Serge Autexier Jacques Calmet
David Delahaye Patrick D.F. Ion
Laurence Rideau Renaud Rioboo
Alan P. Sexton (Eds.)

LNAI 6167

Intelligent Computer Mathematics

10th International Conference, AISC 2010
17th Symposium, Calculemus 2010
and 9th International Conference, MKM 2010
Paris, France, July 2010, Proceedings

Springer

Lecture Notes in Artificial Intelligence 6167

Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

Serge Autexier Jacques Calmet
David Delahaye Patrick D.F. Ion
Laurence Rideau Renaud Rioboo
Alan P. Sexton (Eds.)

Intelligent Computer Mathematics

10th International Conference, AISC 2010
17th Symposium, Calculemus 2010
and 9th International Conference, MKM 2010
Paris, France, July 5-10, 2010
Proceedings

Springer

Series Editors

Randy Goebel, University of Alberta, Edmonton, Canada
Jörg Siekmann, University of Saarland, Saarbrücken, Germany
Wolfgang Wahlster, DFKI and University of Saarland, Saarbrücken, Germany

Volume Editors

Serge Autexier, DFKI Bremen, Germany
E-mail: serge.autexier@dfki.de

Jacques Calmet, Karlsruhe Institute of Technology, Germany
E-mail: calmet@ira.uka.de

David Delahaye, Conservatoire National des Arts et Métiers, Paris, France
E-mail: David.Delahaye@cnam.fr

Patrick D.F. Ion, AMS, Ann Arbor, USA
E-mail: ion@ams.org

Laurence Rideau, INRIA Sophia-Antipolis, France
E-mail: Laurence.Rideau@inria.fr

Renaud Rioboo, ENSIIE, Evry, France
E-mail: Renaud.Rioboo@ensiie.fr

Alan P. Sexton, University of Birmingham, UK
E-mail: a.p.sexton@cs.bham.ac.uk

Library of Congress Control Number: 2010929251

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

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN	0302-9743
ISBN-10	3-642-14127-7 Springer Berlin Heidelberg New York
ISBN-13	978-3-642-14127-0 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

This volume contains the collected contributions of three conferences, AISC 2010, Calculemus 2010 and MKM 2010. AISC 2010 was the 10th International Conference on Artificial Intelligence and symbolic computation. Its area of concern is the use of AI techniques within symbolic computation as well as the application of symbolic computation to AI problem solving. Calculemus 2010 was the 17th Symposium on the Integration of Symbolic Computation and Mechanised Reasoning, dedicated to the combination of computer algebra systems and automated deduction systems. MKM 2010 was the 9th International Conference on Mathematical Knowledge Management, an emerging interdisciplinary field of research in the intersection of mathematics, computer science, library science, and scientific publishing. All three conferences are thus concerned with providing intelligent computer mathematics. Although the conferences have separate communities and separate foci, there is a significant overlap of interest in building systems for intelligent computer mathematics.

As in 2008 and 2009, the three events were colocated. In 2010 this was at the Conservatoire National des Arts et Métiers (CNAM), Paris, France, under the umbrella of the Conferences on Intelligent Computer Mathematics (CICM 2010), organized by Renaud Rioboo and Laurence Rideau. This collocation is intended to counteract the tendency towards fragmentation of communities working on different aspects of various independent branches of our general field; traditional branches (e.g., computer algebra, theorem proving and artificial intelligence in general), as well as newly emerging ones (on user interfaces, knowledge management, theory exploration, etc.). This also facilitates the development of systems for intelligent computer mathematics that will be routinely used by mathematicians, computer scientists and engineers in their every-day work.

While the proceedings are shared, the submission process was separate. The responsibility for acceptance/rejection rests completely with the three separate Program Committees. In total, 25 papers were submitted to AISC. For each paper there were at least four reviews, and on average 5.3 reviews by Program Committee members, out of which 9 papers were accepted for publication in these proceedings. Calculemus received 14 submissions. For each paper, there were at least 3 reviews, and 3.2 reviews on average. After discussion, 7 papers were accepted for publication in these proceedings and 3 papers were selected to appear in the track of emerging trends (not published in this volume). MKM received 27 submissions. For each paper there were at least 3 reviews, and 3.2 reviews on average. After discussion, 16 papers were accepted for publication.

We were very pleased that the keynote talks of AISC, Calculemus, MKM and associated workshops were presented by an array of such highly distinguished speakers. We have included abstracts or full papers for most of these talks in the proceedings. The keynote speakers were:

Andrea Asperti	University of Bologna, Italy
Jacques Calmet	Karlsruhe Institute of Technology, Germany
John Campbell	University College London, UK
Jacques Carette	McMaster University, Canada
Pierre Cartier	Institut des Hautes Études Scientifiques, France
James H. Davenport	University of Bath, UK
Bruno Salvy	INRIA Rocquencourt, France
Masakazu Suzuki	Kyushu University, Japan
Doron Zeilberger	Rutgers University, USA

In the preparation of these proceedings and in managing the whole discussion process, Andrei Voronkov's EasyChair conference management system proved itself an excellent tool.

May 2010

Serge Autexier
Jacques Calmet
David Delahaye
Patrick Ion
Renaud Rioboo
Alan Sexton

AISC, Calculemus and MKM Organization

Conference Chairs	Laurence Rideau, Renaud Rioboo
Local Organization	David Delahaye, Catherine Dubois, Véronique Viguié-Donzeau-Gouge
Website	http://cicm2010.cnam.fr/

Sponsors

- Conservatoire National des Arts et Métiers http://www.cnam.fr
- Institut National de Recherche en Informatique et Automatique http://www.inria.fr
- École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise http://www.ensiie.fr
- GDR Génie de la programmation et du logiciel http://gdr-gpl.cnrs.fr
- Texas Instruments http://www.ti.com
- ACM Special Interest Group for Symbolic and Algebraic Manipulation http://www.sigsam.org

AISC 2010 Organization

Program Chair

Serge Autexier	DFKI Bremen, Germany

Steering Committee

Serge Autexier	DFKI Bremen, Germany
Jacques Calmet	Karlsruhe Institute of Technology, Germany
John Campbell	University College London, UK
Eugenio Roanes-Lozano	Universidad Complutense de Madrid, Spain
Volker Sorge	University of Birmingham, UK

Program Committee

Alessandro Armando	University of Genova, Italy
Franz Baader	TU Dresden, Germany
Christoph Benzmüller	Articulate Software, USA
Russell Bradford	University of Bath, UK
Jacques Calmet	University of Karlsruhe, Germany
John Campbell	University College London, UK

Jacques Carette	McMaster University, Canada
Arjeh Cohen	Eindhoven University of Technology, The Netherlands
Simon Colton	Imperial College London, UK
Timothy Daly	Carnegie Mellon, USA
Lucas Dixon	University of Edinburgh, UK
Bill Farmer	McMaster University, UK
Martin Charles Golumbic	University of Haifa, Israel
Tetsuo Ida	University of Tsukuba, Japan
Tom Kelsey	University of St. Andrews, UK
Claude Kirchner	INRIA Bordeaux, France
George Labahn	University of Waterloo, Canada
Petr Lisonek	Simon Fraser University, Canada
Ralf Möller	TU Harburg, Germany
Bijan Parsia	University of Manchester, UK
Renaud Rioboo	ENSIIE, France
Eugenio Roanes-Lozano	Universidad Complutense de Madrid, Spain
Chung-chieh Shan	Rutgers, USA
Jörg Siekmann	Universität des Saarlandes, DFKI Saarbrücken, Germany
Elena Smirnova	Texas Instruments, USA
Volker Sorge	University of Birmingham, UK
Toby Walsh	NICTA and UNSW, Australia
Dongming Wang	Beihang University, China and UPMC-CNRS, France
Stephen M. Watt	University of Western Ontario, Canada
Wolfgang Windsteiger	RISC, Austria
Michael Witbrock	Cycorp, USA

External Reviewers

Paul Barry	James Brotherston	Peter Hancox
Helmut Jürgensen	Sascha Klüppelholz	François Lemaire
Paul Levy	Xiaoliang Li	Andreas Nonnengart
Özgür Özcep	Wei Pan	Adam Pease
Ron Petrick	Loic Pottier	Silvio Ranise
Marvin Schiller	Armin Straub	Nicolai Vorobjov
Karl-Heinz Zimmermann		

Calculemus 2010 Organization

Program Chairs

David Delahaye	CNAM, France
Renaud Rioboo	ENSIIE, France

Calculemus Trustees

Jacques Carette	McMaster University, Canada
Thierry Coquand	Chalmers University of Technology, Sweden
James H. Davenport	University of Bath, UK
Lucas Dixon	University of Edinburgh, UK
Paul Jackson	University of Edinburgh, UK
Marc Moreno Maza	University of Western Ontario, Canada
Renaud Rioboo	ENSIIE, France
Julio Rubio	Universidad de La Rioja, Spain
Stephen M. Watt	University of Western Ontario, Canada
Makarius Wenzel	ITU Munich, Germany
Freek Wiedijk	Radboud University Nijmegen, The Netherlands

Program Committee

Markus Aderhold	TU Darmstadt, Germany
Arjeh Cohen	Eindhoven University of Technology, The Netherlands
Thierry Coquand	Chalmers University of Technology, Sweden
James H. Davenport	University of Bath, UK
David Delahaye	CNAM, France (Chair)
Lucas Dixon	University of Edinburgh, UK
William M. Farmer	McMaster University, Canada
Temur Kutsia	RISC, Austria
Assia Mahboubi	INRIA Saclay, France
Renaud Rioboo	ENSIIE, France (Chair)
Julio Rubio	Universidad de La Rioja, Spain
Volker Sorge	University of Birmingham, UK
Stephen M. Watt	University of Western Ontario, Canada
Freek Wiedijk	Radboud University Nijmegen, The Netherlands
Wolfgang Windsteiger	RISC, Austria

External Reviewers

Catherine Dubois	Laurent Théry	Makarius Wenzel
Manuel Kauers	Pierre Weis	Russell O'Connor

MKM 2010 Organization

Program Chairs

Patrick Ion	University of Michigan, USA
Alan P. Sexton	University of Birmingham, UK

MKM Trustees

Serge Autexier	DFKI Bremen, Germany
James Davenport	University of Bath, UK
Michael Kohlhase	Jacobs University Bremen, Germany
Claudio Sacerdoti Coen	University of Bologna, Italy
Alan Sexton	University of Birmingham, UK
Petr Sojka	Masaryk University, Brno, Czech Republic
Bill Farmer (Treasurer)	McMaster University, Hamilton, Canada

Program Committee

Serge Autexier	DFKI Bremen, Germany
Laurent Bernardin	Maplesoft, Canada
Thierry Bouche	Université de Grenoble I, France
John Burns	JSTOR, USA
David Carlisle	NAG Ltd., UK
James Davenport	University of Bath, UK
Bill Farmer	McMaster University, Hamilton, Canada
Herman Geuvers	Radboud University Nijmegen, The Netherlands
Mateja Jamnik	University of Cambridge, UK
Manfred Kerber	University of Birmingham, UK
Paul Libbrecht	DFKI Saarbrücken, Germany
Bruce Miller	NIST, USA
Adam Naumowicz	University of Bialystok, Poland
Florian Rabe	Jacobs University Bremen, Germany
Eugénio A.M. Rocha	University of Aveiro, Portugal
Claudio Sacerdoti Coen	University of Bologna, Italy
Elena Smirnova	Texas Instruments, USA
Petr Sojka	Masaryk University, Brno, Czech Republic
Masakazu Suzuki	University of Kyushu, Japan
Enrico Tassi	INRIA, France
Dongming Wang	Laboratoire d'Informatique de Paris 6, France

External Reviewers

Christoph Lange	Freek Wiedijk	Josef Urban
Wolfgang Windsteiger	Xiaoyu Chen	Hans Cuypers

Workshops Organization

Digital Mathematical Libraries

Mathematical Users Intefaces

Programming Languages for Mechanzied Mathematics

Thorsten Altenkirch	University of Nottingham, UK
Serge Autexier	DFKI, Germany
David Delahaye	CNAM, Paris, France
James Davenport (PC Co-chair)	University of Bath, UK
Lucas Dixon (PC Co-chair)	University of Edinburgh, UK
Gudmund Grov	University of Edinburgh, UK
Ewen Maclean	Herriot Watt University, UK
Dale Miller	INRIA, France
Gabriel Dos Reis	Texas A&M University, USA
Carsten Schuermann	IT University of Copenhagen, Denmark
Tim Sheard	Portland State University, USA
Sergei Soloviev	IRIT, Toulouse, France
Stephen Watt	University of Western Ontario, Canada
Makarius Wenzel	ITU Munich, Germany
Freek Wiedijk	Radboud University Nijmegen, The Netherlands

Table of Contents

Contributions to AISC 2010

The Challenges of Multivalued "Functions" (Keynote Talk) 1
James H. Davenport

The Dynamic Dictionary of Mathematical Functions (Keynote Talk) ... 13
Bruno Salvy

A Revisited Perspective on Symbolic Mathematical Computing and Artificial Intelligence (Keynote Talk) 14
Jacques Calmet and John A. Campbell

I-Terms in Ordered Resolution and Superposition Calculi: Retrieving Lost Completeness 19
Hicham Bensaid, Ricardo Caferra, and Nicolas Peltier

Structured Formal Development with Quotient Types in Isabelle/HOL 34
Maksym Bortin and Christoph Lüth

Instantiation of SMT Problems Modulo Integers 49
Mnacho Echenim and Nicolas Peltier

On Krawtchouk Transforms 64
Philip Feinsilver and René Schott

A Mathematical Model of the Competition between Acquired Immunity and Virus 76
Mikhail K. Kolev

Some Notes upon "When Does $< \mathbb{T} >$ Equal Sat($\mathbb{T}$)?" 89
Yongbin Li

How to Correctly Prune Tropical Trees 101
Jean-Vincent Loddo and Luca Saiu

From Matrix Interpretations over the Rationals to Matrix Interpretations over the Naturals 116
Salvador Lucas

Automated Reasoning and Presentation Support for Formalizing Mathematics in Mizar 132
Josef Urban and Geoff Sutcliffe

Contributions to Calculemus 2010

Some Considerations on the Usability of Interactive Provers (Invited Talk) 147
Andrea Asperti and Claudio Sacerdoti Coen

Mechanized Mathematics (Invited Talk) 157
Jacques Carette

Formal Proof of SCHUR Conjugate Function 158
Franck Butelle, Florent Hivert, Micaela Mayero, and Frédéric Toumazet

Symbolic Domain Decomposition 172
Jacques Carette, Alan P. Sexton, Volker Sorge, and Stephen M. Watt

A Formal Quantifier Elimination for Algebraically Closed Fields 189
Cyril Cohen and Assia Mahboubi

Computing in Coq with Infinite Algebraic Data Structures 204
César Domínguez and Julio Rubio

Formally Verified Conditions for Regularity of Interval Matrices 219
Ioana Paşca

Reducing Expression Size Using Rule-Based Integration 234
David J. Jeffrey and Albert D. Rich

A Unified Formal Description of Arithmetic and Set Theoretical Data Types 247
Paul Tarau

Contributions to MKM 2010

Against Rigor (Invited Talk) 262
Doron Zeilberger

Smart Matching 263
Andrea Asperti and Enrico Tassi

Electronic Geometry Textbook: A Geometric Textbook Knowledge Management System 278
Xiaoyu Chen

An OpenMath Content Dictionary for Tensor Concepts 293
Joseph B. Collins

On Duplication in Mathematical Repositories 300
Adam Grabowski and Christoph Schwarzweller

Adapting Mathematical Domain Reasoners 315
Bastiaan Heeren and Johan Jeuring

Integrating Multiple Sources to Answer Questions in Algebraic Topology 331
Jónathan Heras, Vico Pascual, Ana Romero, and Julio Rubio

sTeXIDE: An Integrated Development Environment for sTeX Collections 336
Constantin Jucovschi and Michael Kohlhase

Proofs, Proofs, Proofs, and Proofs 345
Manfred Kerber

Dimensions of Formality: A Case Study for MKM in Software Engineering 355
Andrea Kohlhase, Michael Kohlhase, and Christoph Lange

Towards MKM in the Large: Modular Representation and Scalable Software Architecture 370
Michael Kohlhase, Florian Rabe, and Vyacheslav Zholudev

The Formulator MathML Editor Project: User-Friendly Authoring of Content Markup Documents 385
Andriy Kovalchuk, Vyacheslav Levitsky, Igor Samolyuk, and Valentyn Yanchuk

Notations Around the World: Census and Exploitation 398
Paul Libbrecht

Evidence Algorithm and System for Automated Deduction: A Retrospective View 411
Alexander Lyaletski and Konstantin Verchinine

On Building a Knowledge Base for Stability Theory 427
Agnieszka Rowinska-Schwarzweller and Christoph Schwarzweller

Proviola: A Tool for Proof Re-animation 440
Carst Tankink, Herman Geuvers, James McKinna, and Freek Wiedijk

A Wiki for Mizar: Motivation, Considerations, and Initial Prototype 455
Josef Urban, Jesse Alama, Piotr Rudnicki, and Herman Geuvers

Author Index 471

The Challenges of Multivalued "Functions"

James H. Davenport

Department of Computer Science
University of Bath, Bath BA2 7AY
United Kingdom
J.H.Davenport@bath.ac.uk

Abstract. Although, formally, mathematics is clear that a function is a single-valued object, mathematical practice is looser, particularly with n-th roots and various inverse functions. In this paper, we point out some of the looseness, and ask what the implications are, both for Artificial Intelligence and Symbolic Computation, of these practices. In doing so, we look at the steps necessary to convert existing texts into

(a) rigorous statements
(b) rigorously proved statements.

In particular we ask whether there might be a constant "de Bruijn factor" [18] as we make these texts more formal, and conclude that the answer depends greatly on the interpretation being placed on the symbols.

1 Introduction

The interpretation of "functions" crosses several areas of mathematics, from almost foundational issues to computer science. We endeavour to clarify some of the different uses, starting with the set-theoretic definitions.

Notation 1. $\mathbf{P}(A)$ *denotes the power set of the set A. For a function f, we write* $\mathrm{graph}(f)$ *for* $\{(x, f(x)) : x \in \mathrm{dom}(f)\}$ *and* $\mathrm{graph}(f)^T$ *for* $\{(f(x), x) : x \in \mathrm{dom}(f)\}$.

There is an implicit convention in practically all texts which mention these underspecified objects, which we will not raise further, though there is an exception which is mentioned in section 4.

Convention 1. *Where an underspecified object, such as $\sqrt{x}$, occurs more than once in a formula, the* same *value, or interpretation, is meant at each occurrence.*

For example, $\sqrt{x} \cdot \frac{1}{\sqrt{x}} = 1$ for non-zero x, even though one might think that one root might be positive and the other negative. More seriously, in the standard formula for the roots of a cubic $x^3 + bx + c$,

$$\frac{1}{6}\sqrt[3]{-108\,c + 12\sqrt{12\,b^3 + 81\,c^2}} - \frac{2b}{\sqrt[3]{-108\,c + 12\sqrt{12\,b^3 + 81\,c^2}}}, \tag{1}$$

the two occurrences of $\sqrt{12\,b^3 + 81\,c^2}$ are meant to have the same value. One could question what is meant by "same formula", and indeed the scope seems in practice to be the entire proof/example/discussion.

Notation 2. *We use the notation $A \overset{?}{=} B$ to denote what is normally given in the literature as an equality with an $=$ sign, but where one of the purposes of this paper is to question the meaning of that, very overloaded [10], symbol.*

We will often need to refer to polar coordinates for the complex plane.

Notation 3. *We write $\mathbf{C} \equiv X \underset{\text{polar}}{\times} Y$ for such a representation $z = re^{i\theta} : r \in X \wedge \theta \in Y$.*

As statements about functions, we consider the following exemplars.

$$\sqrt{z-1}\sqrt{z+1} \overset{?}{=} \sqrt{z^2-1}. \tag{2}$$

$$\sqrt{1-z}\sqrt{1+z} \overset{?}{=} \sqrt{1-z^2}. \tag{3}$$

$$\log z_1 + \log z_2 \overset{?}{=} \log z_1 z_2. \tag{4}$$

$$\arctan x + \arctan y \overset{?}{=} \arctan\left(\frac{x+y}{1-xy}\right). \tag{5}$$

For the reader unfamiliar with this topic, we should point out that, with the conventional single-valued interpretations [1], the validity of the formulae is as follows.

(2) is valid for $\Re(z) > 0$, also for $\Re(z) = 0$, $\Im(z) > 0$. It is nevertheless stated as an equation in [11, p. 297] — see section 4.
(3) is valid everywhere, despite the apparent resemblance to (2). One proof is in [6, Lemma 2].
(4) is valid with $-\pi < \arg(z_1) + \arg(z_2) \le \pi$ — this is [1, 4.1.7].
(5) is valid[1], even for *real* x, y, only when $xy < 1$. This may seem odd, since arctan has no branch cuts over the reals, but in fact there is a "branch cut at infinity", since $\lim_{x\to+\infty} \arctan x = \frac{\pi}{2}$, whereas $\lim_{x\to-\infty} \arctan x = -\frac{\pi}{2}$ and $xy = 1$ therefore falls on this cut of the right-hand side of (5).

2 The Literature

There is a curious paradox, or at least "abuse of notation" (and terminology) in mathematics to do with the word 'function'.

2.1 The (Bourbakist) Theory

In principle, (pure) mathematics is clear.

> On dit qu'un graphe F est un graphe fonctionnel si, pour tout x, il existe au plus un objet correspondant à x par F (I, p. 40). On dit qu'une correspondance $f = (F, A, B)$ est une fonction si son graphe F est un graphe fonctionnel, et si son ensemble de départ A est égal à son ensemble de définition $\mathrm{pr}_1 F$ [pr_1 is "projection on the first component"].
> [5, p. E.II.13]

[1] Only the multivalued form is given in [1], as 4.4.34.

So for Bourbaki a function includes the definition of the domain and codomain, and is *total* and *single-valued*. We will write $(F, A, B)_{\mathcal{B}}$ for such a function definition. We permit ourselves one abuse of notation, though. The natural domains of definition of analytic functions are simply connected open sets (section 2.3), generally referred to as "$\mathbf{C}^n$ with branch cuts". The table maker, or programmer (section 2.5), abhors "undefined", and extends definitions to the whole of $\mathbf{C}^n$ by making the values on the branch cut 'adhere' [3] to one side or the other, expending a definition from D, a slit version of $\mathbf{C}^n$, to the whole of $\mathbf{C}^n$. Rather than just writing $\mathbf{C}^n$ for the domain, we will explicitly write $\overline{D}$ to indicate that it is an extension of the definition with domain D.

2.2 The Multivalued View

Analysts sometimes take a completely multivalued view, as here, discussing our exemplar (4).

> The equation merely states that the sum of one of the (infinitely many) logarithms of z_1 and one of the (infinitely many) logarithms of z_2 can be found among the (infinitely many) logarithms of $z_1 z_2$, and conversely every logarithm of $z_1 z_2$ can be represented as a sum of this kind (with a suitable choice of $\log z_1$ and $\log z_2$).
>
> [7, pp. 259–260] (our notation)

Here we essentially have $\left(\mathrm{graph}(\exp)^T, \mathbf{C}, \mathbf{P}(\mathbf{C})\right)_{\mathcal{B}}$.

Related to this view is the "Riemann surface" view, which can be seen as $\left(\mathrm{graph}(\exp)^T, \mathbf{C}, \mathcal{R}_{\log z}\right)_{\mathcal{B}}$, where $\mathcal{R}_{\log z}$ signifies the Riemann surface corresponding to the function $\log z$. The Riemann surface view is discussed in [6, Section 2.4], which concludes

> Riemann surfaces are a beautiful conceptual scheme, but at the moment they are not computational schemes.

The additional structure imparted by $\mathcal{R}_{\log z}$ (over that of $\mathbf{P}(\mathbf{C})$) is undoubtedly very useful from the theoretical point of view, and provides a global setting for the next, essentially local, view.

2.3 The Branch View

Other views may also be found in the analysis literature, for example [8], where one finds the following series of statements.

p. 32. "The mapping $y \mapsto e^{iy}$ induces an isomorphism ϕ of the quotient group $\mathbf{R}/2\pi\mathbf{Z}$ on the group $\mathbf{U}$. The inverse isomorphism ϕ^{-1} of $\mathbf{U}$ on $\mathbf{R}/\pi\mathbf{Z}$ associates with any complex number u such that $|u| = 1$, a real number which is defined up to the addition of an integral multiple of 2π; this class of numbers is called the argument of u and is denoted by $\arg u$." In our notation this is $\left(\mathrm{graph}(\phi)^T, U, \mathbf{R}/2\pi\mathbf{Z}\right)_{\mathcal{B}}$.

p. 33. "We define

$$\log t = \log|t| + i \arg t, \tag{6}$$

which is a complex number defined only up to addition of an integral multiple of $2\pi i$." In our notation this is $((6), \mathbf{C}, \mathbf{C}/2\pi i\mathbf{Z})_{\mathcal{B}}$.

p. 33. "For any complex numbers t and t' both $\neq 0$ and for any values of $\log t$, $\log t'$ and $\log tt'$, we have

$$\log tt' = \log t + \log t' \pmod{2\pi i}.\text{"} \tag{7}$$

p. 33. "So far, we have not defined $\log t$ as a *function* in the proper sense of the word".

p. 61. "$\log z$ has a branch[2] in any simply connected open set which does not contain 0."

