electronic Information and Communication in Mathematics, LNCS 2730:29-43, 2003.
8. G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*, Springer-Verlag, Berlin, 1980.
9. M. Kohlhase (joint work with G. Bancerek). Towards MIZAR Mathematical Library in OMDOC format. Electronic Proceedings of the Workshop *Mathematics on the Semantic Web*, Eindhoven, 2003, `http://www.win.tue.nl/dw/monet/proceedings/kohlhase-mizarinomdoc.ps`
10. A. Korniłowicz. Jordan Curve Theorem, *Formalized Mathematics*, 13(4):481-491, 2005.
11. R. Matuszewski and P. Rudnicki. MIZAR: the first 30 years. *Mechanized Mathematics and Its Applications*, **4**(1):3–24, 2005.
12. R. Milewski. Fundamental Theorem of Algebra, *Formalized Mathematics*, 9(3): 461-470, 2001.
13. P. Rudnicki, A. Trybulec. On equivalents of well-foundedness. *Journal of Automated Reasoning*, 23(3-4):197–234, 1999.
14. J. Urban. MizarMode - an integrated proof assistance tool for the Mizar way of formalizing mathematics. Journal of Applied Logic, 2005, `doi:10.1016/j.jal.2005.10.004`

[9] `http://wiki.mizar.org/cgi-bin/twiki/view/Mizar/MizarProofAdvisor`

15. J. Urban. MoMM - fast interreduction and retrieval in large libraries of formalized mathematics. International Journal on Artificial Intelligence Tools, 15(1): 109–130, 2006.
16. J. Urban. XML-izing Mizar: making semantic processing and presentation of MML easy. in M. Kohlhase (ed), Proceedings of MKM 2005, Bremen, *LNCS*, 3863: 346-360, 2006.

Integrating Dynamic Geometry Software, Deduction Systems, and Theorem Repositories

Pedro Quaresma[1,⋆] and Predrag Janičić[2,⋆⋆]

[1] Department of Mathematics, University of Coimbra
3001-454 Coimbra, Portugal
pedro@mat.uc.pt
[2] Faculty of Mathematics, University of Belgrade
Studentski trg 16, 11000 Belgrade, Serbia & Montenegro
janicic@matf.bg.ac.yu

Abstract. The axiomatic presentation of geometry fills the gap between formal logic and our spatial intuition. The study of geometry is, and will always be, very important for a mathematical practitioner. GCLCprover, an automatic theorem prover (ATP) integrated with dynamic geometry software (DGS) gives its user a tool to bridge his/her spatial intuition with formal, Euclidean geometry proofs. GeoThms, a system consisting of the mentioned programs and a database geoDB, provides a framework for exploring geometrical knowledge. A GeoThms user can browse through a list of available geometric problems, their statements, illustrations, and proofs. He/she can also interactively produce new geometrical constructions, theorems, and proofs and add new results to the existing ones. GeoThms framework provides an environment suitable for new ways of studying and teaching geometry at different levels. GeoThms also provides a system for storing mathematical knowledge (in a explicit, declarative form) — not only theorem statements, but also their (automatically generated) proofs and corresponding illustrations.

1 Introduction

The axiomatic presentation of geometry fills the gap between formal logic and our spatial intuition. The study of geometry is, and will always be, very important for a mathematical practitioner. Geometry and geometrical proofs always were, and still are, exemplary mathematical contents. In history they often served for guiding development of foundations of mathematics, and today they serve in mathematical education, aimed at acquiring mathematical rigour. Computer technologies give new ways for dealing with geometry: they are used for visualisation of geometrical objects, but also for exploring/testing geometrical conjectures and, finally, for automated proving of geometrical theorems. Integrating these ways of dealing with geometry brings new forms in communicating

[⋆] This work was partially supported by programme POSC.

[⋆⋆] This work was partially supported by the programme POSC, by the Centro International de Matemática (CIM), under the programme "Research in Pairs", while visiting Coimbra University under the Coimbra Group Hospitality Scheme. Also, partially supported by Serbian Ministry of Science and Technology grant 144030.

J.M. Borwein and W.M. Farmer (Eds.): MKM 2006, LNAI 4108, pp. 280–294, 2006.

mathematical (geometrical, in this case) information — theorems, figures, and proofs. In this way, the deductive nature of geometrical conjectures and proofs is linked to the semantic nature of models of geometry and also, to human intuition and to geometrical visualisations. In order to explore such mathematical knowledge, a framework where one can browse through known results and seek for new ones is needed. In this paper, we present a tightly integrated framework that we developed, consisting of a repository of constructive geometry theorems (and proofs), a geometry theorem prover, and dynamic geometry software (as final applications). This complex framework provides an environment suitable for new ways of studying and teaching geometry at different levels and bridging spatial intuition with formal, axiomatic, Euclidean geometry proofs. The user can browse through a list of geometric problems, their statements, illustration, and proofs. He/she can also interactively use geometry software (GCLC, or Eukleides), to describe new geometric constructions (and corresponding figures), and GCLCprover to (try to) prove new conjectures, adding new results to the existing ones. In addition, this framework provides an environment for storing mathematical knowledge (in explicit, declarative way) — about geometrical constructions, proofs, and illustrations. (in this context, geometrical illustrations are not stored as images, but as their formal, explicit descriptions; while mathematical illustrations may carry information, the original message cannot always be reproduced from the illustration itself; mathematical/geometrical images stored via formal descriptions are easy to maintain, understand, modify, and process in different ways — including for producing images.)

In this paper we present our framework consisting of dynamic geometry software, automated theorem provers, and the repository of constructive geometry conjectures. All constructions and conjectures are stored in formal, declarative representation that can be used as a description of a construction, a description of a figure, and also as a formal description of a conjecture that can be attempted to be proved by the developed theorem prover.