So any given branch would be $(G, D, I)_{\mathcal{B}}$, where D is a simply connected open set which does not contain 0, G is a graph obtained from one element of the graph (i.e. a pair $(z, \log(z))$ for some $z \in D$) by analytic continuation, and I is the relevant image set.

2.4 An 'Applied' View

Applied mathematics is sometimes less unambiguous.

> ...when we say that $f(x)$ is a function of x in some range of values of x we mean that for every value of x in the range one or more values of $f(x)$ exist. ... It will usually also be required that the function shall be *single-valued*, but not necessarily.
>
> [12, p. 17]

So for these authors, a function might or might not be multivalued.

2.5 The Table-Maker's Point of View

This is essentially also the computer designer's point of view, be it hardware or software. From this point of view, it is necessary to specify how to compute $f(x)$ for any given x, irrespective of any "context", and return a single value, even though, in the text accompanying the tables, we may read "only defined up to multiples of $2\pi i$" or some such.

For the purposes of this discussion, we will use the definitions from [1] (augmented by [9]), but the points apply equally to any other definition of these functions that satisfies the table-maker's criterion of unambiguity.

(2) If we substitute $z = -2$, we obtain $\sqrt{-3}\sqrt{-1} \overset{?}{=} \sqrt{3}$, which is false, so the statement is not universally true.

(3) It is impossible to refute this statement.

(4) If we take $z_1 = z_2 = -1$, we obtain $\log(-1) + \log(-1) \overset{?}{=} \log 1$, i.e. $i\pi + i\pi \overset{?}{=} 0$, so the statement is not universally true.

(5) If we take $x = y = \sqrt{3}$, we get $\frac{\pi}{3} + \frac{\pi}{3} \overset{?}{=} \frac{-\pi}{3}$, so the statement is not universally true.

2 [8] only defines the concept "branch of log", not a more general definition.

2.6 Differential Algebra

A completely different point of view of view is the differential-algebraic one [17]. Here $\sqrt{1-z}$ is an object whose square is $1-z$, formally definable as w in $\mathbf{C}(z)[w]/(w^2-(1-z))$. Similarly $\log z$ is a new symbol θ such that $\theta' = 1/z$, and so on for other elementary expressions (we do not say 'functions' here, since they are *not* functions in the Bourbaki sense). From this point of view, our exemplar equations take on a very different allure.

(2) The left-hand side is $vw \in K = \mathbf{C}(z)[v,w]/(v^2-(z-1), w^2-(z+1))$, and the right-hand side is $u \in \mathbf{C}(z)[v,w]/(u^2-(z^2-1))$. But to write the equation we have to express $u^2-(z^2-1)$ in K, and it is no longer irreducible, being $(u-vw)(u+vw)$. Depending on which factor we take as the defining polynomial, the equation
$$vw = u \qquad (2')$$
is either true or false (if one were trying to view these as functions, one would say "identically true/false", but that statement has no meaning), and we have to decide which. Once we have decided which, the equation becomes trivially true (or false). The problem is that, with the standard interpretations (which of course takes us *outside* differential algebra), the answer is "it depends on which value of z you have".

(3) The analysis is identical up to the standard interpretations, at which point it transpires that, for the standard interpretations, $vw = u$ is true for all values of z. But, of course, this is what we were trying to prove in the first place.

(4) Here we define θ_1 such that $\frac{\partial\theta_1}{\partial z_1} = \frac{1}{z_1}$ (and $\frac{\partial\theta_1}{\partial z_2} = 0$), θ_2 such that $\frac{\partial\theta_2}{\partial z_2} = \frac{1}{z_2}$ (and $\frac{\partial\theta_2}{\partial z_1} = 0$) and θ_3 such that $\frac{\partial\theta_3}{\partial z_1} = \frac{z_2}{z_1}$ and $\frac{\partial\theta_3}{\partial z_2} = \frac{z_1}{z_2}$. If we then consider $\eta = \theta_1+\theta_2-\theta_3$, we see that
$$\frac{\partial\eta}{\partial z_1} = \frac{\partial\eta}{\partial z_2} = 0 \qquad (4'),$$
which implies that η "is a constant".

(5) Again, the difference between the two sides "is a constant".

We have said "is a constant", since the standard definition in differential algebra is that a constant is an object all of whose derivatives are 0. Of course, this is related to the usual definition by the following.

Proposition 1. *A differentiable function $f: \mathbf{C}^n \to \mathbf{C}$, all of whose first derivatives are 0 in a connected open set D, takes a single value throughout D, i.e. is a constant in the usual sense over D.*

The difference between the two can be seen in these "corrected" versions of (4) and (5), where the choice expressions are the "constants".

$$\log z_1 + \log z_2 = \log z_1 z_2 + \begin{cases} 2\pi i & \arg z_1 + \arg z_2 > \pi \\ 0 & -\pi < \arg z_1 + \arg z_2 < \pi \\ -2\pi i & \arg z_1 + \arg z_2 < -\pi \end{cases} \qquad 4''$$

$$\arctan x + \arctan y = \arctan\left(\frac{x+y}{1-xy}\right) + \begin{cases} \pi & xy > 1, x > 0 \\ 0 & xy < 1 \\ -\pi & xy > 1, x < 0 \end{cases} \qquad (5')$$

Equation (5′) appears as such, *with* the correction term, as [2, p. 205, ex. 13].

2.7 The Pragmatic View

So, which view actually prevails? The answer depends on the context, but it seems to the current author that the view of *most* mathematicians, *most* of the time, is a blend of 2.3 and 2.6. This works because the definitions of differential algebra give rise to power series, and therefore, given "suitable" initial conditions, the expressions of differential algebra can be translated into functions expressed by power series, which "normally" correspond to functions in some open set around those initial conditions.

Whether this is an 'adequate' open set is a more difficult matter — see point 2 in section 5.6.

3 A Textbook Example

What is one to make of statements such as the following[3]?

$$\int \frac{2^{\sqrt{x}}}{\sqrt{x}} \mathrm{d}x = \frac{2^{1+\sqrt{x}}}{\log 2} + C \qquad (8)$$

We ignore any problems posed by "log 2". The proof given is purely in the setting of section 2.6, despite the fact that the source text is entitled *Calculus*. Translated into that language, we are working in $\mathbf{C}(x,u,\theta)$ where $u^2 = x$ and

$$\theta' = (\theta \log 2)/2u. \qquad (9)$$

(We note that equation (9) implicitly gives effect to Convention 1, in that θ' represents $\left(2^{\sqrt{x}} \log 2\right)/2\sqrt{x}$ where the two occurrences of $\sqrt{x}$ represent the same object.) Similarly the right-hand side is $\frac{2\theta}{\log 2} + C$. Note that, having introduced $2^{\sqrt{x}}$, $2^{1+\sqrt{x}}$ is not legitimate in the language of section 2.6, since the Risch Structure Theorem [16] will tell us that there is a relationship between θ and an η standing for $2^{1+\sqrt{x}}$, viz. that η/θ is constant.

4 A Different Point of View

It is possible to take a different approach to these functions, and say, effectively, that "each use of each function symbol means what I mean it to mean at that point". This is completely incompatible with the table-maker's, or the computer's, point of view (section 2.5), but has its adherents, and indeed uses.

[3] (8) is from [2, p. 189, Example 2].

A classic example of this is given in [11, pp. 294–8], and analysed in [13]. The author considers the Joukowski map $f : z \mapsto \frac{1}{2}\left(z+\frac{1}{z}\right)$ and its inverse $f^{-1} : w \mapsto w+\sqrt{w^2-1}$, in two different cases. If we regard these functions as $(f, D, D')_{\mathcal{B}}$ and $\left(f^{-1}, D', D\right)_{\mathcal{B}}$, the cases are as follows.

(i) $D = \{z : |z| > 1\}$. Here $D' = \mathbf{C} \setminus [-1, 1]$. The problem with f^{-1} is interpreting $\sqrt{w^2-1}$ so that $|w+\sqrt{w^2-1}| > 1$.
(ii) $D = \{z : \Im(Z) > 0\}$. Here $D' = \mathbf{C} \setminus ((-\infty, -1] \cup [1, \infty))$, and the problem with f^{-1} is interpreting $\sqrt{w^2-1}$ so that $\Im(w+\sqrt{w^2-1}) > 0$.

We require f^{-1} to be injective, which is a problem, since in both cases $w \mapsto w^2$ is not. Hence the author applies (2) formally (though he does not say so explicitly), and writes

$$f^{-1}(w) = w + \sqrt{w+1}\sqrt{w-1}. \tag{10}$$

(i) Here he takes both $\sqrt{w+1}$ and $\sqrt{w-1}$ to be uses of the square-root function from [1], viz. $\left(\sqrt{\ \ }, \mathbf{C}, \mathbf{C} \equiv \mathbf{R}^+ \overset{\times}{\text{polar}} (-\frac{\pi}{2}, \frac{\pi}{2}] \cup \{0\}\right)_{\mathcal{B}}$. We should note that this means that $\sqrt{w+1}\sqrt{w-1}$ has, at least potentially[4], an argument range of $(-\pi, \pi]$, which is impossible for any single-valued (as in section 2.5) interpretation of $\sqrt{w^2-1}$.
(ii) Here he takes $\sqrt{w+1}$ as before, but $\sqrt{w-1}$ to be an alternative interpretation: $\left(\sqrt{\ \ }, \mathbf{C}, \mathbf{C} \equiv \mathbf{R}^+ \overset{\times}{\text{polar}} [0, \pi) \cup \{0\}\right)_{\mathcal{B}}$.

In terms of section 2.1, of course, $(f, D, D')_{\mathcal{B}}$ is a bijection (in either case), so $\left(f^{-1}, D', D\right)_{\mathcal{B}}$ exists, and the question of whether there is a "formula" for it is not in the language.

5 Formalisations of These Statements?

Of course, the first question is "which kind of statement are we trying to formalise". This matters in two sense — which of the views in section 2 are we trying to formalise, and are we trying to formalise just the statement, or the statement *and* its proof. The question "which view" seems to be a hard one — when reading a text one often has few clues as to the author's intentions in this area. Nevertheless, let us suppose that the view is given.

5.1 The (Bourbakist) Theory

In this view a function is defined by its graph, there is no language of formulae, and the graph of the inverse of a bijective function is the transpose of the graph of the original. Therefore the task of formalising any such statements is the general one of formalising (set-theoretic) mathematical texts.

[4] This is attained: $w = -2$ gives $-\sqrt{3}$, with argument π, whereas $w = -2 - \epsilon i$ gives a result with argument $-\pi + \epsilon'$.

5.2 The Multivalued View

Convention 2. *We use*[5] *capital initial letters to denote the multivalued equivalents of the usual functions, so* $\mathrm{Log}(z) = \{w : \exp(w) = z\}$. *By analogy, we write* $\mathrm{Sqrt}(z) = \{w : w^2 = z\}$.

Here, an expression from the usual mathematical notation, such as (4), becomes, as stated in section 2.2,

$$\begin{aligned}\forall w_3 \in \mathrm{Log}(z_1 z_2) \exists w_1 \in \mathrm{Log}(z_1), w_2 \in \mathrm{Log}(z_2) : w_3 = w_1 w_2 \wedge \\ \forall w_1 \in \mathrm{Log}(z_1), w_2 \in \mathrm{Log}(z_2) \exists w_3 \in \mathrm{Log}(z_1 z_2) : w_3 = w_1 w_2,\end{aligned} \tag{11}$$

There is significant expansion here, and one might be tempted to write

$$\mathrm{Log}(z_1 z_2) = \mathrm{Log}(z_1) + \mathrm{Log}(z_2) \tag{12}$$

using set-theoretic addition and equality of sets, which looks reassuringly like a multivalued version of (4).

However, there are several caveats. One is that, as explained in [6, section 2.3], the correct generalisation of $\log(z^2) = 2\log(z)$ is

$$\mathrm{Log}(z^2) = \mathrm{Log}(z) + \mathrm{Log}(z) \tag{13}$$

(note that Convention 1 does *not* apply here, since we are considering sets of values, rather than merely underspecified values) and *not* $\mathrm{Log}(z^2) = 2\,\mathrm{Log}(z)$ (which, effectively, *would* apply the convention). The second is that not all such equations translate as easily: the multi-valued equivalent of

$$\arcsin(z) \overset{?}{=} \arctan\left(\frac{z}{\sqrt{1-z^2}}\right) \tag{14}$$

is in fact

$$\mathrm{Arcsin}(z) \cup \mathrm{Arcsin}(-z) = \mathrm{Arctan}\left(\frac{z}{\mathrm{Sqrt}(1-z^2)}\right). \tag{15}$$

Conclusion 1. *Translating statements about these functions to the multivalued view is not as simple as it seems, and producing* correct *translations can be difficult.*

It *might* be possible to define a rewrite with constant expansion (a "de Bruijn factor" in the sense of [18]), e.g. by defining new functions such as $\mathcal{AS}(z) = \mathrm{Arcsin}(z) \cup \mathrm{Arcsin}(-z)$, but to the author's knowledge this has not been done, and would probably be a substantial research project.

It would be tempting to wonder about the difficulties of translating proofs, but, other than his and his colleagues', the author has only seen proofs which work by reduction modulo $2\pi i$, and therefore do not generalise to equations like (15), for which the author only knows his own (unpublished) proof.

As has been said, we see little hope for formalising the more general 'Riemann surfaces' version of this view.

[5] Note that this convention is often reversed in Francophone countries.

5.3 The Branch View

In this view, a function is defined locally, in a simply-connected open set, and the statements made informally in mathematics are true in this interpretation if they are true in the informal sense. The first real problem comes in determining the side conditions, such as “not containing 0”. For a *fixed* vocabulary of functions, such as the elementary functions (which can all be derived from exp and log) this can probably be achieved, but when new functions can be introduced, it becomes much harder.

The second problem is to determine what such a suitable open set is, and whether one can be found which is large enough for the mathematical setting envisaged. This is often equivalent to the problem of finding a suitable path, and the challenges are really those of formalising traditional analysis.

5.4 An ‘Applied’ View

This can hardly be said to be formal, and could therefore be ignored. However, in practice a lot of texts are written this way, and it would seem an interesting challenge to see which statements *can* be formalised. But this is clearly harder than formalising a statement once one knows what the context is.

5.5 The Table-Maker’s Point of View

For the elementary functions, the table-maker now has an effective methodology, implicit in [1] and made explicit in [9].

1. Choose a branch cut $\mathcal{X}$ for log, and this defines the value of $\log(z)$ for $z \in \mathbf{C} \setminus \mathcal{X}$ by integration from $\log 1 = 0$.
2. Choose an *adherence rule* to define the value of log on $\mathcal{X}$.
3. For each other function f, choose an expression for $f(x)$ in terms of log. There may be several such choices, and none are perfect.
4. As a consequence of step 3, write down the various simplification rules that f (and other functions) must satisfy.
5. If one is unhappy with these results, return to step 3. **Do not** attempt to rewrite the rules — this leads to inconsistencies, with which tables have been (and alas continue to be) bothered over the years.

Conclusion 2. *In the table-maker’s view,* statements *about multi-valued functions, if correct, are the same as usually stated. However, they may require amplification, as in (5”) versus (5). At least naïvely, such expansion may be unbounded.*

Proofs are a trickier matter. As far as the author knows, such proofs were generally not published before the days of computer algebra, though the table-makers certainly had intuitive understandings of them, at least as regards real variables. Many such proofs are *ad hoc*, but the dangers in an intuitive approach can be seen in (2) versus (3), where apparently similar equations have very different regions of validity. [4] presents a methodology which, for most[6] such equations is guaranteed to either prove or refute them. However, these methods are expensive, and, as they depend on cylindrical algebraic decomposition, the complexity grows doubly-exponentially with the number of variables. Fortunately, there are very few such identities in practice which require more that two complex variables, but even then the methodology has to treat then as four real variables.

Conclusion 3. *Producing (formal) proofs of such statements is a developing subject, even in the context of a computer algebra system. Converting them into an actual theorem prover is a major challenge. Unfortunately, cylindrical algebraic decomposition, as used here, does not seem to lend itself to being used as an 'oracle' by theorem provers.*

5.6 Differential Algebra

The translation from *well-posed* statements of analysis into this viewpoint is comparatively easy, as seen in section 2.6. There are, however, two significant problems.

1. "Well-posed", in the context of differential algebra, means that every extension that purports to be transcendental really is, *and* introduces no new constants. Hence every simplification rule essentially reduces to "correct up to a constant", and beyond here differential algebra does not help us, as seen with (5′).
2. There is no guarantee that the expressions produced by differential algebra, when interpreted as functions, will be well-behaved. This point has been well-explored elsewhere [14,15] so we will do no more than draw attention to it here.

5.7 The Pragmatic View

The fundamental problem with the pragmatic view is that it is a hybrid, and the formalisms in sections 5.3 and 5.6 are different. Indeed, it is precisely this difference that causes most of the problems in practice. The pragmatist takes formulae produced in the differential-algebraic viewpoint, and interprets them in the branch viewpoint. In the branch viewpoint, every integral is continuous, but, as in point 2 above, there is no guarantee of this remaining true in the hybrid view unless special care is taken.

Conclusion 4. *The pragmatist's view, while useful, is indeed a hybrid, and great care must be taken when translated from one viewpoint to the other.*

6 Conclusion

The handling of (potentially) multi-valued functions in mathematics is bedevilled by the variety of viewpoints that can be adopted. Texts are not good at stating

[6] There are some technical limitations on the nesting allowed, but these seem not to occur in practice.

which viewpoint is being adopted, and trying to deduce which (if any) is being adopted is an untackled AI problem.

The most useful viewpoint, it seems, is based on a dual view (section 2.7) of power-series analysis (section 2.3) and differential algebra (section 2.6): the second being quintessentially symbolic computation. These are ultimately based on different formalisms, and there is no guarantee of a smooth translation of the statements, and there may be a requirement for additional proofs that the translations are indeed valid.

Acknowledgements

The author is grateful to the organisers of AISC for inviting this talk, to Patrizia Gianni and Barry Trager for their hospitality while the talk was conceived, and Russell Bradford, Nalina Phisanbut and Stephen Watt for useful comments.

References

1. Abramowitz, M., Stegun, I.: Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, 9th printing. US Government Printing Office (1964)
2. Apostol, T.M.: Calculus, 1st edn., vol. I. Blaisdell (1961)
3. Beaumont, J.C., Bradford, R.J., Davenport, J.H., Phisanbut, N.: Adherence is Better Than Adjacency. In: Kauers, M. (ed.) Proceedings ISSAC 2005, pp. 37–44 (2005)
4. Beaumont, J.C., Bradford, R.J., Davenport, J.H., Phisanbut, N.: Testing Elementary Function Identities Using CAD. AAECC 18, 513–543 (2007)
5. Bourbaki, N.: Théorie des Ensembles. In: Diffusion CCLS (1970)
6. Bradford, R.J., Corless, R.M., Davenport, J.H., Jeffrey, D.J., Watt, S.M.: Reasoning about the Elementary Functions of Complex Analysis. Annals of Mathematics and Artificial Intelligence 36, 303–318 (2002)
7. Carathéodory, C.: Theory of functions of a complex variable. Chelsea Publ., New York (1958)
8. Cartan, H.: Elementary Theory of Analytic Functions of One or Several Complex Variables. Addison-Wesley, Reading (1973)
9. Corless, R.M., Davenport, J.H., Jeffrey, D.J., Watt, S.M.: According to Abramowitz and Stegun. SIGSAM Bulletin 34(2), 58–65 (2000)
10. Davenport, J.H.: Equality in computer algebra and beyond. J. Symbolic Comp. 34, 259–270 (2002)
11. Henrici, P.: Applied and Computational Complex Analysis I. Wiley, Chichester (1974)
12. Jeffreys, H., Jeffreys, B.: Methods of Mathematical Physics, 3rd edn. Cambridge University Press, Cambridge (1956)
13. Kahan, W.: Branch Cuts for Complex Elementary Functions. In: Iserles, A., Powell, M.J.D. (eds.) The State of Art in Numerical Analysis, pp. 165–211 (1987)
14. Lazard, D., Rioboo, R.: Integration of Rational Functions - Rational Computation of the Logarithmic Part. J. Symbolic Comp. 9, 113–115 (1990)

15. Mulders, T.: A note on subresultants and the Lazard/Rioboo/Trager formula in rational function integration. J. Symbolic Comp. 24, 45–50 (1997)
16. Risch, R.H.: Algebraic Properties of the Elementary Functions of Analysis. Amer. J. Math. 101, 743–759 (1979)
17. Ritt, J.F.: Differential Algebra. In: American Mathematical Society Colloquium Proceedings, vol. XXXIII (1950)
18. Wiedijk, F.: The De Bruijn Factor (2000), http://www.cs.kun.nl/~freek/notes/factor.pdf

The Dynamic Dictionary of Mathematical Functions

Bruno Salvy

Mathematical tables have long been used by developers of computer algebra systems. Special functions are implemented in a system using formulas obtained in these tables for the computation of their derivatives, series or asymptotic expansions, numerical evaluation, simplification, integral transforms,... Recently, more and more powerful algorithms have been developed by the computer algebra community, so that nowadays, the flow can be reversed and mathematical tables can be generated automatically for large classes of functions. The gains are obvious: a full automation of the process that reduces the risk for typographical or other errors; more flexibility and extra interactivity for the readers, not only in the navigation, but also in the choice of the actual functions or formulas they need.

Our team has started the development of a Dynamic Dictionary of Mathematical Functions focusing on solutions of linear differential equations (available at `http://ddmf.msr-inria.inria.fr/`). This development raises several interesting issues. Algorithmic ones obviously, but also more fundamental ones (what exactly is needed to define a function?), or practical ones (how should the interaction between computer algebra systems and dynamic mathematical content on the web be organized?). The talk will discuss these issues and present some of the choices that were made in our design.

A Revisited Perspective on Symbolic Mathematical Computing and Artificial Intelligence

Jacques Calmet[1] and John A. Campbell[2]

[1] Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany
calmet@ira.uka.de
[2] Department of Computer Science, University College London, Malet Place, London WC1E 6BT, England
jac@cs.ucl.ac.uk

Abstract. We provide a perspective on the current state and possible future of links between symbolic mathematical computing and artificial intelligence, on the occasion of the 10th biennial conference (AISMC, later AISC) devoted to those connections. It follows a similar perspective expressed for the first such conference in 1992 and then revised and expanded 5 years later. Issues related to the computational management of mathematical knowledge are highlighted.

Keywords: Symbolic Computation, Artificial Intelligence.

1 Introduction

On the occasion of the first conference on symbolic mathematical computing (SMC) and artificial intelligence we wrote [1] a short survey paper on the state of the territory common to the two domains. Because of the background to the inception of the conference, this was aimed mainly at suggesting the potential impact of AI on SMC. Following the evolution of the territory and the experience of the first three conferences, we wrote a larger paper [2] in 1997 on the perspective and likely prospects for research. For the 10th conference in the series, it is timely to take another look at its subject, to see how well the past has agreed (or not agreed) with the suggestions and predictions we made when it was still the future, and to draw lessons as far as possible from both the hits and the misses.

2 The Picture in 1997

The emphasis in our 1997 remarks was somewhat more towards the usefulness of AI approaches for the enhancement of SMC than the converse. Broadly speaking, this followed from our view that knowledge representation, a perennial topic in AI, was likely to be a central feature of the common research ground between AI and SMC. (From the hindsight of 2010, it amounted to a reasonable prediction, though its realization has turned out to be rather different from what we would have expected.)

Certainly until 1965, and to a decreasing extent until the early 1970s, most of SMC could have been regarded as a branch of AI. This came about because finding heuristics for solving various kinds of mathematical problems (notably in indefinite integration), and then putting them to work through SMC, was generally agreed to be a good way of demonstrating progress in AI. But the further research that led to more efficient programs and packages gained this efficiency by finding algorithms for processes that were previously heuristic. To the extent that algorithms replaced heuristics, SMC ceased to be regarded as AI: a good example (but by no means the only one: most of what is now called computer vision, and most parts of natural-language processing, are the largest of the others) of the saying "AI exports its successes".

In the 1990s the situation was that SMC was mainly algorithmic, on an algorithmic-heuristic scale, while AI was towards the other end of the scale. (We except from this contrast the most formal and theoretical areas of both subjects. Those areas have rather different primary concerns.) Again broadly speaking, we saw a situation where AI had many techniques for handling heuristics - which in principle included the considerable amount of heuristic mathematical knowledge held by specialists in SMC but not expressible in their algorithmic systems - while the algorithmically-flavored SMC of that time had much less to offer AI specialists looking for new tools to help them solve their own problems.

We suggested the following lines of development for the future.

- Exploration of the variety of knowledge representations found in AI, to express and manage mathematical knowledge, leading to ....
- Construction of knowledge-based computing environments that combine ideas from AI and SMC;
- Finding suitable annotations of algorithmic knowledge to make it available for use, within heuristic frameworks, in knowledge-based computation;
- Concentration on formal reasoning about mathematical knowledge;
- Use of planning (in the AI sense) in mathematical problem-solving;
- Use of case-based reasoning for mathematical problem-solving;
- Combining formal and informal reasoning within such frameworks;
- A focus on mathematical modeling (as an application area) and its associated mathematical knowledge (e.g. graphs, methods for network modeling, etc.);
- Application of any or all of the above to autonomous agents and multiagent systems;
- Use of any or all of the above in knowledge-based computing environments for the teaching of mathematics.

We also discussed at some length the achievements and prospects of qualitative reasoning, constraint-based reasoning, and machine learning.

3 Hits

We made no distinction in 1997 between likely early developments and those that could be expected to take longer to mature. However, some of the targets we mentioned, e.g. construction of knowledge-based teaching environments, evidently required earlier progress towards others in the list, e.g. at least the first five highlighted items.

As reference to the contents of the AISC conference volumes shows [3], each of those "earlier progress" predictions achieved a hit – though partly because of a more general development of the subject which we did not predict explicitly. (We admit that in 1997 we should have been able to see it coming.) This was the trend towards mathematical knowledge management (MKM) as a topic in its own right. The trend is responsible, among other things, for the "MKM" series of conferences, which exist now in a federation with the AISC and Calculemus conferences. Most of the list in section 2 expresses parts of the current research agenda of mathematical knowledge management.

4 Misses

As we go further down the list, we find fewer post-1997 papers on the relevant developments when we look in the AISC proceedings and elsewhere. For some items, such as teaching, the "miss" is likely to be only a temporary miss, which should be remedied over the next decade. For others, particularly those in the final paragraph of section 2, we did not at all succeed in predicting the future - partly because those topics have become more specialized and self-contained over the last decade, which was not an historically necessary evolution. From the perspective of 2010 the suggestions remain as interesting as they were in 1997, but it appears from the contents of the last seven AISC conferences that the authors of papers found other directions to be more rewarding.

In section 3 we have claimed MKM as an implicit hit of our 1997 suggestions. A critic of [2] might say at the same time, with some justification, that the fact that we did not refer there to MKM as such was an explicit miss. Probably this happened because we took it for granted, given our AI cultural background, that the automated management of mathematical knowledge was 'obviously' part of the goal of our business. We ought to have been aware that consequences of such implicit reasoning can be unexpected.

Our section 2 recommendation of "Combining formal and informal reasoning" was in effect a recommendation for the combination of heuristic and algorithmic computation - scientifically and practically desirable in itself, but also a means of bringing AI and SMC more closely together than they have been since the mid-1960s. It amounts to bypassing a purely 'mathematical' concept of symbolic computation. It means also the introduction of heuristics for specific application problems when purely algorithmic constructive procedures will be difficult or impossible to achieve in the foreseeable future.

A straightforward example is to provide symbolic solutions for linear homogeneous ordinary differential equations. The optimal relevant algorithm has been known for almost 20 years, but its implementation requires solving representational problems for Galois groups that are very challenging. Methods of AI, also even reference to numerical methods, could be of great help. A related example, more open-ended, deals with systems of partial differential equations for which even the most optimistic computer algebraist cannot expect a significant breakthrough within the next 20 years. Here is where heuristics for specific problems, and general computational experiments with heuristics, could definitely be helpful. A practical instance lies within biology and associated health sciences, where basic models consist of systems of partial differential equations. Constraint-based programming with a strong flavor of AI could have an impact. Even qualitative reasoning could be worth investigating here. We have in mind a rather different scenario from the recent past picture of applications of computer algebra or constraint-based programming in biological areas, which appears to consist of mainly isolated uses of methods or programs that were no longer providing new results in the field(s) where they were first developed.

Our 1997 'miss' with respect to use of AI plus SMC for teaching, e.g. smart tutoring systems for the use of AI tools without requiring users to have any detailed knowledge of information technology, is still an open recommendation for the future, and can rely on results achieved already through progress in the 'hit' areas. Another example relevant to this part of our perspective is cultural reasoning based on concepts originating in the integration of computer algebra and theorem proving [4].

5 Suggestions for the Future

First, we have no reason to back down from any of our 1997 suggestions that have not been taken up in or around SMC since 1997. They are still useful stimuli for researchers in AI and SMC. In effect, they are restated in section 2. We intend to consider how they can be promoted through identification of suitable test applications; while they would not be 'grand challenges' (a phrase popular in some physical and mathematical sciences), we might be justified in calling any such test a minigrand challenge.

In order to succeed in MKM from a user's point of view, we need to clarify what knowledge should be managed. Typical MKM research has no difficulty in identifying the most formal and textbook-oriented kinds of knowledge, but users (e.g, students) tend to have trouble and require help with epiphenomena around the edges of the 'real' knowledge too. Even something as deceptively simple as notation can be a source of trouble, partly because it conceals (and is the product of) implicit or semi-implicit expert mathematical knowledge. Making this 'epi' knowledge explicit, and representing it for MKM, is a development on which any future widespread take-up of MKM systems will depend.

The biggest general change in the textbook picture of AI between the late 20th century and 2010, as shown by the textbooks from the two periods, is the arrival of the Bayesian outlook. There was no trace of it in our 1997 paper. In effect, Bayesian methodologies are now being seen as some kind of generic methodology for knowledge representation. Here, especially if more attention is paid to our 1997 suggestion about graphs as a medium for mathematical modeling, SMC is in an excellent strategic position to establish links with applied probability theory and general AI at the same time.

In a neighboring domain, it is worth considering whether attempts to describe ontologies in the language of category theory could provide a link between a formal description of SMC and AI.

Finally, we use our freedom as invited contributors to AISC 2010 to say something more speculative. Starting from a mathematical culture, it could be at least a good intellectual exercise to think about a question that has been raised by physicists and (some) AI experts: Is there (an) AI for everything? One might answer "yes" if one believes, as some physicists do, that everything can be computed. But in a physicist's terms the problem is so difficult that only incremental progress can be hoped for. It is surely (also) a philosophical question: we should not forget that John McCarthy was pointing out the link between philosophy and AI from the very beginning of the latter. From the side of SMC, constructive procedures can be seen as the scaffolding on/from which to design algorithms. This is a very challenging job. Usually one tries a simpler approach instead - starting the job from heuristics. ("Starting the job from heuristics" accounts for the history of the divergence between SMC and AI that was easiest to observe while it was happening from 1965 to 1975.) Perhaps we can tackle both approaches at the same time. For systems of partial differential equations that state physicists' models of some parts of the physical world, for example, we could look at the integrability of their solutions [5], e.g. starting from Galois cohomology or involutive bases. And to add a speculation to a speculation for the future, perhaps a Bayesian approach to assigning probabilities to the pieces of knowledge available in that domain can have a part to play.

References

1. Calmet, J., Campbell, J.A.: Artificial Intelligence and Symbolic Mathematical Computations. In: Campbell, J., Calmet, J. (eds.) AISMC 1992. LNCS, vol. 737, pp. 1–19. Springer, Heidelberg (1993)
2. Calmet, J., Campbell, J.A.: A perspective on symbolic mathematical computing and artificial intelligence. Annals of Mathematics and Artificial Intelligence 19(3-4), 261–277 (1997)
3. Lists of the contents of the proceedings of all the conferences, `http://www.informatik.uni-trier.de/~ley/db/conf/aisc/index.html`
4. Calmet, J.: Abstraction-Based Information Technology: A Framework for Open Mechanized Reasoning. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) Calculemus 2009. LNCS, vol. 5625, pp. 14–26. Springer, Heidelberg (2009)
5. Calmet, J., Seiler, W.M., Tucker, R.W. (eds.): Global Integrability of Field Theories. Proc. of GIFT 2006. Universitätsverlag Karlsruhe, Karlsruhe (2006)

I-Terms in Ordered Resolution and Superposition Calculi: Retrieving Lost Completeness*

Hicham Bensaid[1,2], Ricardo Caferra[2], and Nicolas Peltier[2]

[1] INPT/LIG, Avenue Allal Al Fassi, Madinat Al Irfane,
Rabat, Morocco
bensaid@inpt.ac.ma

[2] LIG, Grenoble INP/CNRS
Bâtiment IMAG C - 220, rue de la Chimie 38400 Saint Martin d'Hères, France
Ricardo.Caferra@imag.fr, Nicolas.Peltier@imag.fr

Abstract. Ordered resolution and superposition are the state-of-the-art proof procedures used in saturation-based theorem proving, for non equational and equational clause sets respectively. In this paper, we present extensions of these calculi that permit one to reason about formulae built from *terms with integer exponents* (or *I-terms*), a schematisation language allowing one to denote infinite sequences of iterated terms [8]. We prove that the ordered resolution calculus is still refutationally complete when applied on (non equational) clauses containing *I*-terms. In the equational case, we prove that the superposition calculus is not complete in the presence of *I*-terms and we devise a new inference rule, called *H-superposition*, that restores completeness.

Keywords: Automated reasoning, term schematisation, terms with integer exponents, resolution and superposition calculi, refutational completeness.

The range of application of a proof procedure obviously depends on the language that is used to represent and manipulate information. Richer languages guide the design of provers offering new possibilities to the users. In the last years [4,5] the authors have used *term schematisations* with the aim of improving the generality and expressive power of automated reasoning systems. Term schematisations can be seen as formalisms allowing one to *structurally* describe (infinite) sets of standard terms, obtained by applying repeatedly a given *inductive pattern* on some *base term*. For instance the sequence of terms $a, f(a), f(f(a)), \ldots$ may be represented by the term schema $f(\diamond)^N.a$ where N is an *arithmetic variable*, to be instantiated by a natural number (formal definitions will be given in Section 1.1). $f(\diamond)$ denotes the repeated pattern ($\diamond$ is a special symbol denoting a *hole*) and a is the base term. Several formalisms have been proposed to grasp such sequences of iterated terms. They differ mainly by the class of inductive patterns they consider. The concept of *recurrent terms* has been initially introduced in [7]

* This work has been partly funded by the project ASAP of the French *Agence Nationale de la Recherche* (ANR-09-BLAN-0407-01).

(see also [9]) and then extensions of the original formalism have been proposed, which gave rise to a hierarchy of term schematisation languages, with various expressiveness and complexity: the *terms with integer exponents* [8], the *R-terms* [20] and the *primal grammars* [10].

These formalisms have been introduced in order to avoid divergence of symbolic computation procedures (particularly in rewriting). In our approach, schematisations are intended to be applied to terms appearing as arguments of predicate symbols in the inference process. The usefulness of such formalisms in automated deduction is almost obvious. They allow more compact and more natural formalisations and may improve the termination behaviour of the calculus. For instance the set of formulae $S = \{\mathit{even}(0), \forall x\, \mathit{even}(x) \Rightarrow \mathit{even}(s(s(x)))\}$ that could in principle generate an infinite number of consequences $\mathit{even}(s(s(0)))$, $\mathit{even}(s(s(s(s(0)))))$, $\ldots$ can be replaced by a single axiom $\{\forall N\, \mathit{even}(s(\diamond)^{2N}.0)\}$[1]. Proof length and readability may also be improved, for instance in the above specification $\mathit{even}(s^{2k}(0))$ can be proven in only one step, instead of k without using term schemata.

Inference rules have been proposed to generate automatically such schematisations from standard specifications (e.g. to infer the formula $\forall N\, \mathit{even}(s(\diamond)^{2N}.0)$ from S), see e.g. [20,15,16]. This is done by (a particular) detection of cycles in proof search. Inductive approaches can also be used to generate such formulae, by analysing the search space in order to detect regularities [4]. This is useful for partially describing provers search spaces, for avoiding divergence in the inference process or for discovering lemmata (particularly in inductive theorem proving). Incorporating such capabilities to (automated or assisted) theorem provers is a way of improving them *qualitatively*.

Obviously, in order to apply term schematisations in automated theorem proving, a basic requirement is to integrate these languages into the most successful existing proof procedures, especially in ordered resolution and superposition calculi (see e.g. [2,14]). In [5] we described DEI, the first theorem prover able to handle a particular class of term schematisations: the terms with integer exponents (or *I-terms*) of [8]. DEI[2] is an extension of the E-prover [21] and it uses ordered resolution and superposition as base calculi. The purpose of the present work is to establish the theoretical properties of these calculi in presence of I-terms.

As can be expected, more expressive power entails in general losing "nice" properties of the calculus and in our case the addition of I-terms actually destroys refutational completeness of the superposition calculus (this is not the case of ordering resolution, for which completeness is preserved, as shown in Section 2). In order to recover completeness, we propose to add a new inference rule, called *H-superposition*, especially devoted to handle I-terms and encoding in a single step an *unbounded* number of applications of the superposition rule.

[1] Of course in this trivial case the resolution calculus obviously terminates if ordering restrictions are considered. Notice actually that this is not the case if positive refinements of the calculus are used such as hyper-resolution [17] or hyper-tableaux [3] (positive calculi are extensively used in model building, see e.g. [6]).

[2] The system is available at `http://membres-lig.imag.fr/peltier/dei.html`

The language of I-terms has been chosen because it is used in our current implementation and because it presents a good tradeoff between expressiveness and simplicity, but actually the same ideas can be applied to other schematisation languages.

The paper is structured as follows. Section 1 introduces the main definitions, in particular the syntax and semantics of I-terms and recalls some basic notions and definitions in saturation based theorem proving. Section 2 describes the extended calculus in the non equational case and proves its completeness. Section 3 deals with the equational case, using an extension of the superposition calculus. Section 4 briefly concludes the paper.

1 Definitions and Notations

1.1 Terms with Integer Exponents

We consider three disjoint (nonempty) sets of symbols: a set of *ordinary variables* V_X, a set of *function symbols* $\mathcal{F}$ and a set of *arithmetic variables* V_N. Let $\diamond$ be a special symbol not occurring in $V_X \cup V_N \cup \mathcal{F}$. $\diamond$ is called the *hole* and is used to define inductive contexts (see Definition 1). Arithmetic variables will be denoted by capital letters $N, M, \ldots$ and ordinary variables by $x, y, \ldots$.

We first define the set of terms with integer exponents (or I-terms) and the set of terms with one hole (originally introduced in [8]). For the sake of readability, examples are given before formal definitions.

Example 1. (I-terms) $f(a, \diamond, b)^N.g(c)$ is an I-term denoting the infinite set of (standard) terms $\{g(c), f(a, g(c), b), f(a, f(a, g(c), b), b), ...\}$.

E_N denotes the set of arithmetic expressions built as usual on the signature 0, s, $+$ and V_N (in particular $\mathbb{N} \subseteq E_N$).

Definition 1. *The set of* terms with integer exponents T_I *(or I-*terms*) and the set of* terms with one hole $T_\diamond$ *are the smallest sets verifying:*

- $\diamond \in T_\diamond$ *and* $V_X \subset T_I$.
- *If* $t_1, \ldots, t_k \in T_I$, $f \in \mathcal{F}$ *and* $arity(f) = k$ *then* $f(t_1, \ldots, t_k) \in T_I$.
- *If* $arity(f) = k > 0$, $t_j \in T_I$ *for* $j \in [1..k], j \neq i$ *and* $t_i \in T_\diamond$ *then* $f(t_1, \ldots, t_n) \in T_\diamond$.
- *If* $t \in T_\diamond$, $t \neq \diamond$, $s \in T_I$ *and* $N \in E_N$ *then* $t^N.s \in T_I$. *An I-term of this form is called an N-*term. *t is the* inductive context*, N is the* exponent *and s is the* base term.

1.2 Positions and Subterms

A *position* is a finite sequence of natural numbers. Λ denotes the empty position and . denotes the *position concatenation* (it is sometimes omitted, e.g. we write p^k for $p. \ldots .p$). We define a function *Pos* giving the set of *positions* in an I-term (slightly different from that of [8], but using the same notations). $Pos(t)$ contains all the positions of the I-terms occurring in t (even the terms occurring in an inductive context or in a base term of an N-term).

Example 2. Let $t = f(a, g(\diamond))^N.g(b)$. $Pos(t) = \{\Lambda, 1.1, 2, 2.1\}$. The position Λ corresponds to the term t itself (root position). 1.1 corresponds to the subterm a, 2 to the subterm $g(b)$ and 2.1 to the subterm b. Notice that $1, 1.2$ and 1.2.1 do not occur in $Pos(t)$ because they correspond to the terms $f(a, g(\diamond))$, $g(\diamond)$ and $\diamond$ respectively which are *not* I-terms.

Definition 2 (I-term positions). *The function Pos giving the set of I-*term positions *of a term in $T_I \cup T_\diamond$ is defined as:*

- $Pos(t) = \{\Lambda\}$ *if* $t \in V_X$.
- $Pos(\diamond) = \emptyset$.
- $Pos(f(t_1, \ldots, t_n)) = E \cup \{i.q \mid t_i \in T_I \cup T_\diamond, q \in Pos(t_i)\}$, *where* $E = \{\Lambda\}$ *if* $t_1, \ldots, t_n \in T_I$ *and* $E = \emptyset$ *otherwise.*
- $Pos(t^n.s) = \{\Lambda\} \cup \{1.q \mid q \in Pos(t)\} \cup \{2.q \mid q \in Pos(s)\}$.

The notation $t|_p$ (where $t \in T_I \cup T_\diamond$ and $p \in Pos(t)$) denotes (as usual) the subterm of t occurring in the position p.

- $t|_p = t$ if $p = \Lambda$.
- $f(t_1, \ldots, t_n)|_{i.q} = t_i|_q$ if $i \in [1..n], q \in Pos(t_i)$.
- $(t^N.v)|_{1.q} = t|_q$ if $q \in Pos(t)$.
- $(t^N.v)|_{2.q} = v|_q$.

Proposition 1. *For every term $t \in T_I \cup T_\diamond$ and for every $p \in Pos(t)$, $t|_p$ is an I-term.*

$t[s]_p$ denotes as usual the term obtained from t by replacing the I-term at position $p \in Pos(t)$ by s, formally defined as follows:

- $t[s]_p = s$ if $p = \Lambda$.
- $f(t_1, \ldots, t_n)[s]_{i.q} = f(t_1, \ldots, t_{i-1}, t_i[s]_q, t_{i+1}, \ldots, t_n)$ if $i \in [1..n], q \in Pos(t_i)$.
- $(t^N.v)[s]_{1.q} = (t[s]_q)^N.v$ if $q \in Pos(t)$.
- $(t^N.v)[s]_{2.q} = t^N.(v[s]_q)$ if $q \in Pos(v)$.

Proposition 2. *For every term $t \in T_I$ (resp. $t \in T_\diamond$) we have $t[s]_p \in T_I$ (resp. $t[s]_p \in T_\diamond$).*

When no confusion is possible, the notation $t[s]_p$ can be used also to ensure that $t|_p$ is syntactically equal to s, i.e. $t|_p = s$. Notice that if t is a standard term then the previous notions coincide with the usual ones.

If t is a term with a hole, we denote by $t[s]_\diamond$ the term obtained from t by replacing the (unique) occurrence of the hole (not in an N-term) by s. Formally, $t[s]_\diamond$ is defined inductively: $\diamond[s]_\diamond = s$ and $f(t_1, \ldots, t_n)[s]_\diamond = f(t_1, \ldots, t_{i-1}, t_i[s]_\diamond, t_{i+1}, \ldots, t_n)$ where $t_i \in T_\diamond$.

Given an I-term t, $var(t)$ denotes the *set of variables* of t (from both V_X and V_N). t is *ground* if $var(t) = \emptyset$.

1.3 Semantics of *I*-Terms

The semantics of I-terms are defined by the following rewrite rules:

$$t^0.s \to_I s \qquad t^{n+1}.s \to_I t[t^n.s]_\diamond$$

This rewrite system is convergent and together with the rules of Presburger arithmetic, it reduces any I-term not containing arithmetic variables to a standard term. We denote by $t\downarrow$ the normal form of t w.r.t. $\to_I$.

In the following, we use $=_I$ to denote equality modulo unfolding, i.e. $t =_I s$ iff $t\downarrow = s\downarrow$. To simplify technicalities and w.l.o.g. we assume that the considered terms are normalised w.r.t. the previous rules and that every (non ground) exponent occurring in an N-term is a variable (by unfolding, e.g. $f(\diamond)^{2N+1}.a$ is equivalent to $f(f(f(\diamond))^N.a))$. Notice that every ground I-term is $=_I$-equivalent to a ground (standard) term.

A *substitution* is a function mapping every standard variable to an I-term and every arithmetic variable to an arithmetic expression. For any expression (term, clause, ...) $\mathcal{E}$, we denote by $\mathcal{E}\sigma$ the expression obtained from $\mathcal{E}$ by replacing every variable $x \in V_X \cup V_N$ by its image by σ. $\mathcal{E}\sigma$ is called an *instance* of $\mathcal{E}$. $dom(\sigma)$ denotes the set of variables x s.t. $x\sigma \neq x$. A substitution σ is $\mathbb{N}$-*grounding* if $dom(\sigma) = V_N$ and $\forall N \in V_N$, $N\sigma \in \mathbb{N}$ (in practice, of course, the domain of σ does not need to be infinite: it may be restricted to the variables really occurring in the considered terms).

An $\mathbb{N}$-instance of an I-term t is obtained by instantiating all the integer variables in t by natural numbers (the other variables are *not* instantiated):

Definition 3 ($\mathbb{N}$-instance). *A term s is said to be an* $\mathbb{N}$-instance *of an I-term t if and only if there exists an $\mathbb{N}$-grounding substitution σ s.t. $s = t\sigma\downarrow$.*

For example, x, $f(x)$ and $f(f(x))$ are $\mathbb{N}$-instances of $f(\diamond)^N.x$, $f(a)$ or $f(\diamond)^{N+1}(b)$ are not (although these last two terms are instances of $f(\diamond)^N.x$).

1.4 *I*-Term Ordering

We assume given an ordering $\succ$ on I-terms, that satisfies the following properties:

1. $\succ$ is *total* on *ground* I-terms.
2. $\succ$ is *well founded*, i.e. there is no infinite sequence of terms $t_1, \ldots, t_n, \ldots$ such that $\forall i, t_i \succ t_{i+1}$.
3. $\succ$ is *closed by substitution*, i.e. for any substitution σ and terms t, s: $t \succ s \Rightarrow t\sigma \succ s\sigma$.
4. $\succ$ is *compatible with unfolding*, i.e. for all I-terms t, s: $(t =_I s) \Rightarrow (\forall u, t \bowtie u \Rightarrow s \bowtie u)$ where $\bowtie \in \{\succ, \prec\}$.

Properties $1, 2, 3$ are standard, Property 4 ensures that $\succ$ is compatible with the semantics of I-terms.

Given any ordering $\succ$ on terms satisfying the properties 1-3 above, we can extend $\succ$ to an ordering $\succ_I$ on I-terms satisfying properties 1-4 as follows: $t \succ_I s$

if one of the following conditions holds (we assume that t, s are normalised w.r.t. the rules $\rightarrow_I$ above and x denotes a variable not occurring in t, s):

- t, s are standard terms and $t \succ s$.
- $t = f(t_1, \ldots, t_n), s = f(s_1, \ldots, s_n)$ and $\forall i \in [1..n], t_i \succeq s_i$ and $\exists i \in [1..n], t_i \succ s_i$.
- $t = u^N.v$ and $v \succ_I s$.
- $t = u^N.v$, $u[x]_\diamond \succ_I s$ and $v \succ s\{N \rightarrow 0\}$.
- $t = u^N.v$, $s = w^N.r$, $u[x]_\diamond \succeq w[x]_\diamond$ and $v \succ_I w$.

For instance, we have $f(f(\diamond))^N.g(g(a)) \succ_I f(\diamond)^N.g(a) \succ_I a$ and $g(f(\diamond)^N.a, \diamond)^N.a \succ_I f(\diamond)^N.a$, but $g(f(\diamond)^N.x, \diamond)^N.a \not\succ_I f(\diamond)^N.x$ (take $N = 0$ and $x \succ a$). Of course more refined definitions are possible, yielding more restrictive orderings. Defining refined versions of the usual reduction orderings and efficient algorithms for computing these relations is out of the scope of the present paper and is part of future work.

For any two unifiable (standard) terms t, s, $mgu(t, s)$ denotes the (unique up to variable renaming) most general unifier of t and s. If t, s are I-terms, they can have *several* (but finitely many) most general unifiers [8]. Thus we denote by $mgu(t, s)$ the (finite) set of most general unifiers. Unification algorithms for I-terms can be found in [8,20].