Paper overview. Section 2, briefly discusses geometric constructions, the domain of our integrated framework; Section 3 talks about parts of our framework, with §3.1 about dynamic geometry software, especially GCLC and Eukleides, §3.2 about automated theorem proving in geometry and especially the prover GCLCprover, based on the area method, and §3.3 about geoDB, a repository of constructive geometric theorems and proofs. Section 4 is about our integrated geometry framework and its features, and Section 5 presents the whole system through a step-by-step example. Section 7 discusses further work and the issue of standards for mathematical (geometrical, in this case) contents; Section 8 draws final conclusions.

2 Geometry Constructions

For hundreds, or even thousands, of years geometric construction problems have been one of the most attractive parts of geometry and mathematics. A geometric construction is a sequence of specific, primitive construction steps. These primitive construction steps (also called *elementary constructions*) are based on using a *ruler* (or a *straightedge*[1]) and a *compass*, and they are:

- construction (with a *ruler*) of a line such that two given points belong to it;
- construction of a point which is an intersection of two lines (if such a point exists);
- construction (with a *compass*) of a circle such that its centre is one given point and such that the second given point belongs to it;
- construction of a segment connecting two points;
- construction of intersections between a given line and a given circle (if such points exist).

By using the set of primitive constructions, one can define more complex constructions (e.g., the construction of a right angle, a construction of the midpoint of a line segment, etc.).

Abstract (i.e., formal, axiomatic) nature of geometric objects have to be distinguished from their usual interpretations. A geometric construction is a procedure consisting of abstract steps and it is not a picture, but for each construction there is its counterpart in the standard Cartesian model.

Construction problems are often studied (in schools and universities) because they require rigour, but are in the same time intuitive (since they require effective procedures and since the level of abstraction is higher than the level of geometry axioms). The study of geometry and construction problems also represents a suitable field for interactive teaching supported by software tools.

3 Building Blocks

In this section, we present the building blocks of our geometry framework.

3.1 Dynamic Geometry Software, GCLC and Eukleides

Dynamic geometry software (e.g., *Cinderella*, *Geometer's Sketchpad*, *Cabri*[2]) visualise geometric objects and link formal, axiomatic nature of geometry (most often — Euclidean) with its standard models (e.g., Cartesian model) and corresponding illustrations. The common experience is that dynamic geometry software significantly help students to acquire knowledge about geometric objects.

GCLC [6,8] is a tool for teaching and studying mathematics, especially geometry and geometric constructions, and also for storing descriptions of mathematical

[1] The term "straightedge" is sometimes used instead of "ruler" in order to emphasise there are no markings which could be used to make measurements.

[2] See `http://www.cinderella.de`, `http://www.keypress.com/sketchpad/`, `http://www.cabri.com`

figures and producing digital illustrations of high quality.[3] GCLC provides support for a range of geometric constructions and isometric transformations. Although its primary initial goal is describing formal geometric constructions, GCLC also provides a support for some non-constructible objects too. In GCLC there is also support for symbolic expressions, second order curves, parametric curves, while-loops, etc. Thus, GCLC is more than a geometry tool.

GCLC is based on the idea that constructions are formal procedures, rather than drawings. Thus, in GCLC, producing mathematical illustrations is based on "describing figures" rather than of "drawing figures" (in a sense, this system is in spirit close to the LaTeX system [10], with its logical design of texts). All mathematical figures (not only geometric ones) are described in this spirit, in GC language. These descriptions directly reflect meaning of mathematical objects to be presented, and are easily understandable to mathematicians. In that sense, this language is more a high-level language than a script language.

WinGCLC is the Windows version of GCLC, with a rich graphical interface and provides a range of additional functionalities to GCLC. It supports interactive work, animations, traces, "watch window" for monitoring values of selected objects ("geometry calculator") etc. [8].

Eukleides[4] [14,16] is an Euclidean geometry drawing language. Two programs are related to it. First, `eukleides`, a compiler for typesetting geometric figures within a (La)TeX document. It can also convert such figures to EPS format or to various other vector graphic formats. Second, `xeukleides`, a GUI front-end for creating interactive geometric figures. This program can also be used for editing and tuning Eukleides code. Eukleides, like GCLC has been designed to be close to the traditional language of elementary Euclidean geometry. In many cases, it is possible to completely avoid the use of Cartesian coordinates.

We have developed a tool `euktogclcprover`, that converts Eukleides files to GCLCprover files, enabling the prover to be used with geometric constructions described within Eukleides.

3.2 Automated Theorem Proving in Geometry and GCLCprover

Automated theorem proving in geometry has two major lines of research: synthetic proof style and algebraic proof style (see, for instance, [12] for a survey). Algebraic proof style methods are based on reducing geometry properties to algebraic properties expressed in terms of Cartesian coordinates. These methods

[3] GCLC package is freely available from `www.matf.bg.ac.yu/~janicic/gclc/`. The mirrored version is available from EMIS (The European Mathematical Information Service) `www.emis.de/misc/index.html`. There are command-line version and graphic interface versions of GCLC for Windows, while there is only a command-line version of GCLC for Linux.

[4] Eukleides is available from `http://www.eukleides.org`, There are versions for a number of languages. The first author of this paper is responsible for the Portuguese version of Eukleides: EukleidesPT is available from `http://gentzen.mat.uc.pt/~EukleidesPT/`

are usually very efficient, but the proofs they produce do not reflect the geometry nature of the problem and they give only a yes/no conclusion. Synthetic methods attempt to automate traditional geometry proof methods that produce human-readable proofs.

We have extended GCLC, with a theorem prover that allows formal deductive reasoning about constructions made in the (main) drawing module. The built-in prover, GCLCprover, is based on the area method [3,4,13]. It produces proofs that are human-readable, and with a clear justification for every proof step. The prover can be used in conjunction with other dynamic geometry software, which demonstrate the flexibility of the developed deduction module.