1.5 Clauses, Redundancy Criteria and Selection Functions

We consider a set of predicate symbols containing in particular the equality predicate $\approx$ (in infixed notation). An *I-atom* is of the form $p(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are I-terms and p is a predicate symbol. An *I-literal* is either an I-atom or the negation of an I-atom. An *I-clause* is a disjunction of I-literals. The empty (I-)clause is denoted by $\Box$. It is clear that an I-clause containing no N-term is a clause (in the usual sense).

The notions of instance and $\mathbb{N}$-instance can obviously be extended to I-clauses. Clearly, all $\mathbb{N}$-instances of an I-clause are (standard) clauses. Interpretations are defined as usual. An interpretation I *validates* a set of I-clauses S (written $I \models S$) if I validates *all* $\mathbb{N}$-instances of the I-clauses in S (in the usual sense).

The ordering $\succ$ is extended as usual to I-atoms (in case of equational atoms we use a multiset extension of the ordering on terms), I-literals (by ignoring the negation symbol) and to I-clauses (by multiset extension).

Proofs are constructed by using the *ordered resolution calculus* [18,13,2] for the non equational case and the *superposition calculus* in the equational case [1,14]. In addition to the inference rules these calculi use *redundancy elimination rules* to prune the search space. The following definition [1] formalises the notions of *redundancy* and *saturation*:

Definition 4 (Redundancy and saturation). *An I-clause C is said to be* redundant *w.r.t. a set of I-clauses S if for every ground instance $C\sigma$ of C, there exist n clauses $D_1, \ldots, D_n \in S$ and n ground substitutions $\theta_1, \ldots, \theta_n$ such that $D\theta_i \prec C\sigma$ and $\{D_i\theta_i \mid i \in [1..n]\} \models C\sigma$. A set S is* saturated *(w.r.t. a given set of inference rules) iff every I-clause that is deducible from S (using the considered rules) is redundant w.r.t. S.*

As well known, this definition covers the subsumption and elimination of tautology rules.

The calculi are parameterised as usual by a *selection function sel* mapping every I-clause C to a set of *selected literals* in C, such that either $sel(C)$ contains a negative literal or $sel(C)$ contains all maximal (w.r.t. $\succ$) positive literals in $sel(C)$. We also assume that $sel(C)$ is compatible with $=_I$ i.e. if $C =_I D$ then $sel(C) =_I sel(D)$ and that $L\sigma \in sel(C\sigma) \Rightarrow L \in sel(C)$, for every substitution σ.

2 Non Equational Case

In this section, we consider only clauses and I-clauses not containing the equality predicate $\approx$. The following definition extends the *ordered resolution calculus* used in the context of non equational clauses to I-clauses. If A, B are two atoms, $mgu(A, B)$ denotes the set of most general unifiers of A and B.

Definition 5 (Ordered resolution calculus). *The* ordered resolution *calculus ($\mathcal{OR}$) is defined by the two following inference rules*:

Ordered resolution

$$\frac{C \vee A \qquad D \vee \neg B}{(C \vee D)\sigma}$$

where $\sigma \in mgu(A, B)$, $A\sigma$ and $\neg B\sigma$ are selected in $(C \vee A)\sigma$ and $(D \vee \neg B)\sigma$ respectively.

Ordered factorisation

$$\frac{C \vee A \vee B}{(C \vee B)\sigma}$$

where $\sigma \in mgu(A, B)$ and $B\sigma$ is selected in $(C \vee A \vee B)\sigma$.

The definition almost coincides with the usual one (see for example [13]) except that the condition $\sigma = mgu(A, B)$ is replaced by $\sigma \in mgu(A, B)$ (since unification of I-terms is not unitary, $mgu(A, B)$ is no longer unique). It is well-known that $\mathcal{OR}$ is refutationally complete when applied on standard clauses.

The following lemma relates the saturatedness of a set of I-clauses S_I to that of the underlying set of $\mathbb{N}$-instances.

Lemma 1. *Let S_I be a set of I-clauses and let S be the set of* all $\mathbb{N}$*-instances of S_I. If S_I is saturated under $\mathcal{OR}$ then so is S.*

Proof. Suppose that S_I is saturated under $\mathcal{OR}$. Let C be a clause derived from S. We show that C is redundant w.r.t. S. We distinguish two cases:

– If C is derived from some clauses $D = D' \vee A$ and $E = E' \vee \neg B$ by ordered resolution where $D'\sigma$ and $E'\sigma$ are selected in $D\sigma$ and $E\sigma$ respectively, then $C = (D' \vee E')\sigma$ with $\sigma = mgu(A, B)$. By definition, there exist I-clauses D_I and E_I in S_I such that D and E are respectively instances of D_I and E_I. D_I and E_I are of the form $D'_I \vee A_I$ and $E'_I \vee \neg B_I$, where there exist $\mathbb{N}$-grounding substitutions θ_D, θ_E s.t. $D'_I\theta_D =_I D'$, $E'_I\theta_E =_I E'$, $A_I\theta_D =_I A$ and $B_I\theta_E =_I B$.

Let $\theta = \theta_D\theta_E$. By definition, since D_I and E_I are variable-disjoint, we have $A_I\theta =_I A$ and $B_I\theta =_I B$, hence A_I and B_I are unifiable: $A_I\theta\sigma =_I B_I\theta\sigma$. Since $\theta\sigma$ is a unifier of A_I and B_I, there exist an m.g.u. $\sigma_I \in mgu(A_I, B_I)$ and a substitution τ s.t. $\theta\sigma = \sigma_I\tau$. Since $A\sigma \in sel(D\sigma)$ and $\neg B\sigma \in sel(E\sigma)$ we have $A_I\sigma_I \in sel(D_I\sigma_I)$ and $\neg B_I\sigma_I \in sel(E_I\sigma_I)$ (sel is closed by substitution and compatible with $=_I$).

Therefore, the ordered resolution rule can be applied between D_I and E_I on the literals A_I and $\neg B_I$ yielding an I-clause of the form $C_I = (D'_I \vee E'_I)\sigma_I$.

We have $C = C_I\tau$. Since S_I is saturated, C_I is redundant w.r.t. S_I. Consequently, C is also redundant (since any ground instance of C is also a ground instance of C_I). But then, since by definition, S and S_I have the same ground instances, C is also redundant w.r.t. S.

– The proof is similar if C is derived by factorisation.

Theorem 1 states the refutational completeness of $\mathcal{OR}$.

Theorem 1 ($\mathcal{OR}$ refutation completeness). *Every unsatisfiable set of I-clauses that is saturated w.r.t. $\mathcal{OR}$ contains the empty clause.*

Proof. Let S_I be a saturated (w.r.t. $\mathcal{OR}$) unsatisfiable set. Then S the set of all S_I $\mathbb{N}$-instances is unsatisfiable. By Lemma 1, S is saturated. Since $\mathcal{OR}$ is (refutationally) complete when applied on standard clauses, S contains the empty clause $\Box$. Thus by definition S_I contains $\Box$ (no other I-clause has as instance $\Box$).

3 Equational Case

Equality can of course be handled by adding the equality axioms, but this technique is not efficient. In practice, it is preferable to consider the equality predicate as a part of the language syntax and to use dedicated calculi handling for instance the substitution of equals by equals at deep positions in the clauses. The most restrictive and successful version of these calculi is *superposition* [1]. We now extend this calculus to I-clauses. As usual, we assume in this section that equality is the only predicate symbol (atoms of the form $p(t_1, \dots, t_n)$ are replaced by $p(t_1, \dots, t_n) \approx true$).

Definition 6 (Superposition calculus). *The* superposition calculus *($\mathcal{SC}$) is defined by the following inference rules:*

Superposition (S)

$$\frac{Q \vee (l \approx r) \qquad D \vee (w[t]_p \bowtie u)}{(Q \vee D \vee (w[r]_p \bowtie u))\sigma}$$

where $\bowtie \in \{\approx, \not\approx\}$, $\sigma \in mgu(l, t)$, $l\sigma \not\preceq r\sigma$, $w\sigma \not\preceq u\sigma$, $(l \approx r)\sigma$, $(w \bowtie u)\sigma$ are selected in $(Q \vee l \approx r)\sigma$ and $(D \vee w \bowtie u)\sigma$ respectively, and t is not a variable.

Equality resolution (ER)

$$\frac{Q \vee (l \not\approx r)}{Q\sigma}$$

where $\sigma \in mgu(l, r)$ and $(l \not\approx r)\sigma$ is selected in $(Q \vee l \not\approx r)\sigma$.

Equality factoring (EF)

$$\frac{Q \vee (t \approx s) \vee (l \approx r)}{(Q \vee (s \not\approx r) \vee (l \approx r))\sigma}$$

where $\sigma \in mgu(l,t)$, $l\sigma \not\preceq r\sigma$, $t\sigma \not\preceq s\sigma$ *and* $(t \approx s)\sigma$ *is selected in* $(Q \vee l \approx r \vee t \approx s)\sigma$.

Again, the only difference with the usual case is the non unicity of the m.g.u. $\mathcal{SC}$ coincides with the usual superposition calculus when the considered clauses contain no N-terms.

Uncompleteness of $\mathcal{SC}$

The superposition calculus is *not* refutationally complete when applied on I-clauses. The following example (already given in [5]) illustrates this assertion:

Example 3

$$\begin{cases} (1)\ p(f(a,\diamond)^N.b) \\ (2)\qquad \neg p(c) \\ (3)\quad f(a,b) \approx d \\ (4)\quad f(a,d) \approx c \end{cases}$$

This set of I-clauses is clearly unsatisfiable since instantiating N by 2 gives $f(a, f(a, b))$ and then replacing successively $f(a, b)$ by d and $f(a, d)$ by c leads to the contradiction $p(c)$ and $\neg p(c)$. However, we cannot derive the empty clause with the previous rules. We only have one possible unification: $f(a,\diamond)^N.b$ and $f(a, b)$ with the substitution $\sigma = \{N \to 1\}$. The superposition rule generates the clause $p(d)$ from (1) and (3). d is unifiable neither with $f(a, b)$ nor with $f(a, d)$. No superposition is then possible and proof search ends without deriving the empty clause (a saturated set is obtained).

Restoring Completeness

The problem in Example 3, where superposition involves a subterm in the hole path (the base term), is that when we unify $f(a,\diamond)^M.b$ with its instance $f(a, b)$, the N-term structure is "lost". The idea then is to change the superposition rule to preserve this structure. A closer investigation reveals that incompleteness occurs only when superposition is applied on a subterm in the path leading to the hole of an N-term (including the base term). Thus in the new calculus that we propose, resolution and factorisation rules are kept without changes, as well as superposition involving subterms not occurring in the hole path.

Defining a complete superposition calculus for clauses containing I-terms is not a straightforward task. The reason is that an I-clause denotes an infinite sequence of standard clauses, with an unbounded number of distinct (non variable) positions. For instance, the $\mathbb{N}$-instances of $p(f(\diamond)^N.a)$ contain the positions $\Lambda, 1, 1.1, 1.1.1, \ldots, 1.\underbrace{1.\ldots.1}_{N \text{ times}}\ldots$, although $Pos(p(f(\diamond)^N.a))$ contains only

$\{\Lambda, 1, 1.2\}$. In principle, the superposition rule has to be applied at each of these positions, which would make the calculus infinitary (i.e. one could deduce infinitely many I-clauses by one step of application of the inference rules, which makes the calculus very difficult to handle in practice). This makes an important difference with the usual case: although a first-order clause also represents an infinite number of ground clauses (with an unbounded number of positions), it is well-known that applying the superposition rule inside variables is useless, thus actually the rule can be restricted to the positions occurring in the first-order term. This is not the case here and Example 3 shows that restricting to the positions occurring in the I-terms is not sufficient.

We now show how to overcome this problem. The idea is that, thanks to the expressive power of the I-terms, all these consequences may actually be encoded into a single I-clause.

We define H-superposition[3] rule as follows.

H-superposition (HS)

$$\frac{Q \vee (l \approx r) \qquad D \vee (w[t[\diamond]_p^M.s]_o \bowtie u)}{(Q \vee D \vee (w[t^N.t[r]_q]_o \bowtie u))\sigma_M \sigma}$$

where $\bowtie \in \{\approx, \not\approx\}$, q is a non empty prefix of p (with possibly $p = q$), $o \in Pos(w)$ and:

- N, N' are two distinct arithmetic variables not occurring in the premises.
- $\sigma_M = \{M \to N + N' + 1\}$.
- $\sigma \in mgu(t[t^{N'}.s]_\diamond|_q \sigma_M, l)$.

$\mathcal{IS}$ denotes the calculus defined by the above rules: H-superposition, superposition, equality resolution, equality factoring. Before proving the completeness of the calculus, we give some examples. For the sake of clarity, we distinguish two cases.

- Assume first that $q = p$. In this case the N-term $t^M.s$ is unified with an I-term l and the solution substitution will instantiate M to some natural number $m \in \mathbb{N}$. $t^m.s$ unifies with l, thus can be replaced by the term r in the right-hand side of the equation. But actually, if we consider the set of $\mathbb{N}$-instances of $t^M.s$, the replacement of $t^m.s$ does not need to be performed only at the root level in the term $t^M.s$. For the sake of completeness, it must occur also in $t[t^m[s]_p]_p$ and in $t[t[t^m[s]_p]_p]_p$ and so on *for all possible instantiations* of M (greater or equal than m). Using standard superposition will discard all the other possible unifications except for the case where M is instantiated to m (unifying $t^m.s$ and l). With H-superposition however, we will consider all these possibilities: $t^m.s$ is unified with l and the N-term structure is preserved by introducing a new (fresh) variable N (encoding the number of unfoldings of the term $t^M.s$ before we reach the position on which the replacement is performed). Roughly speaking, M is decomposed into two integer variables: $M = N+1+N'$. The term $t[\diamond]_p^M.s$ can then be represented

[3] H stands for "**H**ole".

as $t[\diamond]_p^N.t^1.t[\diamond]_p^{N'}.s$. The new base term $t[\diamond]_p^{N'}.s$ is unified with l and the new iterated part (with exponent N) allows us to keep the N-term structure. Applying these considerations to the previous (unsatisfiable) example we can derive the empty clause:

(1)	$p(f(a,\diamond)^M.b) \approx true$		
(2)	$p(c) \not\approx true$		
(3)	$f(a,b) \approx d$		
(4)	$f(a,d) \approx c$		
(5)	$p(f(a,\diamond)^N.f(a,d)) \approx true$	HS (3),(1)	$\sigma=\{M \to N+N'+1\}\{N' \to 0\}$
(6)	$p(c) \approx true$	S (4),(5)	$\sigma=\{N \to 0\}$
(7)	$true \not\approx true$	S (6),(2)	
(8)	$\square$	EIR (7)	

– If $p \neq q$ then l is unified with a subterm of $t[\diamond]_p^M.s$ occurring along the inductive path. In this case no unification is possible without unfolding. Again, after unfolding, considering unification directly without decomposing the variable M leads as in the previous case to "loose" the N-term structure. There is however a difference, namely the subterm concerned with the unification can occur in any (non empty) position along the hole path. This is why we consider in our rule all (not empty) prefixes of the hole position. For instance consider the following example:

$$\begin{cases} (1)\ p(f(a,g(\diamond))^M.b) \approx true \\ (2)\ p(c) \not\approx true \\ (3)\ f(a,e) \approx d \\ (4)\ f(a,g(d)) \approx c \\ (5)\ g(b) = e \end{cases}$$

Again this is an unsatisfiable set but no (standard) superposition rule can be found to be applied: No unification is possible among left hand side terms. Using our rules however allows us to overcome this problem:

(6)	$p(f(a,g(\diamond))^N.f(a,e)) = true$	HS (5),(1)	$\sigma=\{M \to N+N'+1\}\{N' \to 0\}$
(7)	$p(f(a,g(\diamond))^N.d) = true$	S (3),(6)	
(8)	$p(c) = true$	S (4),(7)	$\sigma=\{N \to 1\}$
(9)	$true \not\approx true$	S (8),(2)	
(10)	$\square$	ER (9)	

We now show the refutational completeness of $\mathcal{IS}$. It deserves to be noted that without using the redundancy criterion in Definition 4, it can be very hard to establish completeness. For instance, consider the set $S_I = \{p(f(a,\diamond)^n.c), a \approx b\}$. Using $\mathcal{SC}$ on S, the set of all S_I ground instances, we can derive the clause $p(f(b,f(a,c)))$ for example. With $\mathcal{IS}$ however, we can derive only $p(f(b,\diamond)^n.c)$. Thus, arguments used in proof of Lemma 1 cannot be used directly. The clause $p(f(b,f(a,c)))$ however is redundant w.r.t the set $\{p(f(b,\diamond)^n.c), a \approx b\}$. Hence, using redundancy elimination, allows one to discard this kind of clauses and to establish completeness.

We start by proving an essential lemma which relates the subterms occurring in an I-term t to the ones occurring in an $\mathbb{N}$-instance of t.

Lemma 2. *Let t_I be an I-term. Let $t = t_I\sigma\downarrow$ be an $\mathbb{N}$-instance of t_I. If s is a subterm of t then one of the following conditions holds:*

1. *t_I contains a term s_I s.t. $s_I\sigma\downarrow= s$.*
2. *t_I contains an N-term $u^N.v$ and there exists $m < N\sigma$ s.t. s occurs in $u[u^m.v]_\diamond\sigma\downarrow$, at a position that is a non empty prefix of the hole position in u.*

Proof. The proof is by induction on the term t_I. By definition, if $s = t$ then Condition 1 holds. Otherwise, since t cannot be a variable, t_I is not a variable. If t_I is of the form $f(t_I^1,\ldots,t_I^n)$, then s must occur in a term $t_I^i\sigma\downarrow$ for $i \in [1..n]$. By the induction hypothesis, one of the conditions 1 or 2 must hold for t_I^i, thus also for t_I since t_I contains t_I^i. The proof is similar if t_I is of the form $u^N.v$ and s occur in $v\sigma\downarrow$ or in $u\sigma\downarrow$. If $t_I = u^N.v$ and s occurs neither in $u\sigma\downarrow$ nor in $v\sigma\downarrow$, then s must occur in $t_I\sigma\downarrow$ along the position $p^{N\sigma}$, where p denotes the position of the hole $\diamond$ in u. If the position of s in $t_I\sigma\downarrow$ is empty, then Condition 1 holds, by definition (since $s = t_I\sigma\downarrow$). Otherwise, s occurs at a position of the form $p^k.q$ where q is a non empty prefix of p and $k < N\sigma$. In this case, s occurs in $u[u^{N\sigma-1-k}.v]_\diamond\sigma\downarrow$ hence Condition 2 is satisfied.

The second lemma states that replacing a subterm occurring in an I-term can be simulated by *several* replacements in the underlying $\mathbb{N}$-instances.

Lemma 3. *Let t_I, s_I be two I-terms and let $p \in Pos(t_I)$. Let σ be an $\mathbb{N}$-grounding substitution. Let $t'_I = t_I[s_I]_p$, $t = t_I\sigma\downarrow$, $s = s_I\sigma\downarrow$ and $t' = t'_I\sigma\downarrow$. Let u_I be the subterm occurring at position p in t_I and let $u = u_I\sigma\downarrow$. t can be obtained from t' by replacing some occurrences of s by u.*

Proof. The proof is by induction on t_I. We distinguish several cases according to the position p and to the form of t_I. If $p = \Lambda$, we have by definition $t_I = u_I$, $t = u$ and $t' = s$. Thus the proof is immediate. If $p = i.q$ and t_I is of the form $f(t_I^1,\ldots,t_I^n)$, then $t'_I = f(t_I^1,\ldots,t_I^i[s_I]_q,\ldots,t_I^n)$. By the induction hypothesis, $t_I^i\sigma\downarrow$ is obtained from $t_I^i[s_I]_q\sigma\downarrow$ by replacing some occurrences of s by u, thus the same holds for $t = f(t_I^1\sigma\downarrow,\ldots,t_I^n\sigma\downarrow)$ and $t' = t'_I = f(t_I^1\sigma\downarrow,\ldots,t_I^i[s_I]_q\sigma\downarrow,\ldots,t_I^n\sigma\downarrow)$.

Now assume that t_I is of the form $v^N.w$. If $p = 2.q$ where q is a position in w then the proof is similar to the previous one. Otherwise, $p = 1.q$ where q is a position in v (notice that by definition of $Pos(t_I)$, $v|_q$ cannot contain the hole). Let $r = v[s_I]_q$. We have $t'_I = r^N.w$, $t_I\sigma\downarrow= v[\ldots[v[w\sigma\downarrow]_\diamond]\ldots]_\diamond$ and $t'_I\sigma\downarrow= r[\ldots[r[w\sigma\downarrow]_\diamond]\ldots]_\diamond$. By the induction hypothesis (applied on a subterm of v containing the term at position q but not containing the hole), r is obtained from v by replacing some occurrences of s by u, thus the same holds for t and t' (by repeating these replacements for each occurrence of v and r).

Lemma 4. *Let S_I be a set of I-clauses and S the set of* all *$\mathbb{N}$-instances of S_I. If S_I is saturated under $\mathcal{IS}$ then S is saturated under $\mathcal{SC}$.*

Proof. Suppose that S_I is saturated under $\mathcal{IS}$. Let C be a clause derived from S (with $\mathcal{SC}$). We show that C is redundant w.r.t. S. We distinguish all possible cases:

– C is derived by superposition. There exist two clauses $Q = Q' \vee (l \approx r)$, $D = D' \vee (w[t]_q \bowtie u)$ s.t. $C = (Q' \vee D' \vee (w[r]_q \bowtie u))\sigma$ where $\sigma = mgu(l,t)$. By definition, there exist I-clauses Q_I and D_I in S_I such that Q and D are $\mathbb{N}$-instances of Q_I and D_I respectively. Thus there exist four I-terms l_I, r_I, w_I, u_I, two I-clauses Q'_I and D'_I and two substitutions θ_Q, θ_D s.t. $l_I\theta_Q =_I l$, $r_I\theta_Q =_I r$, $Q'_I\theta_Q = Q'$, $w_I\theta_D =_I w[t]_q$, $u_I\theta_D =_I u$.
We denote by θ the substitution $\theta_D\theta_Q$ (we assume that D and Q share no variables). We distinguish two cases:
1. w_I contains a subterm t_I, at a position $p \in Pos(w_I)$, s.t. $t_I\theta =_I t$. By definition of the superposition rule, t cannot be a variable (superposition is not allowed on variables). Since t is an $\mathbb{N}$-instance of t_I, t_I cannot be a variable. Moreover, $p \in Pos(w_I)$ and t_I is unifiable with l_I, with unifier $\theta\sigma$. By definition, there exists an m.g.u. $\sigma_I \in mgu(t_I, l_I)$ and a substitution τ s.t. $\theta\sigma = \sigma_I\tau$. Furthermore, since $l\theta\sigma \not\preceq r\theta\sigma$ and since $\succ$ is closed under substitution, we must have $l_I\sigma_I \not\preceq r_I\sigma_I$ (and similarly for the other ordering and selection restrictions in the application conditions of the superposition rule). Hence, the superposition rule is applicable between Q_I and D_I yielding an I-clause $C_I = (Q'_I \vee D'_I \vee w_I[r_I]_p \bowtie u_I)\sigma_I$. Consider the clause $C_I\tau$. Since $\sigma_I\tau = \theta\sigma$, we have $C_I\tau = (Q'_I \vee D'_I \vee w_I[r_I]_p \bowtie u_I)\theta\sigma =_I (Q'\sigma \vee D'\sigma \vee w_I[r_I]_p\theta\sigma \bowtie u\sigma)$. Notice that this clause is *not* $=_I$-equivalent to C in general. Indeed, the replacement of t_I by r_I in w_I may create several copies of this term, after unfolding the different contexts in which it occurs, whereas in w, only one instance of this term is replaced. By Lemma 3, $w_I\sigma_I\tau\downarrow$ can be obtained from $w_I[r_I]_p\sigma_I\tau\downarrow$ by replacing some occurrences of $l\sigma$ by $r\sigma$. Thus the same holds for $w[r]_q\sigma = w_I\sigma_I\tau$ and C can be obtained from $C_I\tau\downarrow$ by replacing some occurrences of $r\sigma$ by $l\sigma$. This immediately implies that C is redundant w.r.t. $\{C_I\tau, Q\}$. Since S_I is saturated, C_I must be redundant w.r.t. S_I, hence C is redundant w.r.t. S_I. But since S_I and S have the same ground instances, C is also redundant w.r.t. S.
2. Otherwise, by Lemma 2, w_I contains (at some position o) an N-term $s_I^M.v$ such that $t =_I s_I[s_I^m.v]_\diamond|_p\theta$, where p is a (non empty) prefix of the position of the hole in s_I and $m < M\sigma$. Let N, N' be two fresh integer variables and let $\sigma_M = \{M \to N + N' + 1\}$ (following the notations of HS rule). θ is extended to N, N' as follows: $N'\theta = m$ and $N\theta = M\theta - 1 - N'\theta$ (thus $M\theta = N\theta + N'\theta + 1$). By the previous relations we have $t =_I s_I[s_I^{N'}.v]_\diamond|_p\theta$. Thus $s_I[s_I^{N'}.v]_\diamond|_p$ and l_I have an m.g.u. σ_I that is more general than $\theta\sigma$, i.e. there exists τ such that $\theta\sigma = \sigma_I\tau$. By the same reasoning than for the previous case, we show that the ordering restrictions in the application conditions of the superposition rule hold. Thus the H-superposition rule can be applied between Q_I and D_I yielding an I-clause $C_I = (Q'_I \vee D'_I \vee (w_I[s_I^N.s_I[r_I]_p]_o \bowtie u_I))\sigma_M\sigma_I$. Since

$l_I\sigma_I = s_I[s_I^{N'}.v]_\diamond|_p\sigma_I$, we have $(s_I^{N+N'+1}.v)\sigma_I =_I (s_I^N.s_I^1.s_I^{N'}.v)\sigma_I =_I (s_I^N.s_I)\sigma_I[l_I\sigma_I]_p$. By Lemma 3, $w_I[s_I^N.s_I[r_I]_p]_o\sigma_I\tau \downarrow$ can be obtained from $w_I\sigma_I\tau \downarrow$ by several replacements of $(s_I^{N+N'+1}.v)\sigma_I\tau \downarrow$ by $(s_I^N.s_I[r_I]_p)\sigma_I\tau \downarrow$. Thus C is redundant w.r.t. $C_I\tau \cup S_I$. Since S_I is saturated, C_I is actually redundant w.r.t. S_I hence C is redundant w.r.t. S_I (hence to S).

- C is derived by equality resolution. The proof is similar to the one for the resolution rule (see Lemma 1).
- C is derived by equality factoring. The proof is similar to that of the previous case.

Theorem 2 ($\mathcal{IS}$ refutation completeness). *Every saturated (w.r.t. $\mathcal{IS}$) unsatisfiable set of I-clauses contains the empty clause.*

Proof. The proof is similar to that of Theorem 1.

4 Conclusion

The present work is a natural continuation of [5], in which we described the theorem prover DEI, an extension of the well-known E-prover [21] devoted to handle I-terms. The present paper focuses on completeness issues. It shows that the resolution calculus (with ordering restrictions and selection functions) is refutationally complete when applied on clauses containing I-terms (in the non equational case). However, this property is not fulfilled by the superposition calculus (in the equational case). We show that this problem can be fixed by adding a new inference rule into the calculus. Detailed completeness proofs are provided (soundness is of course straightforward). Our work is, to the best of our knowledge, the first attempt to propose refutationally complete calculus for such languages.

Future work includes the implementation of this calculus and the evaluation of its practical performances. The extension of the usual reduction orderings to I-terms also deserves to be investigated. Comparision with the works on metaterms [12] and on deduction modulo [11] will also be considered.

References

1. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. Journal of Logic and Computation 3(4), 217–247 (1994)
2. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson and Voronkov [19], pp. 19–99
3. Baumgartner, P.: Hyper Tableaux — The Next Generation. In: de Swart, H. (ed.) TABLEAUX 1998. LNCS (LNAI), vol. 1397, pp. 60–76. Springer, Heidelberg (1998)
4. Bensaid, H., Caferra, R., Peltier, N.: Towards systematic analysis of theorem provers search spaces: First steps. In: Leivant, D., de Queiroz, R. (eds.) WoLLIC 2007. LNCS, vol. 4576, pp. 38–52. Springer, Heidelberg (2007)

5. Bensaid, H., Caferra, R., Peltier, N.: Dei: A theorem prover for terms with integer exponents. In: Schmidt, R.A. (ed.) Automated Deduction – CADE-22. LNCS, vol. 5663, pp. 146–150. Springer, Heidelberg (2009)
6. Caferra, R., Leitsch, A., Peltier, N.: Automated Model Building. Applied Logic Series, vol. 31. Kluwer Academic Publishers, Dordrecht (2004)
7. Chen, H., Hsiang, J., Kong, H.-C.: On finite representations of infinite sequences of terms. In: Okada, M., Kaplan, S. (eds.) CTRS 1990. LNCS, vol. 516, pp. 100–114. Springer, Heidelberg (1991)
8. Comon, H.: On unification of terms with integer exponents. Mathematical Systems Theory 28(1), 67–88 (1995)
9. Farmer, W.M.: A unification algorithm for second-order monadic terms. Annals of Pure and Applied Logic 39(2), 131–174 (1988)
10. Hermann, M., Galbavý, R.: Unification of infinite sets of terms schematized by primal grammars. Theor. Comput. Sci. 176(1-2), 111–158 (1997)
11. Hermant, O.: Resolution is cut-free. Journal of Automated Reasoning 44(3), 245–276 (2010)
12. Kirchner, H.: Schematization of infinite sets of rewrite rules generated by divergent completion processes. Theoretical Comput. Sci. 67(2-3), 303–332 (1989)
13. Leitsch, A.: The resolution calculus. In: Texts in Theoretical Computer Science. Springer, Heidelberg (1997)
14. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson and Voronkov [19], pp. 371–443
15. Peltier, N.: Increasing the capabilities of model building by constraint solving with terms with integer exponents. Journal of Symbolic Computation 24, 59–101 (1997)
16. Peltier, N.: A General Method for Using Terms Schematizations in Automated Deduction. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 578–593. Springer, Heidelberg (2001)
17. Robinson, J.A.: Automatic deduction with hyperresolution. Intern. Journal of Computer Math. 1, 227–234 (1965)
18. Robinson, J.A.: A machine-oriented logic based on the resolution principle. J. Assoc. Comput. Mach. 12, 23–41 (1965)
19. Robinson, J.A., Voronkov, A. (eds.): Handbook of Automated Reasoning, vol. 2. Elsevier/MIT Press (2001)
20. Salzer, G.: The unification of infinite sets of terms and its applications. In: Voronkov, A. (ed.) LPAR 1992. LNCS, vol. 624, pp. 409–420. Springer, Heidelberg (1992)
21. Schulz, S.: System Description: E 0.81. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004. LNCS (LNAI), vol. 3097, pp. 223–228. Springer, Heidelberg (2004)

Structured Formal Development with Quotient Types in Isabelle/HOL

Maksym Bortin[1] and Christoph Lüth[2]

[1] Universität Bremen, Department of Mathematics and Computer Science
maxim@informatik.uni-bremen.de
[2] Deutsches Forschungszentrum für Künstliche Intelligenz, Bremen
christoph.lueth@dfki.de

Abstract. General purpose theorem provers provide sophisticated proof methods, but lack some of the advanced structuring mechanisms found in specification languages. This paper builds on previous work extending the theorem prover Isabelle with such mechanisms. A way to build the quotient type over a given base type and an equivalence relation on it, and a generalised notion of folding over quotiented types is given as a formalised high-level step called a design tactic. The core of this paper are four axiomatic theories capturing the design tactic. The applicability is demonstrated by derivations of implementations for finite multisets and finite sets from lists in Isabelle.

1 Introduction

Formal development of correct systems requires considerable design and proof effort in order to establish that an implementation meets the required specification. General purpose theorem provers provide powerful proof methods, but often lack the advanced structuring and design concepts found in specification languages, such as *design tactics* [20]. A design tactic represents formalised development knowledge. It is an abstract development pattern proven correct once and for all, saving proof effort when applying it and guiding the development process. If theorem provers can be extended with similar concepts without loss of consistency, the development process can be structured within the prover. This paper is a step in this direction. Building on previous work [2] where an approach to the extension of the theorem prover Isabelle [15] with theory morphisms has been described, the contributions of this paper are the representation of the well-known type quotienting construction and its extension with a generalised notion of folding over the quotiented type as a design tactic. Two applications of the tactic are presented, demonstrating the viability of our approach.

The paper is structured as follows: we first give a brief overview of Isabelle and theory morphisms to keep the paper self-contained. Sect. 3 describes the four theories which form the design tactic, giving more motivation for it and sketching the theoretical background. Further, Sect. 4 shows how the design tactic can be applied in order to derive implementations of finite multisets and finite sets. Finally, Sect. 5 contains conclusions and sketches future work.

2 Isabelle and Theory Morphisms

Isabelle is a logical framework and LCF-style theorem prover, where the meta-level inference system implements an intuitionistic fragment of the higher order logic extended with Hindley-Milner polymorphism and type classes.

Isabelle, and other LCF provers, structure developments in hierarchical theories. This goes well with the predominant development paradigm of conservative extension, which assures consistency when developing large theories from a small set of axioms (such as HOL or ZF). A different approach, going back to Burstall and Goguen [3], is to use *theory morphisms* as a structuring device. Instead of one large theory we have lots of *little theories* [8], related by theory morphisms. Structuring operations are given by *colimits* of diagrams of morphisms [9], of which (disjoint and non-disjoint) unions and parametrisation are special cases. Early systems in this spirit include IMPS [8] and Clear [4], later systems the OBJ family with its most recent offspring CafeOBJ [6], and the SpecWare system [21]. An extension of the early Edinburgh LCF with theory morphisms was described in [18], but never integrated into later LCF systems. A recent development in this vein is a calculus for reasoning in such structured developments [13], as used in the CASL specification languages [14].

The morphism extension package for Isabelle [2] provides implementations of key concepts such as signature and theory morphisms, and seamlessly extends Isabelle's top-level language Isar with the commands necessary to express these notions; we will use these commands in the following. A crucial property is that any theory morphism $\tau : \mathcal{T} \longrightarrow \mathcal{T}'$ from a theory $\mathcal{T}$ to a theory $\mathcal{T}'$ firstly induces the homomorphic extension $\overline{\sigma}_\tau$ of the underlying signature morphism σ_τ to propositions, and secondly the extension $\overline{\tau}$ of τ to proof terms. This allows the translation of any theorem ϕ in $\mathcal{T}$ to a theorem $\overline{\sigma}_\tau(\phi)$ in $\mathcal{T}'$, translating the proof π of ϕ to $\overline{\tau}(\pi)$ and replaying it in $\mathcal{T}'$. It is syntactically represented in Isar by the command **translate-thm** ϕ **along** τ.[1]

Furthermore, the approach gives a simple notion of a parameterised theory, extending the theory hierarchy: a theory $\mathcal{B}$ is parameterised by $\mathcal{P}$ (denoted $\langle \mathcal{P}, \mathcal{B} \rangle$) if an inclusion morphism $\iota : \mathcal{P} \hookrightarrow \mathcal{B}$ exists or, in other words, $\mathcal{B}$ imports $\mathcal{P}$; an instantiation of $\langle \mathcal{P}, \mathcal{B} \rangle$ is given by a theory morphism $\tau : \mathcal{P} \longrightarrow \mathcal{I}$ as shown by the following diagram

$$\begin{tikzcd}
\mathcal{P} \arrow[r, hook] \arrow[d, "\tau"'] & \mathcal{B} \arrow[d, dashed, "\tau^\sharp"] \\
\mathcal{I} \arrow[r, hook, dashed] & \mathcal{I}^\sharp
\end{tikzcd} \tag{1}$$

[1] The current release 0.9.1 for Isabelle2009-1 can be downloaded at `http://www.informatik.uni-bremen.de/~cxl/awe` and all theories presented here can be found in the directory `Examples/Quotients`.

where the extended theory $\mathcal{I}^\sharp$ and the dashed morphisms are automatically derived. In other words, the resulting theory $\mathcal{I}^\sharp$ is the pushout of the diagram, and is computed via the Isar command **instantiate-theory** $\mathcal{B}$ **by-thymorph** τ.

3 Folding Quotient Types Using Hylomorphisms

A design tactic can be encoded as a parametrisation $\langle \mathcal{P}, \mathcal{B} \rangle$, where $\mathcal{P}$ contains formal parameters and their axiomatic specifications, and $\mathcal{B}$ contains deductions in form of definitions and theorems using the formal parameters and axioms imported from $\mathcal{P}$. In this section, we introduce a design tactic which performs two constructions: firstly, it constructs the quotient of a type with respect to an equivalence relation, and secondly, it gives a generic mechanism to define 'folding' functions on the quotient type. The tactic has two parameters: the type with the equivalence relation, and the parameters of the fold. Thus, the design tactic comprises two parametrisations in the sense of (1) above:

$$\mathit{QuotientType\text{-}Param} \hookrightarrow \mathit{QuotientType} \hookrightarrow \mathit{Fold\text{-}Param} \hookrightarrow \mathit{Fold} \qquad (2)$$

The first parametrisation $\langle \mathit{QuotientType\text{-}Param}, \mathit{QuotientType} \rangle$ comprises the basic machinery regarding equivalence classes, class operations, quotient types and congruences. The core of the design tactic is the second parametrisation $\langle \mathit{Fold\text{-}Param}, \mathit{Fold} \rangle$, describing how to construct hylomorphisms on quotient types, and will be explicitly described in Sect. 3.5 and Sect. 3.6.

3.1 Quotient Types

Roughly, any equivalence relation $\simeq$ on a type τ induces a partition on $\mathit{Univ}(\tau)$, i.e. on the set containing all elements of this type. Elements of this partition are predicates and correspond to the $\simeq$-equivalence classes. This is a well-known technique. Indeed, the quotient type is a powerful construction, and implemented in many theorem provers, either axiomatically [16] (for NuPRL) or as a derived construction. The former always bears the danger of inconsistencies (see [10] for a model construction; [5] presents an implementation for Coq); the latter is made easier by the presence of a choice operator and extensionality, allowing quotient types in HOL [12] or Isabelle [19,17]. However, the main novelty here is the way in which a fold operator is defined on the quotient types as a hylomorphism in the abstract setting of parameterised theories, combining the advantages of the little-theories approach with a construction motivated from type theory.

3.2 The Theory *QuotientType-Param*

This theory declares an unary type constructor T and a relation $\simeq$ as a polymorphic constant, together with axioms specifying $\simeq$ as an equivalence relation:

typedecl $\alpha\,\mathsf{T}$
const $\_ \simeq \_ :: (\alpha\,\mathsf{T} \times \alpha\,\mathsf{T})\ \mathsf{set}$
axioms (E1) : $s \simeq s$
(E2) : $s \simeq t \Longrightarrow t \simeq s$
(E3) : $s \simeq t \Longrightarrow t \simeq u \Longrightarrow s \simeq u$

3.3 The Theory *QuotientType*

We are interested in the partition of $\mathit{Univ}(\alpha \mathsf{T})$: $Q_{\simeq} \equiv \{\{v | u \simeq v\} | u \in \mathit{Univ}(\alpha \mathsf{T})\}$, and introduce the new unary quotient type constructor $\mathsf{T}/_{\simeq}$

typedef $\alpha\, \mathsf{T}/_{\simeq} = Q_{\simeq}$

Further, we define the class operations $\mathit{class\text{-}of}_{\simeq} :: \alpha\, \mathsf{T} \Rightarrow \alpha\, \mathsf{T}/_{\simeq}$ (as usually denoted by $[\_]_{\simeq}$) and $\mathit{repr}_{\simeq} :: \alpha\, \mathsf{T}/_{\simeq} \Rightarrow \alpha\, \mathsf{T}$, such that the following familiar properties, essential for equivalence relations and quotients, can be proven:

$$([s]_{\simeq} = [t]_{\simeq}) = (s \simeq t) \tag{3}$$
$$[\mathit{repr}_{\simeq}\ (q)]_{\simeq} = q \tag{4}$$
$$\mathit{repr}_{\simeq}\ ([s]_{\simeq}) \simeq s \tag{5}$$

The crucial observation is that the entire development from now on relies only on these three basic properties of the class operations, i.e. we essentially abstract over the particular representation of quotients.

A function $f :: \alpha\, \mathsf{T} \Rightarrow \beta$ is called a $\simeq$-*congruence* if it respects $\simeq$ [17]; this is expressed by the predicate $\mathit{congruence}_{\simeq} :: (\alpha\, \mathsf{T} \Rightarrow \beta)\ \mathsf{set}$ defined as

$$\mathit{congruence}_{\simeq} \equiv \{f \mid \forall\, s\, t.\, \neg\, s \simeq t\ \vee\ f\ s = f\ t\}$$

Moreover, the higher order function $\_^{\mathsf{T}/\simeq} :: (\alpha\, \mathsf{T} \Rightarrow \beta) \Rightarrow (\alpha\, \mathsf{T}/_{\simeq} \Rightarrow \beta)$, which factors any $\simeq$-congruence $f :: \alpha\, \mathsf{T} \Rightarrow \beta$ through the projection $\mathit{class\text{-}of}_{\simeq}$, i.e. such that

$$f \in \mathit{congruence}_{\simeq} \Longrightarrow f^{\mathsf{T}/\simeq}\ [s]_{\simeq} = f\ s \tag{6}$$

holds, is defined as $f^{\mathsf{T}/\simeq} \equiv f \circ \mathit{repr}_{\simeq}$. The direction $\Longleftarrow$ in (6) can be then shown as well, emphasising that the congruence condition is also necessary. Further, let $g :: \alpha\, \mathsf{T} \Rightarrow \alpha\, \mathsf{T}$ be a function. The instantiation of f by $\mathit{class\text{-}of}_{\simeq} \circ g$ in (6) gives

$$(\mathit{class\text{-}of}_{\simeq} \circ g) \in \mathit{congruence}_{\simeq} \Longrightarrow (\mathit{class\text{-}of}_{\simeq} \circ g)^{\mathsf{T}/\simeq}\ [s]_{\simeq} = [g\ s]_{\simeq} \tag{7}$$

All these derived properties are well-known, but note that the complete development here is parameterised over the type constructor T and the relation $\simeq$, and thus can be re-used in a variety of situations.

3.4 Defining Functions over Quotient Types

In order to define a function f on the quotient type $\alpha\, \mathsf{T}/_{\simeq}$, we have to show that f agrees with the equivalence relation $\simeq$. Equation (6) gives us sufficient conditions for this. The following theory development makes use of this for a design tactic which axiomatises sufficient conditions to conveniently define linear recursive functions, or *hylomorphisms* [7], on the quotient type. We first motivate the development by sketching the special case of lists, and then generalise to arbitrary data types.

In Isabelle, the parameterised type $\alpha\,\mathsf{list}$ of lists of elements of type α is freely generated by the constructors $Nil :: \alpha\,\mathsf{list}$ and the infix operator $\# :: \alpha \Rightarrow \alpha\,\mathsf{list} \Rightarrow \alpha\,\mathsf{list}$. Suppose we would like to prove

$$(\forall ys)\frac{xs \sim ys \qquad (f,e) \in C}{foldr\ f\ e\ xs = foldr\ f\ e\ ys}$$

by structural induction on the list xs, where $\sim$ is an equivalence relation on lists, and f and e are additionally restricted by some (usually non-trivial) side condition C. The crucial point would be the induction step, where based on the assumption $x\#xs \sim ys$ we need to find some list zs satisfying $xs \sim zs$ and, moreover allowing us to conclude $f\ x\ (foldr\ f\ e\ zs) = foldr\ f\ e\ ys$. In many cases such zs can be computed by a function $Transform\ x\ xs\ ys$ constructing a list which satisfies the desired properties under the premises $x\#xs \sim ys$ and $(f,e) \in C$; thus, we can say the proof is parameterised over the function $Transform$.

Hylomorphisms are particular kinds of recursive functions which can be expressed in terms of (co-)algebras for the same type. Consider a parameterised type $\alpha\,\Sigma$, together with an action Σ on functions (normally called *map*; the map on types and functions together form a functor). Then an *algebra* for Σ is a type γ and a function $A :: \gamma\,\Sigma \Rightarrow \gamma$, a *coalgebra* is a type β and a function $B :: \beta \Rightarrow \beta\,\Sigma$, and the solution of the *hylo-equation* [7]

$$\phi = A \circ \Sigma\phi \circ B \tag{8}$$

is a function $\phi :: \beta \Rightarrow \gamma$, called the *hylomorphism* from B to A. Hylomorphisms correspond to linear recursive functions and can be compiled efficiently; hence, deriving them via a general design tactic is relevant.

In the case of lists, the list signature is represented by the type $(\alpha,\beta)\,\Sigma_{\mathsf{list}} \overset{def}{=} \mathbf{1} + \alpha \times \beta$ (as usual, $\times$ denotes the product and $+$ the disjoint sum of two types), together with the map function $\Sigma_{\mathsf{list}} :: (\beta \Rightarrow \gamma) \Rightarrow (\alpha,\beta)\,\Sigma_{\mathsf{list}} \Rightarrow (\alpha,\gamma)\,\Sigma_{\mathsf{list}}$ defined by the equations

$$\begin{aligned} \Sigma_{\mathsf{list}}\ f\ (\iota_L\ *) &= \iota_L\ * \\ \Sigma_{\mathsf{list}}\ f\ (\iota_R\ (u,x)) &= \iota_R\ (u, f\ x) \end{aligned}$$

The type $\alpha\ \mathsf{list}$ from above, together with the function $in_{\mathsf{list}} :: (\alpha, \alpha\,\mathsf{list})\,\Sigma_{\mathsf{list}} \Rightarrow \alpha\,\mathsf{list}$, defined in the obvious way sending $\iota_L *$ to Nil and $\iota_R\,(u,x)$ to $u\#x$, forms the initial Σ_{list}-algebra. Its inverse is the function out_{list}, which forms a Σ_{list}-coalgebra, i.e. we have the right-inverse property: $in_{\mathsf{list}} \circ out_{\mathsf{list}} = id_{\mathsf{list}}$. The initiality of in_{list} means that any Σ_{list}-algebra $A :: (\alpha,\beta)\,\Sigma_{\mathsf{list}} \Rightarrow \beta$ determines the unique algebra homomorphism $\phi_A : \alpha\ \mathsf{list} \Rightarrow \beta$, i.e.

$$\phi_A \circ in_{\mathsf{list}} = A \circ \Sigma_{\mathsf{list}}\phi_A \tag{9}$$

holds. If we compose both sides of (9) with out_{list} on the right and use the right-inverse property of out_{list}, we obtain the fact that ϕ_A satisfies the hylo-equation (8), i.e. is the hylomorphism from out_{list} to A.

The unique function ϕ_A can be defined using *foldr*. That *foldr* determines hylomorphisms from out_{list} to any Σ_{list}-algebra is an important observation, because in the following we want to explore the congruence properties of hylomorphisms. Taking also into account that many structures can be implemented via quotients over lists, we obtain the possibility to extend *foldr* to $foldr^{\mathsf{list}/\sim}$ and to calculate with $foldr^{\mathsf{list}/\sim}$ based on the numerous properties of *foldr*.

3.5 The Theory *Fold-Param*

We will now generalise the previous development to an arbitrary type constructor Σ, and formalise it as a parameterised theory. The parameter theory *Fold-Param* is constructed in four steps; the body theory *Fold* follows in Sect. 3.6.

(1) Representing signatures. First of all, a signature is represented by a declaration of a binary type constructor Σ, together with the two polymorphic constants representing the action of Σ on relations and mappings, respectively.

typedecl $(\alpha, \beta)\, \Sigma$
consts $\Sigma^{Rel} :: (\beta \times \gamma)\, \mathsf{set} \Rightarrow ((\alpha,\beta)\,\Sigma \times (\alpha,\gamma)\,\Sigma)\, \mathsf{set}$
$\Sigma^{Map} :: (\beta \Rightarrow \gamma) \Rightarrow (\alpha,\beta)\,\Sigma \Rightarrow (\alpha,\gamma)\,\Sigma$

Using this, the action $\Sigma^{Pred} :: \beta\, \mathsf{set} \Rightarrow ((\alpha,\beta)\,\Sigma)\, \mathsf{set}$ of Σ on predicates over β can be defined by $\Sigma^{Pred} \equiv mono^P \circ \Sigma^{Rel} \circ mono^E$, where $mono^E :: \alpha\, \mathsf{set} \Rightarrow (\alpha \times \alpha)\, \mathsf{set}$ is the embedding of predicates into relations in form of monotypes, and $mono^P :: (\alpha \times \alpha)\, \mathsf{set} \Rightarrow \alpha\, \mathsf{set}$ the corresponding projection. Furthermore, using Σ^{Rel} we define the extension $\simeq_\Sigma$ of our formal parameter $\simeq$ from *QuotientType-Param* simply by $\simeq_\Sigma \equiv \Sigma^{Rel} \simeq$.