The area method is a synthetic method providing traditional (not coordinate-based), human-readable proofs. The proofs are expressed in terms of higher-level geometric lemmas and expression simplifications. The main idea of the method is to express hypotheses of a theorem using a set of constructive statements, each of them introducing a new point, and to express a conclusion by an equality of expressions in some geometric quantities (e.g., signed area of a triangle), without referring to Cartesian coordinates. The proof is then based on eliminating (in reverse order) the points introduced before, using for that purpose a set of appropriate lemmas. After eliminating all introduced points, the current goal becomes an equality between two expressions in quantities over independent points. If it is trivially true, then the original conjecture was proved valid, if it is trivially false, then the conjecture was proved invalid, otherwise, the conjecture has been neither proved nor disproved. In all stages, different simplifications are applied to the current goal. The method does not have any branching, which makes it very efficient for many non-trivial geometry theorems. The method can transform a conjecture given as a geometry quantity of degree d, involving n constructed points, to a rational expression not involving constructed points, and with a degree at most $5d3^{5n}$ [3].

The area method is applicable to a wide range of constructions and a wide range of geometric conjectures. For this fragment of geometry, the area method gives a decision procedure: a terminating, sound, and complete procedure, i.e., a procedure that can prove any geometry theorem involving only points introduced by using supported constructions, and expressed in terms of geometric quantities. For details of the method, correctness proofs for all simplification steps, and for details about our implementation see [15].

GCLCprover is tightly integrated with geometry software. This means that one can use the prover to reason about a (say) GCLC construction (i.e., about objects introduced in it), without changing and adapting it for the deduction process — the user only needs to add the conclusion he/she wants to prove. The geometric constructions made within GCLC are internally transformed into primitive constructions of the area method, and in some cases, some auxiliary points are introduced.

GCLCprover was implemented in C++ (as GCLC) and is very efficient. The theorem prover produces proofs in LaTeX form and a report about the proving process: whether the conjecture was proved or disproved, CPU time spent, and number of proof steps performed. For each proof step, there is a justification, and (optionally) its semantics counterpart (the semantic information is not used in the proof itself, but it can be used for testing conjectures). The prover can prove many complex geometric problems in milliseconds, producing readable proofs.

3.3 The geoDB Database

The geoDB database gives support to the other programs, keeping the information, and allowing for its fast retrieving whenever necessary. The database is organised in the following form (see the entity-relationship diagram for details – Figure 1):

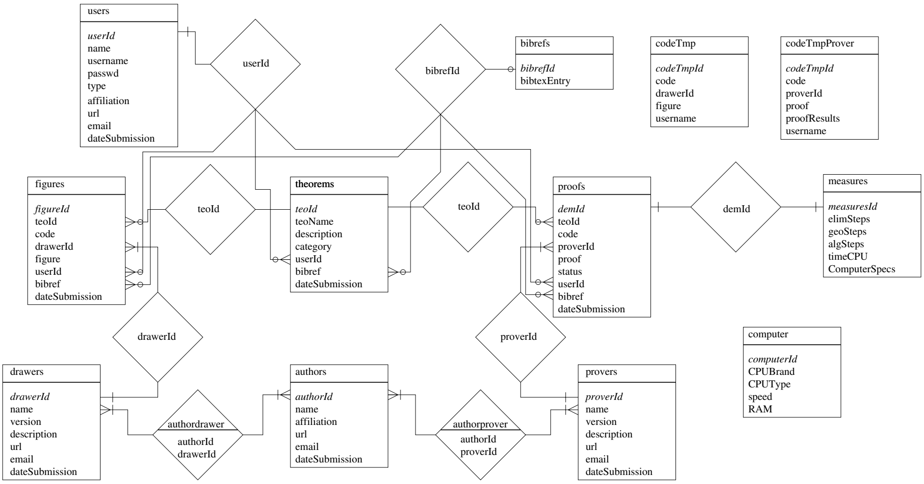


Fig. 1. geoDB — Entity-relationship diagram

Theorems — statements of theorems, in natural-language form, formatted in LaTeX;

Figures — descriptions of geometrical constructions, in DGS's code (GCLC, Eukleides, or other drawing software), they can be used for producing the corresponding figures;

Proofs — geometrical constructions with conjectures in ATP's code (GCLC prover, or other provers), they are used for producing the corresponding proofs;

A geometric theorem can have different figures and/or proofs, made by different software, made by different users. This fact is expressed by the 1 to n relationships between the entities "theorems" and the other two entities (see Figure 1).

The database also has the following auxiliary entities:

Bibrefs — bibliographic references, in BIBTEX format;
Drawers & Provers — information about the programs whose code is kept in the database, and with which the user can interact;
Authors — information about the authors of the programs;
Users — information about registered users.
Computer — information about the computer used as the test bench.

The `codeTmp` and `codeTmpProver` tables are used to store temporary information, deleted after each session, for the interactive section of GeoThms.

The geoDB database is implemented in MySQL, with InnoDB transition safe type of tables, and with foreign key constraints.

4 The Framework

GeoThms[5], is a framework that links dynamic geometry software (GCLC, Eukleides), geometry theorem provers (GCLCprover), and a repository of geometry problems (geoDB) (see Figure 2).