Finally, the rule connecting the actions of Σ is given by axiom (F1), where $R \backslash\!. \, S$ is defined to be $\{x \,|\, \forall a.\ (x,a) \notin R \vee (x,a) \in S\}$, i.e. it is a sort of factoring of the relation R through the relation S:

axiom (F1) : $\Sigma^{Pred}(\simeq \backslash\!.\ ker\, f) \quad \subseteq \quad \simeq_\Sigma \backslash\!.\ ker\,(\Sigma^{Map}\, f)$

(2) The parameter coalgebra. Next, we specify the constant c_{T} representing a Σ-coalgebra with the domain $\alpha\,\mathsf{T}$ satisfying property (F2), where $P :: (\alpha\,\mathsf{T})\,\mathsf{set}$ is an arbitrary predicate and $f^{-1}\langle S\rangle$ denotes the preimage of a function f under a predicate S, i.e. $\{x \mid f\ x \in S\}$:

const $c_{\mathsf{T}} :: \alpha\,\mathsf{T} \Rightarrow (\alpha, \alpha\,\mathsf{T})\,\Sigma$
axiom (F2) : $c_{\mathsf{T}}^{-1}\langle \Sigma^{Pred}\, P\rangle \subseteq P \quad \Longrightarrow \quad Univ(\alpha\,\mathsf{T}) \subseteq P$

The axiom (F2) is a slightly adapted characterisation of so-called Σ-*reductive* coalgebras, which can be found in [7]. It essentially ensures that the sequence $s_0, s_1 \circ s_0, s_2 \circ s_1 \circ s_0, \ldots$ with $s_0 = c_{\mathsf{T}}$ and $s_{n+1} = \Sigma^{Map}\, s_n$, is not infinite and reaches some fixed point $s_k \circ \ldots \circ s_0$ with $k \in \mathbb{N}$. Thus, it also captures an induction principle.

(3) The hylomorphism parameter. The higher-order constant *Fold* is required to return a hylomorphism *Fold A* from c_{T} to A for any $A \in FoldCond$:

$$\begin{array}{ll}\textbf{const} & \mathit{Fold} :: ((\alpha,\beta)\,\Sigma \Rightarrow \beta) \Rightarrow \alpha\,\mathsf{T} \Rightarrow \beta \\ \textbf{axiom} & (\mathrm{F3}) : A \in \mathit{FoldCond} \quad \Longrightarrow \quad \mathit{Fold}\,A = A \circ \Sigma^{Map}(\mathit{Fold}\,A) \circ c_{\mathsf{T}}\end{array}$$

The predicate *FoldCond* on Σ-algebras is completely unspecified at this point, and therefore can be arbitrarily instantiated whenever the tactic is applied.

(4) Transformation function. Finally, we require a transformation function satisfying the properties (F4) and (F5), where *TransformCond* is another Σ-algebra predicate for which merely (F6) is required:

$$\begin{array}{ll}\textbf{const} & \mathit{Transform} :: (\alpha, \alpha\,\mathsf{T})\,\Sigma \Rightarrow (\alpha, \alpha\,\mathsf{T})\,\Sigma \Rightarrow (\alpha, \alpha\,\mathsf{T})\,\Sigma \\ \textbf{axioms} & (\mathrm{F4}) : s \simeq t \quad \Longrightarrow \quad c_{\mathsf{T}}\,s \simeq_{\Sigma} \mathit{Transform}\,(c_{\mathsf{T}}\,s)\,(c_{\mathsf{T}}\,t) \\ & (\mathrm{F5}) : A \in \mathit{TransformCond} \quad \Longrightarrow \quad s \simeq t \quad \Longrightarrow \\ & \qquad A\,(\Sigma^{Map}\,(\mathit{Fold}\,A)\,(\mathit{Transform}\,(c_{\mathsf{T}}\,s)\,(c_{\mathsf{T}}\,t))) = \mathit{Fold}\,A\,t \\ & (\mathrm{F6}) : \mathit{TransformCond} \subseteq \mathit{FoldCond}\end{array}$$

Transform can be considered as a function transforming its second argument w.r.t. its first argument. The axiom (F5) essentially requires that if both arguments comprise images of two elements, which are in the $\simeq$ relation, then *Transform* respects the kernel of $A \circ \Sigma^{Map}\,(\mathit{Fold}\,A)$.

3.6 The Theory *Fold*

The operations and conditions, specified in *Fold-Param* are sufficient in order to derive the congruence property for *Fold A* for any Σ-algebra A, satisfying the transformation condition *TransformCond*. To this end, the theory *Fold* proves the following central property:

Theorem 1. $A \in \mathit{TransformCond} \Longrightarrow \mathit{Fold}\,A \in \mathit{congruence}_{\simeq}$

Proof. The condition $\mathit{Fold}\,A \in \mathit{congruence}_{\simeq}$ can be equivalently restated using the factoring operator by $\mathit{Univ}(\alpha\mathsf{T}) \subseteq \simeq \backslash_{\bullet}\, \mathit{ker}(\mathit{Fold}\,A)$, such that we can proceed by induction using the reductivity axiom (F2). Further, by monotonicity of the preimage operator and the axiom (F1) we have then to show

$$c_{\mathsf{T}}^{-1}\langle \simeq_{\Sigma} \backslash_{\bullet}\, \mathit{ker}\,(\Sigma^{Map}\,(\mathit{Fold}\,A))\rangle \quad \subseteq \quad \simeq \backslash_{\bullet}\, \mathit{ker}\,(\mathit{Fold}\,A)$$

Unfolding the definitions of the factoring and preimage operators, this yields the ultimate goal: $\mathit{Fold}\,A\,s = \mathit{Fold}\,A\,t$ for any s, t of type $\alpha\,\mathsf{T}$, such that $s \simeq t$ and

$$(\forall\, u)\; \frac{c_{\mathsf{T}}\,s \simeq_{\Sigma} u}{\Sigma^{Map}\,(\mathit{Fold}\,A)\,(c_{\mathsf{T}}\,s) = \Sigma^{Map}\,(\mathit{Fold}\,A)\,u} \tag{10}$$

hold. This can be shown as follows

$$\begin{aligned}\mathit{Fold}\,A\,s &= A\,(\Sigma^{Map}(\mathit{Fold}\,A)\,(c_{\mathsf{T}}\,s)) \\ &= A\,(\Sigma^{Map}\,(\mathit{Fold}\,A)\,(\mathit{Transform}\,(c_{\mathsf{T}}\,s)\,(c_{\mathsf{T}}\,t))) \\ &= \mathit{Fold}\,A\,t\end{aligned}$$

where the first step follows by axiom (F3), the second by instantiating u in (10) with $\mathit{Transform}\,(c_{\mathsf{T}}\,s)\,(c_{\mathsf{T}}\,t)$ provided by axiom (F4), and the third by axioms (F5), (F6) and the premise $s \simeq t$. □

$$\begin{tikzcd}
\mathit{QuotientType\text{-}Param} \arrow[r, hook] \arrow[d, "\tau_1"'] & \mathit{QuotientType} \arrow[r, hook] \arrow[d, dashed, "\tau_1^\sharp"'] & \mathit{Fold\text{-}Param} \arrow[r, hook] \arrow[d, "\tau_2"'] & \mathit{Fold} \arrow[d, dashed, "\tau_2^\sharp"'] & \\
\mathcal{T}_1 \arrow[r, hook, dashed] & \mathcal{T}_1^\sharp \arrow[r, hook] & \mathcal{T}_2 \arrow[r, hook, dashed] & \mathcal{T}_2^\sharp & \mathcal{T}_{Spec} \arrow[l, "\tau"']
\end{tikzcd}$$

Fig. 1. Applying the fold quotient design tactic

As the immediate consequence for the function $\mathit{Fold}^{\mathsf{T}/\simeq} :: ((\alpha, \beta)\ \Sigma \Rightarrow \beta) \Rightarrow \alpha\ \mathsf{T}/_{\simeq} \Rightarrow \beta$, we can finally derive from (6) via Theorem 1:

$$\frac{A \in \mathit{TransformCond}}{\mathit{Fold}^{\mathsf{T}/\simeq}\ A\ [s]_{\simeq} = \mathit{Fold}\, A\, s} \tag{11}$$

Taking for instance *foldr* for *Fold* and a list algebra A, interpreting # by a function f satisfying *TransformCond*, this means that $\mathit{foldr}^{\mathsf{list}/\simeq}\ A\ [x\#xs]_{\simeq}$ can always be replaced by $\mathit{foldr}\ A\ (x\#xs) = f\ x\ (\mathit{foldr}\ A\ xs)$, and thus by $f\ x\ (\mathit{foldr}^{\mathsf{list}/\simeq}\ A\ [xs]_{\simeq})$.

4 Applying the Design Tactic

In this section, the presented design tactic for quotients and hylomorphism extension will be applied in order to derive implementations of bags and finite sets from lists. Recall the structure of the design tactic from (2); to apply it to a given type, we proceed in the following steps (see Fig. 1):

(i) we first provide a theory $\mathcal{T}_1$ and a morphism $\tau_1 : \mathit{QuotientType\text{-}Param} \longrightarrow \mathcal{T}_1$ which instantiates the type constructor and equivalence relation;
(ii) by instantiating *QuotientType*, we obtain $\mathcal{T}_1^\sharp$ with the quotient type;
(iii) we now extend $\mathcal{T}_1^\sharp$ into a theory $\mathcal{T}_2$, such that we can provide a theory morphism $\tau_2 : \mathit{Fold\text{-}Param} \longrightarrow \mathcal{T}_2$ instantiating the parameters for *Fold*;
(iv) by instantiating *Fold*, we obtain the theory $\mathcal{T}_2^\sharp$ with the desired function over the quotient type and the instantiated of the fold equation (11);
(v) finally, the correctness w.r.t. some axiomatic specification $\mathcal{T}_{Spec}$ is established by constructing a theory morphism $\tau : \mathcal{T}_{Spec} \longrightarrow \mathcal{T}_2^\sharp$.

Note that in Isabelle the theories $\mathcal{T}_1, \mathcal{T}_1^\sharp, \mathcal{T}_2, \mathcal{T}_2^\sharp$ are constructed as intermediate development steps of a single theory extending some base theory (in the following examples this will be the theory *List*).

4.1 Specifying Finite Sets and Bags

The rôle of theory $\mathcal{T}_{Spec}$ from step (v) above will be played by the axiomatic theories *FiniteSet-Spec* and *Bag-Spec*.

The Theory *FiniteSet-Spec*. It specifies finite sets parameterised over the type of its elements as follows. The unary type constructor finite-set is declared, together with the following polymorphic operations on it satisfying axioms (S1)– (S6):

$$\begin{array}{llll}
\textbf{typedecl} & \alpha\ \mathsf{finite\text{-}set} & & \\
\textbf{consts} & \{\#\} & :: \alpha\ \mathsf{finite\text{-}set} & \text{- empty set} \\
 & \_ \lessdot \_ & :: \alpha \Rightarrow \alpha\ \mathsf{finite\text{-}set} \Rightarrow \mathsf{bool} & \text{- membership test} \\
 & \_ \oplus \_ & :: \alpha \Rightarrow \alpha\ \mathsf{finite\text{-}set} \Rightarrow \alpha\ \mathsf{finite\text{-}set} & \text{- insertion} \\
 & \_ \ominus \_ & :: \alpha\ \mathsf{finite\text{-}set} \Rightarrow \alpha \Rightarrow \alpha\ \mathsf{finite\text{-}set} & \text{- deletion} \\
 & \mathit{foldSet} & :: (\alpha \Rightarrow \beta \Rightarrow \beta) \Rightarrow \beta \Rightarrow \alpha\ \mathsf{finite\text{-}set} \Rightarrow \beta & \text{- fold} \\
\textbf{axioms} & \multicolumn{3}{l}{(\mathrm{S1}) : \neg\, a \lessdot \{\#\}} \\
 & \multicolumn{3}{l}{(\mathrm{S2}) : (a \lessdot b \oplus S) = (a = b \vee a \lessdot S)} \\
 & \multicolumn{3}{l}{(\mathrm{S3}) : (a \lessdot S \ominus b) = (a \neq b \wedge a \lessdot S)} \\
 & \multicolumn{3}{l}{(\mathrm{S4}) : (\forall a.\ (a \lessdot S) = (a \lessdot T)) \Longrightarrow S = T} \\
 & \multicolumn{3}{l}{(\mathrm{S5}) : \mathit{foldSet}\ f\ e\ \{\#\} = e} \\
 & \multicolumn{3}{l}{(\mathrm{S6}) : f \in \mathit{LeftCommuting} \Longrightarrow \neg\, x \lessdot S \Longrightarrow} \\
 & \multicolumn{3}{l}{\qquad\quad \mathit{foldSet}\ f\ e\ (x \oplus S) = f\ x\ (\mathit{foldSet}\ f\ e\ S)}
\end{array}$$

where $\mathit{LeftCommuting} \equiv \{f \mid \forall\ a\ b\ c.\ f\ a\ (f\ b\ c) = f\ b\ (f\ a\ c)\}$. In this specification only the last axiom ultimately eliminates arbitrary sets from the class of possible implementations of *FiniteSet-Spec*. In other words, without the last axiom the theory morphism, sending α finite-set to α set as well as $\{\#\}$ to $\emptyset$, $\lessdot$ to $\in$ and so on, is constructible.

On the other hand, *foldSet* allows us to define all the basic operations on finite sets, e.g. the cardinality of any finite set S is given by $\mathit{foldSet}\ (\lambda\, x\, N.\, N + 1)\ 0\ S$, and the union $S \sqcup T$ by $\mathit{foldSet}\ (\lambda\, x\, Y.\, x \oplus Y)\ S\ T$. Moreover, we can define the translation function $\mathit{toPred} :: \alpha\ \mathsf{finite\text{-}set} \Rightarrow \alpha\ \mathsf{set}$ by $\mathit{foldSet}\ (\lambda\, x\, P.\ \{x\} \cup P)\ \emptyset$, such that for any $S :: \alpha\ \mathsf{finite\text{-}set}$ and $x :: \alpha$, $x \lessdot S$ holds iff $x \in \mathit{toPred}\, S$ does. Further, we can prove that the translation is also injective, and so the range of *toPred*, which is of type $(\alpha\ \mathsf{set})\mathsf{set}$, defines exactly the subset of finite predicates, isomorphic to α finite-set.

The Theory *Bag-Spec*. It specifies finite multisets in the similar manner. Here, we introduce an unary type constructor α bag together with basically the same operations on it, except that the membership function has the type $\alpha \Rightarrow \alpha\,\mathsf{bag} \Rightarrow \mathsf{nat}$ and thus counts the occurrences of an element in a bag. For the insertion operation this means that we have the rules $a \lessdot a \oplus M = (a \lessdot M) + 1$ and $a \neq b \Longrightarrow a \lessdot b \oplus M = a \lessdot M$. The folding function is now consequently called *foldBag*, and has to satisfy the rule

$$f \in \mathit{LeftCommuting} \Longrightarrow \mathit{foldBag}\ f\ e\ (x \oplus M) = f\ x\ (\mathit{foldBag}\ f\ e\ M)$$

Similarly to finite sets, cardinality, union, intersection etc. are definable via *foldBag* in *Bag-Spec*.

4.2 Implementing Bags

The implementation of bags is on the type of α list from the Isabelle/HOL libraries. The central rôle will be played by the function $\mathit{count} :: \alpha \Rightarrow \alpha\,\mathsf{list} \Rightarrow \mathsf{nat}$, defined recursively as

$$\begin{aligned} &\mathit{count}\ a\ \mathit{Nil} &&= 0 \\ &\mathit{count}\ a\ (b\#xs) &&= \begin{cases} 1 + \mathit{count}\ xs & \text{if } a = b \\ \mathit{count}\ xs & \text{otherwise} \end{cases} \end{aligned}$$

Now, let $xs \sim ys \equiv (\forall a.\ \mathit{count}\ a\ xs = \mathit{count}\ a\ ys)$, be the equivalence relation on α list comprising the intersection of kernels of the family of functions $\langle \mathit{count}\ a\rangle_{a \in \mathit{Univ}(\alpha)}$. We can then define the following theory morphism (step (i) above)

thymorph *bag1* : *QuotientType-Param* $\longrightarrow$ *Bag*
type-map : $[\alpha\ \mathsf{T} \mapsto \alpha\ \mathsf{list}]$
op-map : $[\simeq\ \mapsto\ \sim]$

and instantiate the parameterised theory ⟨*QuotientType-Param*, *QuotientType*⟩

instantiate-theory *QuotientType* **by-thymorph** *bag1*
renames : $[\ \mathsf{T}/_{\simeq} \mapsto \mathsf{bag}]$

This extends the theory *Bag* (step (ii) above), introducing the new quotient type constructor $\mathsf{list}/_{\sim}$ as bag, together with the corresponding congruence predicate $\mathit{congruence}_{\sim} :: (\alpha\ \mathsf{list} \Rightarrow \beta)\ \mathsf{set}$ and extension function $\_^{\mathsf{bag}}$, corresponding to step (ii) above. This step also gives us the theory morphism $\mathit{bag1}^{\sharp}$: *QuotientType* $\longrightarrow$ *Bag*, i.e. $\tau_1^{\sharp}$ in Fig. 1. Using this morphism, the corresponding instances of the properties (3) − (7) can now be translated to *Bag* along $\mathit{bag1}^{\sharp}$ via the **translate-thm** command. It is then routine to prove

1. $\mathit{count}\ x \in \mathit{congruence}_{\sim}$ for any x (this is in fact trivial);
2. $(\mathit{class\text{-}of}_{\sim} \circ (x \# \_)) \in \mathit{congruence}_{\sim}$ for any x;
3. $(\mathit{class\text{-}of}_{\sim} \circ (\mathit{remove1}\ x)) \in \mathit{congruence}_{\sim}$ for any x, where $\mathit{remove1}\ x\ xs$ removes the first occurrence of x from the list xs, if any;

such that the extensions of these functions from α list to α bag give us the implementations for the operations $\_ \lessdot \_$, $\_ \oplus \_$, and $\_ \ominus \_$ from *Bag-Spec*, respectively; for example the insertion $x \oplus M$ is implemented by $(\mathit{class\text{-}of}_{\sim} \circ (x \# \_))^{\mathsf{bag}}\ M$. It remains to give an implementation for *foldBag*.

Deriving ***foldBag*.** In order to proceed with step (iii), i.e. to instantiate the parameterised theory ⟨*Fold-Param*, *Fold*⟩, we need to supply actual parameters for the formal parameters in *Fold-Param*. This corresponds to construction of τ_2 in Fig. 1. First of all, the formal type parameter $(\alpha, \beta)\ \Sigma$, representing a signature, is mapped to $\mathbf{1} + \alpha \times \beta$ (the list signature). Then the parameter constants are mapped as follows:

1. the action of $\mathbf{1} + \alpha \times \beta$ on relations is defined in the standard way by
$$\Sigma^{Rel}\ R\ \equiv\ \{\iota_L *, \iota_L *\} \cup \{(\iota_R(u, x), \iota_R(u, y)) \mid (x, y) \in R, u \in \mathit{Univ}(\alpha)\}$$
where Σ^{Map} is exactly the same as Σ_{list}, defined in Sect. 3.5;
2. the coalgebra parameter c_{T} is instantiated by the coalgebra $\mathit{out}_{\mathsf{list}}$;

3. the hylomorphism is essentially the *foldr*-function:

$$\begin{aligned} \mathit{Fold}\ A &\equiv \mathit{foldr}\ (\lambda v\ x.\ A(\iota_R(v,x)))\ A(\iota_L *) \\ \mathit{FoldCond} &\equiv \mathit{Univ}(\mathbf{1} + \alpha \times \beta \Rightarrow \beta) \quad \text{i.e. the same as } \mathit{True} \end{aligned}$$

4. Finally, the transformation and the transformation condition are defined by

$$\mathit{Transform}\ u\ v \equiv \begin{cases} \iota_R(x, \mathit{remove1}\ x\ (y \# ys)) & \text{if } u = \iota_R(x, xs) \\ & \text{and } v = \iota_R(y, ys) \\ v & \text{otherwise} \end{cases}$$

$$\mathit{TransformCond} \equiv \{A \mid \forall\, x\ y\ z.\ \hat{A}(x, \hat{A}(y,z)) = \hat{A}(y, \hat{A}(x,z))\}$$

   where $\hat{A} \overset{def}{=} A \circ \iota_R$. That is, *TransformCond* specifies the subset of algebras having the left-commutative property, i.e. *LeftCommuting* specified above.

We now need to show the proof obligations arising as instances of axioms (F1) – (F6). For instance, the reductivity property (F2) is proven by structural induction on lists, and the proof of (F5) (which is the most complicated) is based on an auxiliary lemma showing

$$\frac{A \in \mathit{TransformCond} \quad x\ \mathit{mem}\ xs}{\mathit{Fold}\ A\ (x \# (\mathit{remove1}\ x\ xs)) = \mathit{Fold}\ A\ xs}$$

where *mem* denotes the membership test on lists and which can be shown by induction as well. All other proofs mainly comprise unfolding of definitions and case distinctions. Ultimately, we obtain the theory morphism $\mathsf{bag2} : \mathsf{Fold\text{-}Param} \longrightarrow \mathsf{Bag}$ and the instantiation

instantiate-theory *Fold* **by-thymorph** *bag2*

which gives us the theory morphism $\mathsf{bag2}^\sharp : \mathsf{Fold} \longrightarrow \mathsf{Bag}$. Then, the central congruence property ([11]) for $\mathit{Fold}^{\mathsf{bag}}$ can be translated from *Fold* along $\mathsf{bag2}^\sharp$. Based on this, we define the function *foldBag*:

$$\mathit{foldBag}\ f\ e \equiv \mathit{Fold}^{\mathsf{bag}}\ A \quad \text{where } A\,x \overset{def}{=} \begin{cases} f\ u\ v & \text{if } x = \iota_R(u,v) \\ e & \text{otherwise} \end{cases}$$

Altogether, we complete the development with a step constructing a theory morphism from *Bag-Spec* to the current development, corresponding to step (v) above. The emerging proof obligations, i.e. instances of bag axioms, can be now simply shown by unfolding the definitions (e.g. *foldBag*), and applying the congruence properties (e.g. ([11])).

4.3 Implementing Finite Sets

Although the implementation of finite sets is considerably more complicated, it follows the same principle. The following development makes an intermediate step deriving the type of distinct lists, where any element occurs at most once.

Distinct lists. The theory *DList* of distinct lists starts with the definition of the function $Norm :: \alpha\ \mathsf{list} \Rightarrow \alpha\ \mathsf{list}$ by

$$\begin{aligned} &Norm\,Nil &&= Nil \\ &Norm\,(x\#xs) &&= x\#(removeAll\,x\ (Norm\,xs)) \end{aligned}$$

where $removeAll\,x\,xs$ removes all occurrences of x from the list xs. Let $\sim_{Norm}$ abbreviate *ker Norm*, i.e. the kernel relation of *Norm*. Then, the instantiation of *QuotientType* by the theory morphism, sending $\alpha\ \mathsf{T}$ to $\alpha\ \mathsf{list}$ and $\simeq$ to $\sim_{Norm}$, introduces the quotient type constructor dlist (using renaming $\mathsf{T}/_{\simeq} \mapsto \mathsf{dlist}$), the corresponding extension function $\_^{\mathsf{dlist}} :: (\alpha\ \mathsf{list} \Rightarrow \beta) \Rightarrow \alpha\ \mathsf{dlist} \Rightarrow \beta$ and the congruence predicate $congruence_{\sim_{Norm}} :: (\alpha\ \mathsf{list} \Rightarrow \beta)\ \mathsf{set}$. It is now not difficult to show that for any $x :: \alpha$ the functions

1. $x\ mem\ \_$,
2. $class\text{-}of_{\sim_{Norm}} \circ (x \# \_)$, and
3. $class\text{-}of_{\sim_{Norm}} \circ (removeAll\,x)$

are in $congruence_{\sim_{Norm}}$. Let mem^D, put^D and get^D denote their respective extensions to $\alpha\ \mathsf{dlist}$. Moreover, let $empty^D \equiv [Nil]_{\sim_{Norm}}$. The definition of *Norm* provides also another useful property:

$$xs \neq Nil \implies xs \sim_{Norm} ys \implies head\,xs = head\,ys$$

where *head* is a function satisfying the equation $head\,(x\#xs) = x$. So, we can extend *head* to $head^D :: \alpha\ \mathsf{dlist} \Rightarrow \alpha$ such that the proposition

$$xs \neq Nil \implies head^D\ [xs]_{\sim_{Norm}} = head\,xs$$

is derivable. Based on this, we further have the following central decompositional property of distinct lists:

$$ds \neq empty^D \implies ds = put^D\ h\ (get^D\ h\ ds) \quad \text{where } h \overset{def}{=} head^D\ ds$$

To derive a fold-hylomorphism for distinct lists from *foldr*, an application of the ⟨*Fold-Param*, *Fold*⟩ parametrisation is unnecessary. Instead, we can directly define

$$fold^D\ f\ e \equiv (foldr\ f\ e\ \circ\ Norm)^{\mathsf{list}/\sim_{Norm}}$$

and subsequently show

$$fold^D\ f\ e\ empty^D = e \tag{12}$$

$$\frac{\neg\ x\ mem^D\ ds}{fold^D\ f\ e\ (put^D\ x\ ds) = f\ x\ (fold^D\ f\ e\ ds)} \tag{13}$$

$$\frac{f \in LeftCommuting \quad x\ mem^D\ ds}{fold^D\ f\ e\ (put^D\ x\ (get^D\ x\ ds)) = fold^D\ f\ e\ ds} \tag{14}$$

These are the essential properties for the implementation of finite sets below.

The theory *FiniteSet*. The theory *FiniteSet* imports *DList* and defines the equivalence relation $\sim$ on distinct lists by $ds \sim ds' \equiv (\forall x.\, x\, mem^D\, ds = x\, mem^D\, ds')$. Thus, the theory morphism *fset1* : *QuotientType-Param* $\longrightarrow$ *FiniteSet*, sending $\alpha\, \mathsf{T}$ to $\alpha\, \mathsf{dlist}$ and $\simeq$ to $\sim$, provides the instantiation:

instantiate-theory *QuotientType* **by-thymorph** *fset1*
renames : $[\mathsf{T}/_{\simeq} \mapsto \mathsf{finite\text{-}set}]$

which gives us the new quotient type constructor $\mathsf{dlist}/_{\sim}$ as $\mathsf{finite\text{-}set}$ together with the extension function $\_^{\mathsf{finite\text{-}set}} :: (\alpha\, \mathsf{dlist} \Rightarrow \beta) \Rightarrow \alpha\, \mathsf{finite\text{-}set} \Rightarrow \beta$ and the congruence predicate $congruence_{\sim} :: (\alpha\, \mathsf{dlist} \Rightarrow \beta)\, \mathsf{set}$. Regarding the specification of finite sets, we can then prove the $\sim$-congruence properties of mem^D, put^D, and get^D:

1. $x\; mem^D\; \_ \in congruence_{\sim}$ for any x;
2. $(class\text{-}of_{\sim} \circ (put^D\; x)) \in congruence_{\sim}$ for any x;
3. $(class\text{-}of_{\sim} \circ (get^D\; x)) \in congruence_{\sim}$ for any x;

such that $(mem^D)^{\mathsf{finite\text{-}set}}$, $(put^D)^{\mathsf{finite\text{-}set}}$, and $(get^D)^{\mathsf{finite\text{-}set}}$ give us the implementations for the operations $\_ \lessdot \_$, $\_ \oplus \_$, and $\_ \ominus \_$ from *FiniteSet-Spec*, respectively.

We now turn to a derivation of *foldSet* from $fold^D$ using the parametrisation $\langle$*Fold-Param*, *Fold*$\rangle$. The formal type parameter $(\alpha, \beta)\, \Sigma$ is mapped to $\mathbf{1} + \alpha \times \beta$. The parameter constants are mapped as follows:

1. Since the signature instantiation is the same as in *Bag*, the corresponding actions on relations and mappings do not change;
2. The coalgebra parameter c_{T} is instantiated by the function c_{dlist}, defined by

$$c_{\mathsf{dlist}}\; ds \equiv \begin{cases} \iota_L * & \text{if } ds = empty^D \\ \iota_R(head^D\; ds, get^D\; (head^D\; ds)\; ds) & \text{otherwise} \end{cases}$$

3. The hylomorphism *Fold* is given by the $fold^D$-function:

$$Fold\; A \;\equiv\; fold^D\; (\lambda x\, v.\; A(\iota_R(x, v)))\; A(\iota_L *)$$

4. The transformation is defined by

$$Transform\; u\; v \equiv \begin{cases} \iota_R(x, get^D\; x\; (put^D\; y\; ds')) & \text{if } u = \iota_R(x, ds), v = \iota_R(y, ds') \\ v & \text{otherwise} \end{cases}$$

5. Both conditions *FoldCond* and *TransformCond* are defined as in *Bag*.

The proofs of the emerging proof obligations are also similar to those for bags in Sect. 4.2. The proof of (F5) is again the most complicated and uses the properties (12), (13), and (14). Finally, the subsequent instantiation of the theory *Fold* gives the corresponding $\sim$-instance of the congruence property (11) for the extended function $Fold^{\mathsf{finite\text{-}set}}$: for any $A \in TransformCond$, i.e. for any algebra having the left commutative property, the identity $\mathit{Fold}^{\textsf{finite-set}}\ A\ [s]_{\sim} = \mathit{Fold}\ A\ s$ holds. Thus, we define the function *foldSet*:

$$\mathit{foldSet}\ f\ e \equiv \mathit{Fold}^{\textsf{finite-set}}\ A \qquad \text{where } A\ x \overset{def}{=} \begin{cases} f\ u\ v \text{ if } x = \iota_R(u,v) \\ e \qquad \text{otherwise} \end{cases}$$

As the final step, the development is completed by constructing a theory morphism from the specification *FiniteSet-Spec* to the current development. The resulting proof obligations are now straightforward.

5 Conclusions

This paper has presented the formalisation of an abstract design tactic in Isabelle, which provides a way to define hylomorphisms on a quotient type. The design tactic has two parameter theories: first, the type and equivalence relation for the quotient, and second a functor representing a signature, a coalgebra and a transformation function, which providing the setting for a class of 'extensible' hylomorphisms, justified by Theorem 1. To apply the design tactic, concrete instantiations of the parameter theories have to be provided by giving instantiating theories and a morphism mapping the parameter theories. In our case, we have shown how to apply the design tactic for a systematical derivation of correct implementations of finite multisets and finite sets.

The formalisation presented here has used Isabelle; however, the development knowledge represented in the design tactic could be formalised in other theorem provers too, since it formalises conditions for folding over a quotiented type on an abstract level, and the constructions used in the formalisation can be found in most other theorem provers as well.

For future work, the tactic might be also further generalised: for example, we can capitalise on the fact that the type constructor Σ and two actions Σ^{Rel}, Σ^{Map} on relations and mappings form a *relator* [1], pointing to a possible formalisation already at the level of allegories, increasing the application area.

Further, [11] considers behavioural equivalence on algebras over the same signature w.r.t. a set *OBS* of observable types. From this point of view, the theories *Bag-Spec* and *FiniteSet-Spec* are data abstractions, since both specify classes of algebras, each closed under the behavioural equivalence where $OBS_{bags} \overset{def}{=} \{\textsf{nat}\}$ and $OBS_{sets} \overset{def}{=} \{\textsf{bool}\}$. Then the quotient tactic allows us to construct from algebras with lists as carrier, *Bag-Spec* and *FiniteSet-Spec* instances where the extensionality principle (axiom (S4)) additionally holds, introducing new quotient type. Future work includes examining further connections to the constructions in [11], like abstract and behaviour.

Acknowledgements. This research was supported by the German Research Foundation (DFG) under grants LU-707/2-1 and 2-2, and by the German Federal Ministry of Education and Research (BMBF) under grant 01 IM F02 A.

References

1. Bird, R., de Moor, O.: Algebra of Programing. Prentice Hall, Englewood Cliffs (1997)
2. Bortin, M., Johnsen, E.B., Lüth, C.: Structured formal development in Isabelle. Nordic Journal of Computing 13, 2–21 (2006)
3. Burstall, R.M., Goguen, J.A.: Putting theories together to make specifications. In: Proc. Fifth International Joint Conference on Artificial Intelligence IJCAI 1977, pp. 1045–1058 (1977)
4. Burstall, R.M., Goguen, J.A.: The semantics of CLEAR, a specification language. In: Bjorner, D. (ed.) Abstract Software Specifications. LNCS, vol. 86, pp. 292–332. Springer, Heidelberg (1980)
5. Chicli, L., Pottier, L., Simpson, C.: Mathematical quotients and quotient types in Coq. In: Geuvers, H., Wiedijk, F. (eds.) TYPES 2002. LNCS, vol. 2646, pp. 95–107. Springer, Heidelberg (2003)
6. Diaconescu, R., Futatsugi, K.: CafeOBJ Report. World Scientific, Singapore (1998)
7. Doornbos, H., Backhouse, R.C.: Induction and recursion on datatypes. In: Möller, B. (ed.) MPC 1995. LNCS, vol. 947, pp. 242–256. Springer, Heidelberg (1995)
8. Farmer, W.M., Guttman, J.D., Thayer, F.J.: Little theories. In: Kapur, D. (ed.) CADE 1992. LNCS, vol. 607, pp. 567–581. Springer, Heidelberg (1992)
9. Goguen, J.A.: A categorical manifesto. Tech. Rep. PRG-72, Oxford University Computing Laboratory, Programming Research Group, Oxford, England (1989)
10. Hofmann, M.: A simple model for quotient types. In: Dezani-Ciancaglini, M., Plotkin, G. (eds.) TLCA 1995. LNCS, vol. 902, pp. 216–234. Springer, Heidelberg (1995)
11. Hofmann, M., Sannella, D.: On behavioural abstraction and behavioural satisfaction in higher-order logic. Theoretical Computer Science 167, 3–45 (1996)
12. Homeier, P.V.: A design structure for higher order quotients. In: Hurd, J., Melham, T. (eds.) TPHOLs 2005. LNCS, vol. 3603, pp. 130–146. Springer, Heidelberg (2005)
13. Mossakowski, T., Autexier, S., Hutter, D.: Development graphs — proof management for structured specifications. Journal of Logic and Algebraic Programming 67(1-2), 114–145 (2006)
14. Mosses, P.D. (ed.): CASL Reference Manual. LNCS, vol. 2960. Springer, Heidelberg (2004)
15. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)
16. Nogin, A.: Quotient types: A modular approach. In: Carreño, V.A., Muñoz, C.A., Tahar, S. (eds.) TPHOLs 2002. LNCS, vol. 2410, pp. 263–280. Springer, Heidelberg (2002)
17. Paulson, L.C.: Defining functions on equivalence classes. ACM Trans. Comput. Log. 7(4), 658–675 (2006)
18. Sannella, D., Burstall, R.: Structured theories in LCF. In: Protasi, M., Ausiello, G. (eds.) CAAP 1983. LNCS, vol. 159, pp. 377–391. Springer, Heidelberg (1983)
19. Slotosch, O.: Higher order quotients and their implementation in Isabelle/HOL. In: Gunter, E.L., Felty, A.P. (eds.) TPHOLs 1997. LNCS, vol. 1275, pp. 291–306. Springer, Heidelberg (1997)
20. Smith, D.R., Lowry, M.R.: Algorithm theories and design tactics. Science of Computer Programming 14, 305–321 (1990)
21. Srinivas, Y.V., Jullig, R.: Specware: Formal support for composing software. In: Möller, B. (ed.) MPC 1995. LNCS, vol. 947, Springer, Heidelberg (1995)

Instantiation of SMT Problems Modulo Integers*

Mnacho Echenim and Nicolas Peltier

University of Grenoble (LIG, Grenoble INP/CNRS)
Mnacho.Echenim@imag.fr, Nicolas.Peltier@imag.fr

Abstract. Many decision procedures for SMT problems rely more or less implicitly on an instantiation of the axioms defining the theories under consideration, and differ by making use of the additional properties of each theory, in order to increase efficiency. We present a new technique for devising complete instantiation schemes on SMT problems over a combination of linear arithmetic with another theory $\mathcal{T}$. The method consists in first instantiating the arithmetic part of the formula, and then getting rid of the remaining variables in the problem by using an instantiation strategy which is complete for $\mathcal{T}$. We provide examples evidencing that not only is this technique generic (in the sense that it applies to a wide range of theories) but it is also efficient, even compared to state-of-the-art instantiation schemes for specific theories.

1 Introduction

Research in the domain of Satisfiability Modulo Theories focuses on the design of decision procedures capable of testing the satisfiability of ground formulas modulo a given background theory. Such satisfiability checks may arise as a subproblem during the task of proving a more general formula in, e.g., software verification or interactive theorem proving. The background theories under consideration may define usual mathematical objects such as linear arithmetic, or data structures such as arrays or lists. The tools that implement these decision procedures are named SMT solvers, and they are designed to be as efficient as possible. This efficiency is obtained thanks to a sophisticated combination of state-of-the-art techniques derived from SAT solving, and ad-hoc procedures designed to handle each specific theory (see, e.g., [7] for a survey).

The lack of genericity of these theory solvers may become an issue, as additional theories, either new ones or extensions of former ones, are defined. For instance, a programmer may wish to add new axioms to the usual theory of arrays to specify, e.g., dimensions, sortedness, or definition domains. A solution to this lack of genericity was investigated in [4,3], where a first-order theorem prover is used to solve SMT problems. Once it is showed that the theorem prover terminates on SMT problems for a given theory, it can be used as an SMT solver

* This work has been partly funded by the project ASAP of the French *Agence Nationale de la Recherche* (ANR-09-BLAN-0407-01).

for that theory, and no additional implementation is required. Also, under certain conditions such as variable-inactivity (see, e.g., [3,8]), the theorem prover can also be used as an SMT solver for a combination of theories at no further expense. However, first-order theorem provers are not capable of efficiently handling the potentially large boolean structures of SMT problems. A solution to this problem was proposed in [9], with an approach consisting of decomposing an SMT problem in such a way that the theorem prover does not need to handle its boolean part. But even with this technique, theorem provers do not seem capable to compete with state-of-the-art SMT solvers.

A new approach to handling the genericity issue consists in devising a general instantiation scheme for SMT problems. The principle of this approach is to instantiate the axioms of the theories so that it is only necessary to feed a *ground* formula to the SMT solver. The problem is then to find a way to instantiate the axioms as little as possible so that the size of the resulting formula does not blow up, and still retain completeness: the instantiated set of clauses must be satisfiable if and only if the original set is. Such an approach was investigated in [11], and an instantiation scheme was devised along with a syntactic characterization of theories for which it is refutationally complete. One theory that cannot be handled by this approach is the theory of *linear arithmetic*, which is infinitely axiomatized. Yet, this theory frequently appears in SMT problems, such as the problems on arrays with integer indices. Handling linear arithmetic is also a challenge in first-order theorem proving, and several systems have been designed to handle the arithmetic parts of the formulas in an efficient way (see, e.g., [15] or the calculus of [2], which derives from [6]).

In this paper, we devise an instantiation scheme for theories containing particular integer constraints. This scheme, together with that of [11], permits to test the satisfiability of an SMT problem over a combination of linear arithmetic with another theory, by feeding a ground formula to an SMT solver. We show the potential efficiency of this scheme by applying it to problems in the theory of *arrays with integer indices*, and we show that it can generate sets of ground formulas that are much smaller than the ones generated by the instantiation rule of [10]. To emphasize the genericity of our approach, we also use it to integrate arithmetic constraints into a decidable subclass of many-sorted logic.

The paper is organized as follows. After recalling basic definitions from automated theorem proving, we introduce the notion of $\mathbb{Z}$-clauses, which are a restriction of the abstracted clauses of [6,2], along with the inference system introduced in [2]. We define a way of instantiating integer variables in particular formulas, and show how to determine a set of terms large enough to ensure completeness of the instantiation technique on an SMT problem. We then prove that under some conditions which are fulfilled by the scheme of [11], completeness is retained after using the scheme to instantiate the remaining variables in the SMT problems. We conclude by showing how this combined scheme can be applied on concrete problems. Due to a lack of space, we did not include the proofs in this paper; the detailed proofs can all be found in a technical report available at `http://membres-lig.imag.fr/peltier/inst_la.pdf`. The information on the instantiation method for non integer variables is available in [11].

2 Preliminaries

We employ a many-sorted framework. Let $\mathcal{S}$ denote a set of sorts, containing in particular a symbol $\mathbb{Z}$ denoting integers. Every variable is mapped to a unique sort in $\mathcal{S}$ and every function symbol f is mapped to a unique profile of the form $\mathtt{s}_1 \times \ldots \times \mathtt{s}_n \to \mathtt{s}$, where $\mathtt{s}_1, \ldots, \mathtt{s}_n, \mathtt{s} \in \mathcal{S}$ (possibly with $n = 0$); the sort $\mathtt{s}$ is the *range* of the function f. Terms are built with the usual conditions of well-sortedness. The signature contains in particular the symbols $0, -, +$ of respective profiles $\to \mathbb{Z}$, $\mathbb{Z} \to \mathbb{Z}$, $\mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$. The terms $s^i(0)$, $t + s(0)$, $t + (-s(0))$ and $t + (-s)$ are abbreviated by $i, \mathrm{s}(t), \mathrm{p}(t)$ and $t - s$ respectively. Terms (resp. variables) of sort $\mathbb{Z}$ are called *integer terms* (resp. *integer variables*). A term is *ground* if it contains no variable. We assume that there exists at least one ground term of each sort and that for every function symbol of profile $\mathtt{s}_1 \times \ldots \times \mathtt{s}_n \to \mathbb{Z}$, we have $\mathtt{s}_i = \mathbb{Z}$ for all $i \in [1..n]$: integer terms may only have integer subterms.

An *atom* is either of the form $t \preceq s$ where t, s are two terms of sort $\mathbb{Z}$, or of the form $t \simeq s$ where t, s are terms of the same sort. An atom[1] $t \bowtie s$ is *arithmetic* if t, s are of sort $\mathbb{Z}$. A *clause* is an expression of the form $\Gamma \to \Delta$, where Γ, Δ are sequences of non-arithmetic atoms. A *substitution* σ is a function mapping every variable x to a term $x\sigma$ of the same sort. Substitution σ is *ground* if for every variable x in the domain of σ, $x\sigma$ is ground. For any expression $\mathcal{E}$ (term, atom, sequence of atoms or clause), $V(\mathcal{E})$ is the set of variables occurring in $\mathcal{E}$ and $\mathcal{E}\sigma$ denotes the expression obtained by replacing in $\mathcal{E}$ every variable x in the domain of σ by the term $x\sigma$. Interpretations are defined as usual. A $\mathbb{Z}$-*interpretation* I is an interpretation such that the domain of sort $\mathbb{Z}$ is the set of integers, and that the interpretation of the symbols $0, -, +$ is defined as follows: $I(0) = 0, I(t + s) = I(t) + I(s)$ and $I(-t) = -I(t)$. A ground atom A is *satisfied* by an interpretation I if either A is of the form $t \preceq s$ and $I(t) \leq I(s)$ or A is of the form $t \simeq s$ and $I(t) = I(s)$. A clause $\Gamma \to \Delta$ is *satisfied* by an interpretation I if for every ground substitution σ, either there exists an atom $A \in \Gamma\sigma$ that is **not** satisfied by I, or there exists an atom $A \in \Delta\sigma$ that is satisfied by I. A set of clauses S is *satisfied* by I if I satisfies every clause in S. As usual, we write $I \models S$ if S is satisfied by I and $S_1 \models S_2$ if every interpretation that satisfies S_1 also satisfies S_2. S_1 and S_2 are *equivalent* if $S_1 \models S_2$ and $S_2 \models S_1$. We note $I \models_{\mathbb{Z}} S$ if I is a $\mathbb{Z}$-interpretation that satisfies S; $S_1 \models_{\mathbb{Z}} S_2$ if every $\mathbb{Z}$-interpretation satisfying S_1 also satisfies S_2, and S_1, S_2 are $\mathbb{Z}$-*equivalent* if $S_1 \models_{\mathbb{Z}} S_2$ and $S_2 \models_{\mathbb{Z}} S_1$.

We assume the standard notions of positions in terms, atoms and clauses. As usual, given two terms t and s, $t|_p$ is the subterm occurring at position p in t and $t[s]_p$ denotes the term obtained from t by replacing the subterm at position p by s. Given an expression $\mathcal{E}$ (term, atom, clause...), a position p is a *variable position in* $\mathcal{E}$ if $\mathcal{E}|_p$ is a variable.

The *flattening* operation on a set of clauses S consists in replacing non constant ground terms t occurring in S by fresh constants c, and adding to S the unit clause $t \simeq c$. We refer the reader to, e.g., [4] for more details.

[1] The symbol $\bowtie$ represents either $\simeq$ or $\preceq$.

3 $\mathbb{Z}$-Clauses

We introduce the class of $\mathbb{Z}$-clauses. These are restricted versions of the abstracted clauses of [6,2], as we impose that the arithmetic constraints be represented by atoms, and not literals. We add this restriction for the sake of readability; in fact it incurs no loss of generality: for example, a literal $\neg(a \preceq b)$ can be replaced by the $\mathbb{Z}$-equivalent arithmetic atom $b \preceq \mathrm{p}(a)$. We present some terminology from [2], adapted to our setting.

Definition 1. *A* $\mathbb{Z}$-clause *is an expression of the form $\Lambda \,\|\, \Gamma \rightarrow \Delta$, where:*

- *Λ is a sequence of arithmetic atoms (the* arithmetic part *of $\Lambda \,\|\, \Gamma \rightarrow \Delta$);*
- *$\Gamma \rightarrow \Delta$ is a clause such that every integer term occurring in Γ or in Δ is a variable*[2]*.*

The property that in a $\mathbb{Z}$-clause $\Lambda \,\|\, \Gamma \rightarrow \Delta$, every integer term occurring in Γ or in Δ is a variable is simple to ensure. If this is not the case, i.e., if Γ, Δ contain an integer term t that is not a variable, then it suffices to replace every occurrence of t with a fresh integer variable u, and add the equation $u \simeq t$ to Λ. This way every set of clauses can be transformed into an equivalent set of $\mathbb{Z}$-clauses.

The notions of position, replacement, etc. extend straightforwardly to sequences of atoms and $\mathbb{Z}$-clauses, taking them as terms with 3 arguments. The notion of satisfiability is extended to $\mathbb{Z}$-clauses as follows:

Definition 2. *A substitution σ is a* solution *of a sequence of arithmetic atoms Λ in an interpretation I if σ maps the variables occurring in Λ to integers such that $I \models \Lambda\sigma$. A $\mathbb{Z}$-clause $\Lambda \,\|\, \Gamma \rightarrow \Delta$ is* satisfied *by an interpretation I if for every solution σ of Λ, the clause $(\Gamma \rightarrow \Delta)\sigma$ is satisfied by I.*

Note that, although the signature may contain uninterpreted symbols of sort $\mathbb{Z}$ (e.g. constant symbols that must be interpreted as integers), it is sufficient to instantiate the integer variables by integers only.

Definition 3. *Given a $\mathbb{Z}$-clause $C = \Lambda \,\|\, \Gamma \rightarrow \Delta$, an* abstraction atom *in C is an atom of the form $x \simeq t$ which occurs in Λ. $x \simeq t$ is* grounding *if t is ground. C is* $\mathbb{Z}$-closed *if all its integer variables occur in grounding abstraction atoms and* closed *if it is $\mathbb{Z}$-closed and every variable occurring in C is of sort $\mathbb{Z}$.*

Intuitively, if C is $\mathbb{Z}$-closed, this means that C would not contain any integer variable, had integer terms not been abstracted out. We define an operation permitting to add arithmetic atoms to a $\mathbb{Z}$-clause:

Definition 4. *Consider a $\mathbb{Z}$-clause $C = \Lambda \,\|\, \Gamma \rightarrow \Delta$ and a set of arithmetic atoms Λ'. We denote by $[\Lambda', C]$ the $\mathbb{Z}$-clause $\Lambda', \Lambda \,\|\, \Gamma \rightarrow \Delta$.*

An Inference System for $\mathbb{Z}$-Clauses. We denote by $\mathcal{H}$ the inference system of [2] on abstracted clauses, depicted in Figure 1. Reduction rules are also defined in [2]; the only one that is useful in our context is the *tautology deletion* rule also

Superposition left : $\dfrac{\Lambda_1 \,\|\, \Gamma_1 \rightarrow \Delta_1, l \simeq r \quad \Lambda_2 \,\|\, s[l'] \simeq t, \Gamma_2 \rightarrow \Delta_2}{(\Lambda_1, \Lambda_2 \,\|\, s[r] \simeq t, \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2)\sigma}$

where σ is an mgu of l and l', $l\sigma \not\preceq r\sigma$, $s\sigma \not\preceq t\sigma$, l' is not a variable, $(l \simeq r)\sigma$ is strictly maximal in $(\Gamma_1 \rightarrow \Delta_1, l \simeq r)\sigma$ and $(s[l'] \simeq t)\sigma$ is strictly maximal in $(s[l'] \simeq t, \Gamma_2 \rightarrow \Delta_2)\sigma$.

Superposition right : $\dfrac{\Lambda_1 \,\|\, \Gamma_1 \rightarrow \Delta_1, l \simeq r \quad \Lambda_2 \,\|\, \Gamma_2 \rightarrow \Delta_2, s[l'] \simeq t}{(\Lambda_1, \Lambda_2 \,\|\, \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2, s[r] \simeq t)\sigma}$

where σ is an mgu of l and l', $l\sigma \not\preceq r\sigma$, $s\sigma \not\preceq t\sigma$, l' is not a variable, $(l \simeq r)\sigma$ is strictly maximal in $(\Gamma_1 \rightarrow \Delta_1, l \simeq r)\sigma$ and $(s[l'] \simeq t)\sigma$ is strictly maximal in $\Gamma_2 \rightarrow \Delta_2, (s[l'] \simeq t)\sigma$.

Equality factoring : $\dfrac{\Lambda \,\|\, \Gamma \rightarrow \Delta, l \simeq r, l' \simeq r'}{(\Lambda \,\|\, \Gamma, r \simeq r' \rightarrow \Delta, l' \simeq r')\sigma}$

where σ is an mgu of l and l', $l\sigma \not\preceq r\sigma$, $l'\sigma \not\preceq r'\sigma$ and $(l \simeq r)\sigma$ is maximal in $(\Gamma_1 \rightarrow \Delta_1, l \simeq r, l' \simeq r')\sigma$.