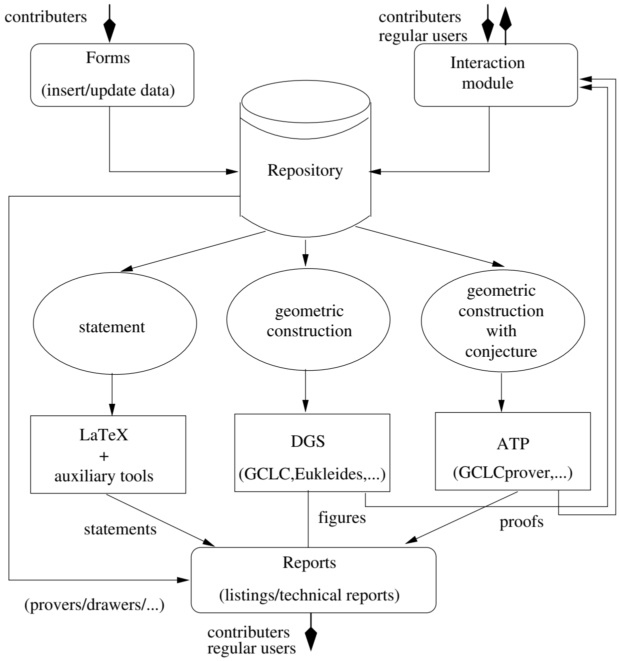
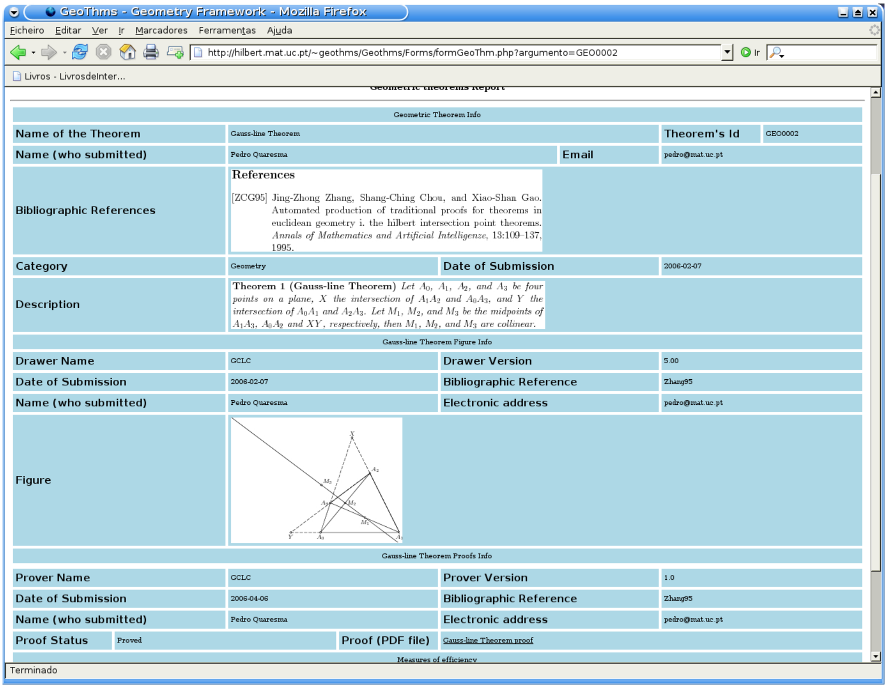


Fig. 2. The GeoThms framework

[5] GeoThms is accessible from `http://hilbert.mat.uc.pt/~geothms`


GeoThms - Geometry Framework - Mozilla Firefox
Ficheiro Editar Ver Ir Marcadores Ferramentas Ajuda
http://hilbert.mat.uc.pt/~geothms/Geothms/Forms/formGeoThm.php?argumento=GEO0002
Ir
Livros - LivrosdeInter...
X
A_2
M_3
A_3
M_2
M_1
Y
A_0
A_1
Terminado

Geometric theorems Report

Geometric Theorem Info

Name of the Theorem	Gauss-line Theorem	Theorem's Id	GEO0002
Name (who submitted)	Pedro Quaresma	Email	pedro@mat.uc.pt
Bibliographic References	References [ZCG95] Jing-Zhong Zhang, Shang-Ching Chou, and Xiao-Shan Gao. Automated production of traditional proofs for theorems in euclidean geometry i. the hilbert intersection point theorems. *Annals of Mathematics and Artificial Intelligenze*, 13:109–137, 1995.		
Category	Geometry	Date of Submission	2006-02-07
Description	**Theorem 1 (Gauss-line Theorem)** *Let A_0, A_1, A_2, and A_3 be four points on a plane, X the intersection of A_1A_2 and A_0A_3, and Y the intersection of A_0A_1 and A_2A_3. Let M_1, M_2, and M_3 be the midpoints of A_1A_3, A_0A_2 and XY, respectively, then M_1, M_2, and M_3 are collinear.*		

Gauss-line Theorem Figure Info

Drawer Name	GCLC	Drawer Version	5.00
Date of Submission	2006-02-07	Bibliographic Reference	Zhang95
Name (who submitted)	Pedro Quaresma	Electronic address	pedro@mat.uc.pt
Figure			

Gauss-line Theorem Proofs Info

Prover Name	GCLC	Prover Version	1.0
Date of Submission	2006-04-06	Bibliographic Reference	Zhang95
Name (who submitted)	Pedro Quaresma	Electronic address	pedro@mat.uc.pt
Proof Status	Proved	Proof (PDF file)	Gauss-line Theorem proof

Measures of efficiency

Fig. 3. GeoThms screenshot - Theorem Report

GeoThms provides a Web workbench in the field of constructive problems in Euclidean geometry. Its tight integration with dynamic geometry software and automatic theorem provers (GCLC, Eukleides, and GCLCprover, for the moment) and its repository of theorems, figures and proofs, give the user the possibility to easily browse through the list of geometric problems, their statements, illustrations and proofs, and also to interactively use the drawing and proving programs (See Figure 3).

The structure of the web interface has two main levels of interaction (see Figure 4). The entry level, accessible to all web-users, has some basic information about GeoThms, including documents about the GeoThms Framework, and about the GCLCprover and the Area Method. This level offers the possibility of registration to anyone interested in using GeoThms, and it gives access to the other levels. A (registered) regular user has access to a second level where he/she can browse the data from the database (in a formatted, or in a plain textual form) and use the drawing/proof programs in an interactive way.

A regular user can apply to the status of *contributer* in which case he/she will have the possibility to insert new data, and/or to update the data he/she had inserted previously.

Constructions are described and stored in declarative languages of dynamic geometry software such as GCLC and Eukleides. Figures are generated directly on the basis of descriptions of constructions, by GCLC and Eukleides and stored

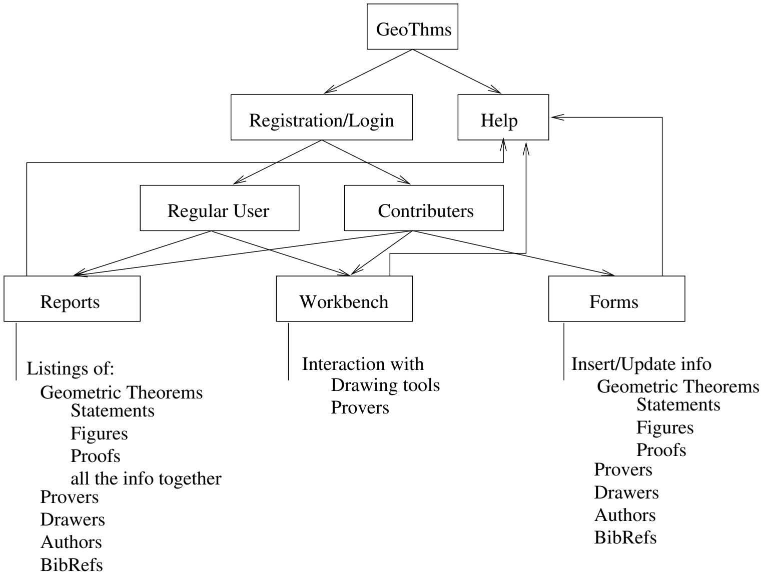


Fig. 4. GeoThms — Web Interface

as JPEG files. Conjectures are described and stored in a a form that extends descriptions of constructions. Descriptions of conjectures is used (directly or via a converter) by GCLCprover. Proofs are generated by GCLCprover and stored as PDF files (after beeing processed by LaTeX, using a specific layout, `gclc_proof` style).

The framework can be simply augmented by other dynamic geometry software and other geometry theorem provers.

GeoThms gives the user a complex framework suitable for new ways of communicating mathematical (geometrical, in this case) knowledge. It provides an open system where one can learn from the existing knowledge base and seek for new results. GeoThms also provides a system for storing mathematical knowledge (in a strict, declarative form) — not only theorem statements, but also their (automatically generated) proofs and corresponding figures, i.e., visualisations.

5 Geothms by Example

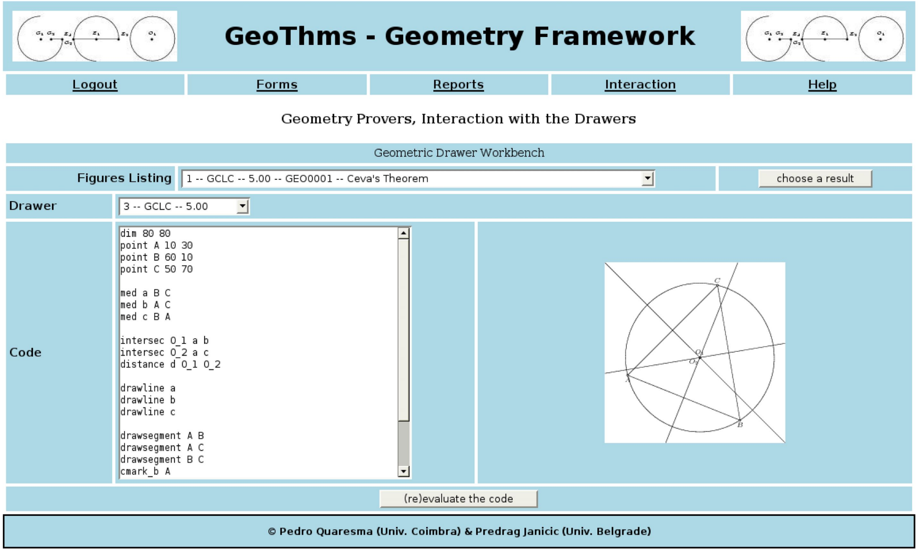
In this section we describe GeoThms framework through a step-by-step example.

The circumcircle of a triangle is the unique circle on which all its three vertices lie. Its center can be constructed as the intersection of any two out of the three perpendicular side bisectors. The crucial point is: do the three perpendicular side bisectors meet in a single point?

We can use GeoThms to answer this question, by describing the construction and proving the property. Using the interactive part of GeoThms, a user can begin by the construction, proceed attempting to prove the conjecture and, if all went as expected, insert all this information, along with the new result statement, in the database.

5.1 Describing the Construction

The constructive specification of the figure has to define: three points A, B, C (the vertices of the triangle); three side bisectors; points O_1 and O_2 defined as the pairwise intersections of these lines. Apart from the construction steps, the figure description also provides the coordinates of the points A, B, and C, and all the "drawing" commands. Note that all these commands are irrelevant for the theorem prover, but are relevant for producing figures (see Figure 5).


GeoThms - Geometry Framework
Logout
Forms
Reports
Interaction
Help
Geometry Provers, Interaction with the Drawers
Geometric Drawer Workbench
choose a result
(re)evaluate the code
C
O1
O2
A
B

Figures Listing: 1 -- GCLC -- 5.00 -- GEO0001 -- Ceva's Theorem

Drawer: 3 -- GCLC -- 5.00

Code:

```
dim 80 80
point A 10 30
point B 60 10
point C 50 70

med a B C
med b A C
med c B A

intersec O_1 a b
intersec O_2 a c
distance d O_1 O_2

drawline a
drawline b
drawline c

drawsegment A B
drawsegment A C
drawsegment B C
cmark_b A
```

© Pedro Quaresma (Univ. Coimbra) & Predrag Janicic (Univ. Belgrade)

Fig. 5. Circumcircle of a triangle — Interaction with the DGS

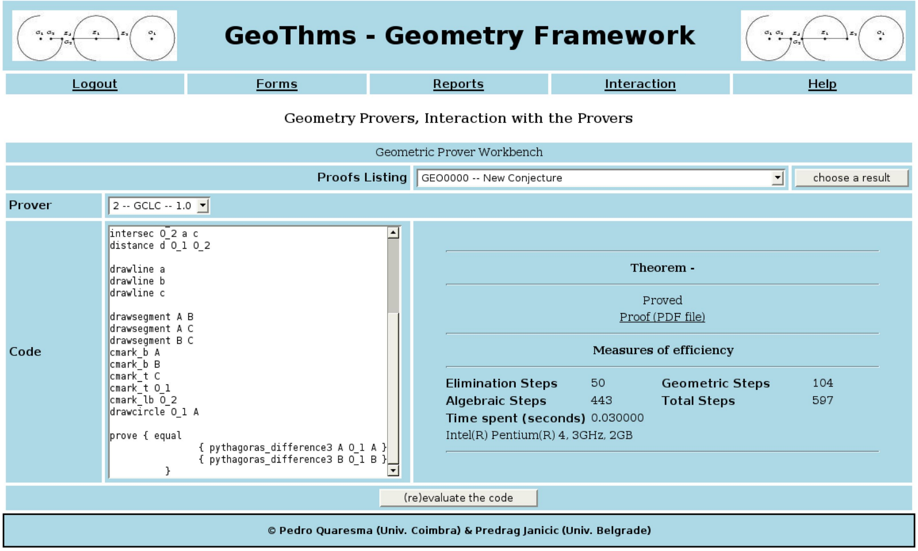
The construction shown in Figure 5 was made using GCLC, but the user can also use Eukleides for describing the construction, by instructions very similar to the given ones.

5.2 Testing the Conjecture

Having described the construction of the figure, now we have to add the conjecture. The property to be proved can be expressed in the following way: the points O_1 and O_2 are identical. The user must express this condition within the command `prove`, and using the geometrical quantities supported by the area method, in this case — via the *Pythagoras difference* geometric quantity (for more details, see [15]).

All the commands used in the construction of the figure are internally (within the prover) transformed into primitive constructions of the area method. The

GCLC's code can be submitted to GCLCprover without modifications, the Eukleides' code needs to be converted with the `euktogclcprover` tool. As shown in Figure 6, the proof status and the measures of efficiency are accessible, the proof is given as a PDF file. Figure 7 shows the last steps of the proof made by GCLCprover. The proof was generated in 0.03 seconds.[6]



Geometry Provers, Interaction with the Provers

Geometric Prover Workbench

Proofs Listing: GEO0000 -- New Conjecture

Prover: 2 -- GCLC -- 1.0

Code:

```
intersec O_2 a c
distance d O_1 O_2

drawline a
drawline b
drawline c

drawsegment A B
drawsegment A C
drawsegment B C
cmark_b A
cmark_b B
cmark_t C
cmark_t O_1
cmark_lb O_2
drawcircle O_1 A

prove { equal
                { pythagoras_difference3 A O_1 A }
                { pythagoras_difference3 B O_1 B }
        }
```

Theorem -

Proved
Proof (PDF file)

Measures of efficiency

```
Elimination Steps      50        Geometric Steps   104
Algebraic Steps        443       Total Steps       597
Time spent (seconds)   0.030000
Intel(R) Pentium(R) 4, 3GHz, 2GB
```

(re)evaluate the code
© Pedro Quaresma (Univ. Coimbra) & Predrag Janicic (Univ. Belgrade)

Fig. 6. Circumcircle of a triangle — Interaction with the ATP

(113) $(0.062500 \cdot (P_{CBC} \cdot S_{BAC})) = \left(\frac{1}{4} \cdot \left(P_{CBM_a^0} \cdot S_{BAM_a^0}\right)\right)$, by algebraic simplifications

(114) $(0.062500 \cdot (P_{CBC} \cdot S_{BAC})) = \left(\frac{1}{4} \cdot \left(\left(P_{CBB} + \left(\frac{1}{2} \cdot (P_{CBC} + (-1 \cdot P_{CBB}))\right)\right) \cdot S_{BAM_a^0}\right)\right)$, by Lemma 29 (point M_a^0 eliminated)

(115) $(0.062500 \cdot (P_{CBC} \cdot S_{BAC})) = \left(\frac{1}{4} \cdot \left(\left(0 + \left(\frac{1}{2} \cdot (P_{CBC} + (-1 \cdot 0))\right)\right) \cdot S_{BAM_a^0}\right)\right)$, by geometric simplifications

(116) $(0.062500 \cdot S_{BAC}) = \left(\frac{1}{8} \cdot S_{BAM_a^0}\right)$, by algebraic simplifications

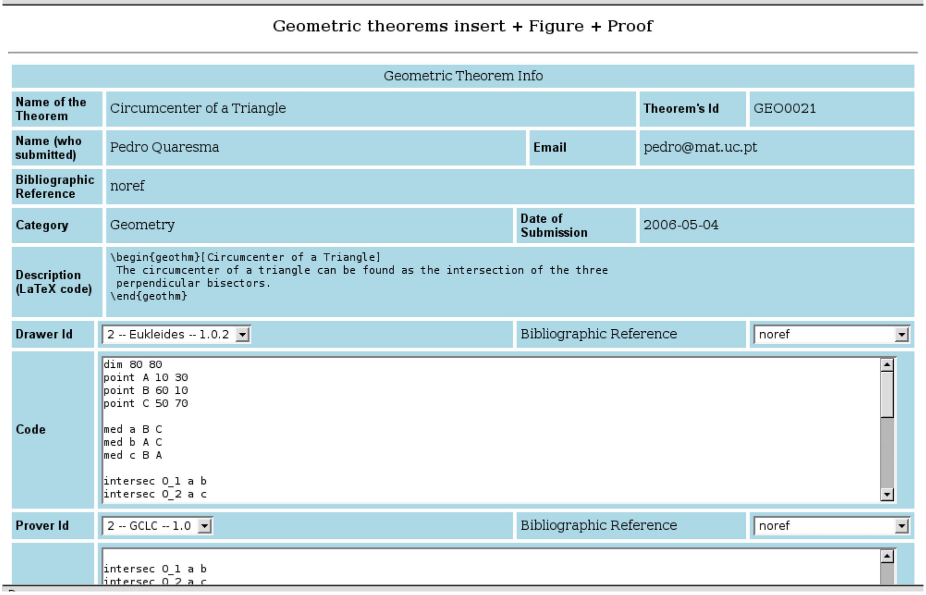
(117) $(0.062500 \cdot S_{BAC}) = \left(\frac{1}{8} \cdot \left(S_{BAB} + \left(\frac{1}{2} \cdot (S_{BAC} + (-1 \cdot S_{BAB}))\right)\right)\right)$, by Lemma 29 (point M_a^0 eliminated)

(118) $(0.062500 \cdot S_{BAC}) = \left(\frac{1}{8} \cdot \left(0 + \left(\frac{1}{2} \cdot (S_{BAC} + (-1 \cdot 0))\right)\right)\right)$, by geometric simplifications

(119) $0 = 0$, by algebraic simplifications

Fig. 7. Last steps of the proof of the Circumcircle theorem

[6] Many complex geometry theorems can be proved by the system in only milliseconds. For instance: theorems by Ceva (0.001s), Gauss (0.029s), Thales (0.001s), Menelaus (0.002s), Pappus' Hexagon (0.040s), midpoint theorem (0.002s), ratio of areas of parallelograms (0.190s), etc.



Geometric theorems insert + Figure + Proof

Geometric Theorem Info

Name of the Theorem	Circumcenter of a Triangle	Theorem's Id	GEO0021
Name (who submitted)	Pedro Quaresma	Email	pedro@mat.uc.pt
Bibliographic Reference	noref		
Category	Geometry	Date of Submission	2006-05-04

Description (LaTeX code)

```
\begin{geothm}[Circumcenter of a Triangle]
 The circumcenter of a triangle can be found as the intersection of the three
 perpendicular bisectors.
\end{geothm}
```

Drawer Id	2 -- Eukleides -- 1.0.2	Bibliographic Reference	noref

Code

```
dim 80 80
point A 10 30
point B 60 10
point C 50 70

med a B C
med b A C
med c B A

intersec O_1 a b
intersec O_2 a c
```

Prover Id	2 -- GCLC -- 1.0	Bibliographic Reference	noref

```

intersec O_1 a b
intersec O_2 a c
```

Fig. 8. Circumcircle of a triangle — Insertion Form

5.3 Inserting a Result in the Database

The user (with the status of contributer) can select the "Forms" section in order to insert a statement for the new result and the corresponding figure and proof (see Figure 8). The statement is kept in the database in LaTeX format and in declarative ATP's code[7], the figure description is kept in DGS's code and also in JPEG format, the proof is kept in PDF format. For these last two, this means that the DGSs and ATPs are called before the actual insertion is made, validating the code. The JPEG and PDF files are kept in order to avoid the re-evaluation of the code each time a user wants to consult the database.

After inserting, this new result became available for all users, not only in the "Reports" section, but also in the "Interaction" section. In all cases the user has access to the code allowing him/her to use it for inclusion in mathematical texts, for testing further results, etc. (see Figure 3).

6 Related Systems

There are, to our knowledge, the following systems, similar to the system presented in this paper: *Geometry Expert* (GEX)[8]; *Ludi Geometrici* (geometriagon)[9];

[7] ATP's code share most of DGS's code, the only difference is the conjecture itself, which does not appear in DGS's code.

[8] GEX tool: `http://woody.cs.wichita.edu/gex/7-10/gex.html`

[9] geometriagon: http://www.polarprof.net/geometriagon/

Cinderella [9]; *Discover* [2]; and *GeoView* [1]. The GEX program (new version currently under development) is a DGS with a web interface; it incorporates an ATP, but, unlike GCLCprover, the GEX prover implements an algebraic proof method, and the user can only select one from a limited number of conclusions (e.g., are three selected point collinear?). The GEX tool does not have an accessible database of problems, and does not provide a formatted output for images and proofs. The geometriagon has an already vast repository of problems in the area of classical constructive (ruler and compass only) Euclidean geometry, a registered user can access/edit all problems and solutions. It does not provide an ATP. The user can perform only valid steps in the construction, using only a limited set of tools, and in this way the system is capable to recognise whenever a user has reach a solution of a problem. The geometriagon does not provide any formatted output. Cinderella uses randomise theorem checking to analyse its users actions and to react properly; it does not provide a proof for a given construction in any form. Discover is a DGS that can communicate with Mathematica[10], using the symbolic capabilities of the latter to implement the Gröebner bases method, hence, it is necessary to translate the geometric construction to an algebraic form and back, from the conclusion in algebraic form to its geometric counterpart. No proof in any form is provided. The Geoview software combines the Coq[11] ATP and the GeoplanJ[12] DGS into a system where it is possible to edit statements of geometrical theorems, and to visualise the statement using the DGS. The proofs are not accessible. None of this last three systems have a database of problems easily accessible to its users.

7 Further Work

Automated theorem provers, applications, and repository of problems are often developed separately. In some cases, joint efforts of numbers of researchers led to standards such as DIMACS (for propositional logic) [5] and SMT (for satisfiability modulo theory) [17] and repositories of problems such as SAT-lib (for propositional logic) [7], TPTP (for predicate logic) [18], SMT-lib (for satisfiability modulo theory) [17] etc. Such efforts, standards, and libraries are fruitful for easier exchange of problems, ideas, and even program code. However, this is often very demanding and there are no many systems smoothly integrating libraries of problems, theorem provers, and real-world applications. In the previous sections, we presented a tightly integrated system consisting of a library of geometry construction problems, dynamic geometry software, and a geometry theorem prover. This system can serve as a good starting point for defining open repository of geometry problems. Currently, geometry conjectures are stored within the description of constructions, in GCLC or in Eukleides language (with additional, natural-language descriptions). This representation is formal, declarative and precise. The strict description of the notion of geometrical constructions and

[10] http://www.wolfram.com

[11] http://coq.inria.fr/

[12] http://erathostene.math.univ-montp2.fr/SPIP/De-Geoplan-Geospace-a-GeoplanJ

also our experience with GCLC, Eukleides and other similar programs show that different languages are very close to each other (primarily dealing with elementary constructions and isometric transformations, but also with dealing with scaling of figures, labelling components of figures, etc.). We believe that descriptions in all these languages can be normalised, i.e., transformed to a single description. We have already developed the converter from Eukleides to GCLC, but similar converters can be made for other pairs of languages. We propose defining such a normal, referent form, and making a repository usable by all geometry programs. Such language should have a XML version (in a similar way as for SMT-LIB [11]), closer to wide relevant mathematical initiatives such as MathML.[13] That way, it would be possible to store descriptions of constructions in a quality form that provides both formal mathematical contents and visual contents. Moreover, the generic XML validation mechanism could be used for verifying whether a given construction is legal.

8 Conclusions

In this paper we presented our framework GeoThms consisting of dynamic geometry software GCLC and Eukleides, automated theorem prover GCLCprover, and the repository of constructive geometry conjectures geoDB, all accessible through a web interface.

This complex framework provides an environment suitable for new ways of studying and teaching geometry at different levels. In addition, this framework provides an environment for storing mathematical knowledge (in explicit, declarative way) — about geometrical constructions, proofs, and illustrations. We hope that support from interested parties will make GeoThms growing and widely used repository.

We are planning to link additional geometry programs and additional theorem provers to our framework and to further develop the web interface. We are also considering developing a referent geometry language that can be linked to all geometry programs dealing with Euclidean constructions.

References

1. Yves Bertot, Frédéric Guilhot, and Locï Pottier. Visualizing geometrical statements with geoview, 2004. http://www-sop.inria.fr/lemme/geoview/geoview.ps.
2. Francisco Botana and José L. Valcarce. A dynamic-symbolic interface for geometric theorem discovery. *Computers and Education*, 38:21–35, 2002.
3. C. C. Chou, OU X. S. Gao, and J. Z. Zhang. Automated production of traditional proofs for constructive geometry theorems. In *Eighth Annual IEEE Symposium on Logic in Computer Science*, 1993.
4. Shang-Ching Chou, Xiao-Shan Gao, and Jing-Zhong Zhang. Automated generation of readable proofs with geometric invariants, I. multiple and shortest proof generation. *Journal of Automated Reasoning*, 17:325–347, 1996.

[13] MathML is the Mathematical Markup Language. It is an XML application for describing mathematical notation and capturing both its structure and content.

5. DIMACS. Satisfiability suggested format. ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/doc/satformat.tex.
6. Mirjana Djorić and Predrag Janičić. Constructions, instructions, interactions. *Teaching Mathematics and its Applications*, 23(2):69–88, 2004.
7. Holger H. Hoos and Thomas Stützle. Satlib: An online resource for research on sat. In *Proceedings of SAT 2000*. IOS Press, 2000. SATLIB is available online at www.satlib.org.
8. Predrag Janičić and Ivan Trajković. WinGCLC — a Workbench for Formally Describing Figures. In *Proceedings of the 18th Spring Conference on Computer Graphics (SCCG 2003)*, pages 251–256, Budmerice, Slovakia, April, 24-26 2003. ACM Press, New York, USA.
9. Ulrich Kortenkamp and Jürgen Richter-Gebert. Using automatic theorem proving to improve the usability of geometry software. In *Procedings of the Mathematical User-Interfaces Workshop 2004*, 2004.
10. Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley Publishing Company, 2nd edition, 1994.
11. Filip Marić and Predrag Janičić. SMT-LIB in XML clothes. In *Proceedings of Workshop Pragmatics of Decision Procedures in Automated Reasoning (PDPAR-2004)*, 2004.
12. Noboru Matsuda and Kurt Vanlehn. Gramy: A geometry theorem prover capable of construction. *Journal of Automated Reasoning*, 32:3–33, 2004.
13. Julien Narboux. A decision procedure for geometry in coq. In *Proceedings TPHOLS 2004*, volume 3223 of *Lecture Notes in Computer Science*. Springer, 2004.
14. Christiam Obrecht. Eukleides. http://www.eukleides.org/.
15. Pedro Quaresma and Predrag Janičić. Framework for constructive geometry (based on the area method). Technical Report 2006/001, Centre for Informatics and Systems of the University of Coimbra, 2006.
16. Pedro Quaresma and Ana Pereira. Visualização de construções geométricas. *Gazeta de Matemática*. To appear.
17. Silvio Ranise and Cesare Tinelli. The SMT-LIB Format: An Initial Proposal. 2003. on-line at: http://goedel.cs.uiowa.edu/smt-lib/.
18. Geoff Sutcliffe. The tptp problem library. `http://www.cs.miami.edu/~tptp/TPTP/TR/TPTPTR.shtml`.

Author Index

Aberdein, Andrew 208
Aboul-Hosn, Kamal 54
Autexier, Serge 67, 94

Baaz, Matthias 82
Ballarin, Clemens 31
Balys, Vaidas 152
Bancerek, Grzegorz 266
Brown, Chad E. 110

Cairns, Paul 237
Chaitin, Gregory J. 1
Colton, Simon 237

Dietrich, Dominik 94

Gross, Christian 251

Hazewinkel, Michiel 152
Hetzl, Stefan 82
Hilf, Eberhard R. 165

Janičić, Predrag 280

Kanahori, Toshihiro 124
Kerber, Manfred 44
Kohlhase, Andrea 179
Kohlhase, Michael 165, 179

Leitsch, Alexander 82
Libbrecht, Paul 251

Naylor, William 222

Padget, Julian 222
Padovani, Luca 194

Quaresma, Pedro 280

Raja, Amar 139
Rayner, Matthew 139
Richter, Clemens 82
Rudzkis, Rimantas 152

Sacerdoti-Coen, Claudio 67
Sexton, Alan 124, 139
Sorge, Volker 124, 139, 237
Spohr, Hendrik 82
Stamerjohanns, Heinrich 165
Suzuki, Masakazu 124

Torres, Pedro 237

Wenzel, Makarius 17

Youssef, Abdou 2

Zacchiroli, Stefano 194