Ordered factoring : $\dfrac{\Lambda \,\|\, \Gamma \rightarrow \Delta, E_1, E_2}{(\Lambda \,\|\, \Gamma \rightarrow \Delta, E_1)\sigma}$

where σ is an mgu of E_1 and E_2, and $E_1\sigma$ is maximal in $(\Gamma \rightarrow \Delta, E_1, E_2)\sigma$.

Equality resolution : $\dfrac{\Lambda \,\|\, \Gamma, s \simeq t \rightarrow \Delta}{(\Lambda \,\|\, \Gamma \rightarrow \Delta)\sigma}$

where σ is an mgu of s and t, and $(s \simeq t)\sigma$ is maximal in $(\Gamma, s \simeq t \rightarrow \Delta)\sigma$.

Constraint refutation : $\dfrac{\Lambda_1 \,\|\, \rightarrow \qquad \cdots \qquad \Lambda_n \,\|\, \rightarrow}{\square}$

where $\Lambda_1 \,\|\, \rightarrow \wedge \cdots \wedge \Lambda_n \,\|\, \rightarrow$ is inconsistent in $\mathbb{Z}$.

As usual the system is parameterized by an ordering among terms, extended into an ordering on atoms and clauses (see [5] for details). The rules are applied modulo the AC properties of the sequences and the commutativity of $\simeq$.

Tautology deletion : $\dfrac{\Lambda \,\|\, \Gamma \rightarrow \Delta}{\overline{}}$,

if $\Gamma \rightarrow \Delta$ is a tautology, or the existential closure of Λ is $\mathbb{Z}$-unsatisfiable.

Fig. 1. The inference system $\mathcal{H}$

depicted in Figure 1. We make the additional (and natural) assumption that the ordering is such that all constants are smaller than all non-flat terms. In order to obtain a refutational completeness result on this calculus, the authors of [6,2] impose the condition of *sufficient completeness* on sets of clauses. Without this condition, we have the following result, stating a weaker version of refutational completeness for the calculus.

Theorem 1. *Let S denote a $\mathbb{Z}$-unsatisfiable set of $\mathbb{Z}$-clauses. Then there exists a $\mathbb{Z}$-unsatisfiable set of clauses $\{\Lambda_i \,\|\, \rightarrow \mid i \in \mathbb{N}\}$ such that for every $i \in \mathbb{N}$, $\Lambda_i \,\|\, \rightarrow$ can be deduced from S by applying the rules in $\mathcal{H}$.*

Note that this does *not* imply refutational completeness, since the set $\{\Lambda_i \,\|\, \rightarrow \mid i \in \mathbb{N}\}$ may be infinite (if this set is finite then the **Constraint refutation** rule

[2] Recall that by definition a clause cannot contain arithmetic atoms.

applies and generates $\square$). For instance, the set of $\mathbb{Z}$-clauses $S = \{x \simeq a \,\|\, p(x) \to, \; x \simeq s(y) \,\|\, p(x) \to p(y),\; p(0),\; a < 0 \,\|\, \to\}$ is clearly unsatisfiable, and the calculus generates an infinite number of clauses of the form $\mathrm{s}^k(0) \simeq a \,\|\, \to$, for $k \in \mathbb{N}$. It is actually simple to see that there is no refutationally complete calculus for sets of $\mathbb{Z}$-clauses, since we explicitly assume that $\mathbb{Z}$ is interpreted as the set of integers. In our case however there are additional conditions on the arithmetic constraints that ensure that only a finite set of $\mathbb{Z}$-clauses of the form $\Lambda \,\|\, \to$ will be generated. Thus, for the $\mathbb{Z}$-clauses we consider, refutational completeness of the calculus will hold, and it will always generate the empty clause starting from an unsatisfiable set of $\mathbb{Z}$-clauses. However, we do not intend to use this result to test the satisfiability of the formulas. The reason is that – as explained in the introduction – the superposition calculus is not well adapted to handle efficiently very large propositional formulas. In this paper, we use the inference system $\mathcal{H}$ only as a theoretical tool to show the *existence* of an instantiation scheme.

4 Instantiation of Inequality Formulas

Given an SMT problem over a combination of a given theory with the theory of linear arithmetic, the inference system of [2] permits to separate the reasoning on the theory itself from the reasoning on the arithmetic part of the formula. If the input set of clauses is unsatisfiable, then the inference system will generate a set of clauses of the form $\{\Lambda_1 \,\|\, \to, \ldots, \Lambda_n \,\|\, \to, \ldots\}$, which is inconsistent in $\mathbb{Z}$. In this section, we investigate how to safely instantiate the Λ_i's, under some condition on the atoms they contain. We shall impose that each Λ_i be equivalent to a formula of the following form:

Definition 5. *An* inequality formula *is of the form* $\phi : \bigwedge_{i=1}^m s_i \preceq t_i$*, where for all* $i = 1, \ldots, m$*,* s_i *and* t_i *are ground terms or variables.*

If A is a set of terms, we use the notation $A \preceq x$ (resp. $x \preceq A$) as a shorthand for $\bigwedge_{s \in A} s \preceq x$ (resp. $\bigwedge_{s\in A} x \preceq s$). We denote by U_x^ϕ the set of variables $U_x^\phi = \{y \in V(\phi) \mid x \preceq y \text{ occurs in } \phi\}$. We may thus rewrite the formula ϕ as $\bigwedge_{x \in V(\phi)} (A_x^\phi \preceq x \wedge x \preceq B_x^\phi \wedge \bigwedge_{y \in U_x^\phi} x \preceq y) \wedge \psi$, where the sets A_x^ϕ and B_x^ϕ are ground for all x, and ψ only contains inequalities between ground terms.

Definition 6. *For all* $x \in V(\phi)$*, we consider the sets* $\bar{B}_x^\phi$*, defined as the smallest sets satisfying* $\bar{B}_x^\phi \supseteq B_x^\phi \cup \bigcup_{y \in U_x^\phi} \bar{B}_y^\phi \cup \{\chi\}$*, where* χ *is a special constant that does not occur in* ϕ*.*

Theorem 2. *Given an inequality formula* ϕ *such that* $V(\phi) = \{x_1, \ldots, x_n\}$ *consider the two following formulas:*

$$[\exists x_1 \cdots \exists x_n . \phi] \qquad (\alpha)$$
$$\left(\bigvee_{s_1 \in \bar{B}_{x_1}^\phi} \cdots \bigvee_{s_n \in \bar{B}_{x_n}^\phi} \phi\{x_i \leftarrow s_i \mid i = 1, \ldots, n\}\right) \quad (\beta)$$

Let I *denote a* $\mathbb{Z}$*-interpretation of* (α) *and* G *denote a ground set containing all ground terms occurring in* ϕ*. Then* $I \models_{\mathbb{Z}} (\alpha)$ *if and only if, for any extension* J *of* I *to the constant* χ*,* $J \models_{\mathbb{Z}} \left(\bigwedge_{t \in G} \neg(\chi \preceq t)\right) \Rightarrow (\beta)$*.*

In our case, the sets $B_{x_i}^{\phi}$ will not be known, since the clauses in which ϕ occurs will not be generated explicitly (see Section 5 for details). Thus we need to use an *over-approximation* of these sets:

Definition 7. *A set of ground terms B is an* upper bound *of an inequality formula ϕ if for all atoms $x \preceq t$ occurring in ϕ, t is an element of B. The set B is an* upper bound *of a set of inequality formulas if it is an upper bound of each formula.*

It is clear that if B is an upper bound of an inequality formula, then Theorem 2 still holds when the variables in ϕ are instantiated by all the elements in $B \uplus \{\chi\}$ instead of just those in the $\bar{B}_x^{\phi}$'s. We define B-definitions which represent grounding instantiations using abstraction atoms:

Definition 8. *Given an inequality formula ϕ such that $V(\phi) = \{x_1, \ldots, x_m\}$ and a set of ground terms B, a B-*definition *of ϕ is a set (i.e. a conjunction) of grounding abstraction atoms $\{x_i \simeq s_i \mid i = 1, \ldots, m\}$, such that every s_i is in B. We denote by $\Theta_B[\phi]$ the set of all B-definitions of ϕ.*

We rephrase a direct consequence of Theorem 2 using B-definitions:

Corollary 1. *Let $\{\phi_1, \ldots, \phi_n\}$ denote a set of inequality formulas over the disjoint sets of variables $\{x_{i,1}, \ldots, x_{i,m_i} \mid i = 1 \ldots, n\}$, let B denote an upper bound of this set, and assume that $\bigvee_{i=1}^{n} \exists x_{i,1} \cdots \exists x_{i,m_i} . \phi_i$ is valid in $\mathbb{Z}$. If Gcontains all ground terms occurring in the inequality formulas and $B' = B \uplus \{\chi\}$, then*

$$\bigwedge_{t \in G} \neg(\chi \preceq t) \Rightarrow \bigvee_{i=1}^{n} \left(\bigvee_{\Lambda_i' \in \Theta_{B'}[\phi_i]} \exists x_{i,1} \cdots \exists x_{i,m_i} . \phi_i \wedge \Lambda_i' \right) \qquad (\gamma),$$

is also valid in $\mathbb{Z}$.

It is important to note that results similar to those of this section could have been proved by considering the terms occurring in atoms of the form $t \preceq x$, instead of those of the form $x \preceq t$, and considering lower bounds instead of upper bounds. This should allow to choose, depending on the problem and which sets are smaller, whether to instantiate variables using lower bounds or upper bounds.

5 Properties of Inferences on $\mathbb{Z}$-Clauses

Corollary 1 shows how to safely get rid of integer variables in a set of inequality formulas, provided an upper bound of this set is known. The goal of this section is first to show that given an initial set of $\mathbb{Z}$-clauses S, such an upper bound can be determined, regardless of the inequality formulas that can be generated from S. Then we show that by instantiating the integer variables in S, it is still possible to generate all necessary instances of the inequality formulas. Thus, S and the corresponding instantiated set will be equisatisfiable.

We first define a generalization of the notion of an upper bound of an inequality formula (see Definition 7), to the case of $\mathbb{Z}$-clauses.

Definition 9. *Given a $\mathbb{Z}$-clause $C = \Lambda \,\|\, \Gamma \rightarrow \Delta$ and a set of ground terms B, we write $C \trianglelefteq B$ if for all atoms $x \preceq t \in \Lambda$,*

- *Λ contains (not necessarily distinct) grounding abstraction atoms of the form $x_i \simeq s_i$, $i = 1, \dots, n$;*
- *there exist variable positions $\{p_1, \dots, p_n\}$ such that variable x_i occurs at position p_i, and $t[s_1]_{p_1} \dots [s_n]_{p_n} \in B$.*

Example 1. Let $C = x \simeq a, y \simeq b, y \simeq c, z \preceq f(g(x,y),y) \,\|\, \rightarrow h(x,y,z) \simeq d$ and $B = \{f(g(a,c),b)\}$. Then $C \trianglelefteq B$.

Intuitively, for a $\mathbb{Z}$-clause $C = \Lambda \,\|\, \Gamma \rightarrow \Delta$, the set B is an upper bound of the inequality atoms in Λ provided for all atoms $x \preceq t$, the variables in t are replaced by the correct terms. In order not to unnecessarily instantiate some of the integer variables in a $\mathbb{Z}$-clause, we distinguish those that appear in abstraction atoms from those that appear in inequality atoms. It will only be necessary to instantiate the latter variables.

Definition 10. *Let $C = \Lambda \,\|\, \Gamma \rightarrow \Delta$; the set of* abstraction variables in C *$V_{\text{abs}}(C)$ and the set of* inequality variables in C *$V_{\text{ineq}}(C)$ are defined as follows: $V_{\text{abs}}(C) = \{x \in V(C) \,|\, \Lambda$ contains an abstraction atom $x \simeq t\}$ and $V_{\text{ineq}}(C) = \{x \in V(C) \,|\, \Lambda$ contains an atom of the form $x \preceq t$ or $t \preceq x\}$*

We may assume without loss of generality that all integer variables in a $\mathbb{Z}$-clause C are in $V_{\text{abs}}(C) \cup V_{\text{ineq}}(C)$. If this is not the case, it suffices to add to the arithmetic part of C the atom $x \preceq x$.

We define the notion of a *preconstrained $\mathbb{Z}$-clause*. If a preconstrained $\mathbb{Z}$-clause is of the form $\Lambda \,\|\, \rightarrow$, then Λ will be equivalent to an inequality formula, and this property is preserved by inferences.

Definition 11. *A $\mathbb{Z}$-clause $C = \Lambda \,\|\, \Gamma \rightarrow \Delta$ is* preconstrained *if every atom in Λ that is not a grounding abstraction atom either has all its variables in $V_{\text{abs}}(C)$, or is of the form $x \preceq t$ or $t \preceq x$, where t is either a variable itself, or has all its variables in $V_{\text{abs}}(C)$.*

Example 2. $x \simeq a, y \simeq b, f(x,y) \simeq g(y), z \preceq g(x) \,\|\, \rightarrow h(x,y,z) \simeq e$ is preconstrained but $x \simeq a, y \preceq g(y) \,\|\, \rightarrow h(x,y,z) \simeq e$ is not because y does not occur in a grounding abstraction atom.

There remains to extend the notion of a B-definition to $\mathbb{Z}$-clauses. Intuitively, a B-definition of such a $\mathbb{Z}$-clause represents a ground instantiation of the inequality variables it contains.

Definition 12. *Given a $\mathbb{Z}$-clause C such that $V_{\text{ineq}}(C) = \{x_1, \dots, x_m\}$ and a set of ground terms B, a* B-definition of C *is a set of grounding abstraction atoms $\{x_i \simeq s_i \,|\, i = 1, \dots, m\}$, such that every s_i is in B. We denote by $\Theta_B[C]$ the set of all B-definitions of C. Given a set of $\mathbb{Z}$-clauses S, we denote by S_B the set $S_B = \{[\Lambda', C] \,|\, C \in S \wedge \Lambda' \in \Theta_B[C]\}$.*

We obtain the main result of this section:

Theorem 3. *Suppose S is a set of $\mathbb{Z}$-clauses and B is a set of ground terms such that for every $\mathbb{Z}$-clause of the form $C = \Lambda \,\|\, \to$ generated from S, C is preconstrained and $C \trianglelefteq B$. Let $B' = B \cup \{\chi\}$, where χ does not occur in S. Then there exists a set of ground terms G containing B and all ground terms of sort $\mathbb{Z}$ in S such that S is $\mathbb{Z}$-satisfiable if and only if $S_{B'} \cup \bigcup_{t \in G}\{\chi \preceq t \,\|\, \to\}$ is $\mathbb{Z}$-satisfiable.*

In particular, since we may assume all the integer variables occurring in S are in a $V_{\text{abs}}(C) \cup V_{\text{ineq}}(C)$ for some $C \in S$, every $\mathbb{Z}$-clause occurring in $S_{B'}$ can be reduced to a $\mathbb{Z}$-clause that is $\mathbb{Z}$-closed, and $S_{B'} \cup \bigcup_{t \in G}\{\chi \preceq t \,\|\, \to\}$ can be reduced to a set of clauses containing no integer variable. Hence, Theorem 3 provides a way of getting rid of all integer variables in a formula.

The instantiated set $S_{B'} \cup \bigcup_{t \in G}\{\chi \preceq t \,\|\, \to\}$ can further be reduced: since χ is strictly greater than any ground term t occurring in S or in B, every atom of the form $\chi \preceq t$ or $t \preceq \chi$ can be replaced by false and true respectively. Furthermore, by construction χ only appears at the root level in the arithmetic terms. Thus we can safely assume that χ does not occur in the arithmetic part of the $\mathbb{Z}$-clause in $S_{B'}$. This implies that the inequations $\chi \preceq t \,\|\, \to$ for $t \in G$ are useless and can be removed; thus, the resulting set does not depend on G.

6 Completeness of the Combined Instantiation Schemes

The aim of this section is to determine sufficient conditions guaranteeing that once the integer variables have been instantiated, another instantiation scheme can be applied to get rid of the remaining variables in the set of clauses under consideration.

Let $\mathcal{C}$ denote a class of clause sets admitting an instantiation scheme, i.e., a function γ that maps every clause set $S \in \mathcal{C}$ to a finite set of ground instances of clauses in S, such that S is satisfiable if and only if $\gamma(S)$ is satisfiable. If $\gamma(S)$ is finite, this implies that the satisfiability problem is decidable for $\mathcal{C}$. Since γ is generic, we cannot assume that it preserves $\mathbb{Z}$-satisfiability. In order to apply it in our setting, we need to make additional assumptions on the instantiation scheme under consideration.

Definition 13. *A term t is* independent *from a set of clauses S if for every non-variable term s occurring in S, if t and s are unifiable, then $t = s$. An instantiation scheme γ is* admissible *if:*

1. *It is is monotonic, i.e. $S \subseteq S' \Rightarrow \gamma(S) \subseteq \gamma(S')$.*
2. *If S is a set of clauses and t, s are two terms independent from S then $\gamma(S \cup \{t \simeq s\}) = \gamma(S) \cup \{t \simeq s\}$.*

The first requirement is fairly intuitive, and is fulfilled by every instantiation procedure of our knowledge. The second one states that adding equalities between particular terms should not influence the instantiation scheme. Generic instantiation schemes such as those in [16,17,12] do not satisfy the second requirement; however, it is fulfilled by the one of [11].

From now on, we assume that γ denotes an admissible instantiation scheme. We show how to extend γ to $\mathbb{Z}$-closed sets of $\mathbb{Z}$-clauses. Such $\mathbb{Z}$-clauses are obtained as the output of the scheme devised in the previous section.

Definition 14. *A set of clauses $S = \{\Lambda_i \,\|\, C_i \mid i \in [1..n]\}$ where Λ_i is a sequence of ground arithmetic atoms and C_i is a clause is γ-*compatible *if $S' = \{C_1, \ldots, C_n\} \in \mathcal{C}$.*

Theorem 4. *Let $S = \{\Lambda_i \,\|\, C_i \mid i \in [1..n]\}$ denote a γ-compatible set of $\mathbb{Z}$-clauses. Let χ denote a constant not occurring in the arithmetic part of the clauses in S or in the scope of a function of range $\mathbb{Z}$ in S, and consider a set G of ground integer terms such that χ occurs in no term in G.*

Then $S \cup \bigcup_{t \in G}\{\chi \preceq t \,\|\, \to\}$ is $\mathbb{Z}$-satisfiable if and only if $\gamma(S)$ is $\mathbb{Z}$-satisfiable.

Summary. To summarize, starting from a set of $\mathbb{Z}$-clauses S:

1. The scheme devised in Section 5 is applied to instantiate all integer variables occurring in S. We obtain a $\mathbb{Z}$-closed set of $\mathbb{Z}$-clauses S'.
2. S' is processed to get rid of all clauses containing arithmetic atoms of the form $\chi \preceq t$, and to get rid of all atoms of the form $t \preceq \chi$ in the remaining clauses. We obtain a set of $\mathbb{Z}$-clauses S''.
3. Then we apply an admissible instantiation scheme (e.g., [11]) γ on the clausal part of the $\mathbb{Z}$-clauses in S'' to instantiate all remaining variables. We obtain a set of closed $\mathbb{Z}$-clauses S_g.
4. Finally we feed an SMT-solver (that handles linear arithmetic) with S_g.

The previous results ensure that S and S_g are equisatisfiable, provided S'' is compatible with γ. This means that the procedure can be applied to test the satisfiability of an SMT problem on the combination of linear arithmetic with, e.g., *any* of the theories that the scheme of [11] is capable of handling, which include the theories of arrays, records, or lists. Note that an efficient implementation of this scheme would not instantiate variables by χ in clauses or literals that are afterwards deleted, but would directly apply the simplification.

Note also that simple optimizations can further be applied to reduce the size of the instantiation set. For example, given a set of clauses S, there is no need to keep in the instantiation set B_S two distinct terms t and s such that $S \models_{\mathbb{Z}} t \simeq s$. Thus, it is useless to store in B_S a constant a and a term $\mathrm{p}(\mathrm{s}(a))$; if S contains a unit clause $t \simeq a$, there is no need for B_S to contain both t and a. Another rather obvious improvement is to use several distinct sorts interpreted as integers. Then the arithmetic variables need only to be instantiated by terms of the same sort. Our results extend straightforwardly to such settings, but we chose not to directly include these optimizations in our proofs for the sake of readability.

7 Applications

We now show two applications of our technique to solve satisfiability problems involving integers.

Arrays with Integer Indices. The theory of *arrays with integer indices* is axiomatized by the following set of clauses, denoted by $\mathcal{A}_{\mathbb{Z}}$:

$$\begin{array}{rll} \| \rightarrow \mathtt{select}(\mathtt{store}(x, z, v), z) \simeq v & & (a_1) \\ w \preceq \mathrm{p}(z) \,\| \rightarrow \mathtt{select}(\mathtt{store}(x, z, v), w) \simeq \mathtt{select}(x, w) & & (a_2) \\ \mathrm{s}(z) \preceq w \,\| \rightarrow \mathtt{select}(\mathtt{store}(x, z, v), w) \simeq \mathtt{select}(x, w) & & (a_3) \end{array}$$

Instead of clauses (a_2) and (a_3), the standard axiomatization of the theory of arrays contains $w \not\simeq z \,\| \rightarrow \mathtt{select}(\mathtt{store}(x, z, v), w) \simeq \mathtt{select}(x, w)$. In order to be able to apply our scheme, we replaced atom $w \not\simeq z$ by the disjunction $w \preceq \mathrm{p}(z) \vee \mathrm{s}(z) \preceq w$, which is equivalent in $\mathbb{Z}$.

We consider SMT problems on integer-indexed arrays of a particular kind, encoded by sets of clauses in which the only non-ground arithmetic atoms are of the form $x \preceq t$ or $t \preceq x$, and in which every term occurring in S with `store` as a head symbol is ground. This leads to the following definition on $\mathbb{Z}$-clauses:

Definition 15. *An* $\mathcal{A}_{\mathbb{Z}}$-inequality problem *is a set of* $\mathbb{Z}$*-clauses* $\mathcal{A}_{\mathbb{Z}} \cup S_0$ *where:*

– *the only variables occurring in* S_0 *are integer variables,*
– *all non-ground arithmetic atoms occurring in* S_0 *that are not abstraction literals are of the form* $x \preceq t$ *or* $t \preceq x$*, where* t *is either a variable or a ground term,*
– *every variable occurring in a term in* $C \in S_0$ *whose head symbol is* `store` *must occur in a grounding abstraction literal in* C*.*

For every $\mathcal{A}_{\mathbb{Z}}$-inequality problem S, we define the following set of ground terms, which will be used throughout this section:

$$\begin{aligned} B_S = &\{t \text{ ground} \,|\, x \preceq t \text{ or } \mathtt{select}(a, t) \text{ occurs in } S\} \\ &\cup \{t' \text{ ground} \,|\, \mathtt{store}(a, u, e) \simeq b \text{ and } u \simeq t' \text{ occur in a same clause in } S\} \\ &\cup \{\mathrm{p}(t') \text{ ground} \,|\, \mathtt{store}(a, u, e) \simeq b, u \simeq t' \text{ occur in a same clause in } S\} \end{aligned}$$

The following lemma is a consequence of Theorem 3.

Lemma 1. *Let* $B'_S = B_S \cup \{\chi\}$*, let* V *denote the set of inequality variables occurring in clauses in* S*, and let* Ω *denote the set of all substitutions of domain* V *and codomain* B'_S*. Then* $\mathcal{A}_{\mathbb{Z}} \cup S_0$ *and* $(\mathcal{A}_{\mathbb{Z}} \cup S_0)\Omega$ *are equisatisfiable.*

Since we assumed all integer variables in S are either abstraction variables or inequality variables (by otherwise adding $x \preceq x$ to the necessary clauses), we conclude that the clauses in $S_0\Omega$ are all ground, and the clauses in $\mathcal{A}_{\mathbb{Z}}\Omega$ are of the form:

$$\begin{array}{rl} \| & \mathtt{select}(\mathtt{store}(x, z, v), z) \simeq v \\ s \preceq \mathrm{p}(z) \,\| & \mathtt{select}(\mathtt{store}(x, z, v), s) \simeq \mathtt{select}(x, s) \\ \mathrm{s}(z) \preceq s \,\| & \mathtt{select}(\mathtt{store}(x, z, v), s) \simeq \mathtt{select}(x, s), \end{array}$$

where s is a ground term. This set of terms can be instantiated using the scheme of [11]. Thus, if Ω' denotes the set of substitutions constructed by the instantiation scheme, then by Theorem 4, the sets S and $S\Omega\Omega'$ are equisatisfiable. The latter is ground, and its satisfiability can be tested by any SMT solver capable of handling linear arithmetic and congruence closure.

An Example. Consider the following sets:

$$E = \{l_i \preceq x_i \preceq u_i \parallel\ \rightarrow \mathtt{select}(a, x_i) \simeq e_i \mid i = 1, \dots, n\},$$
$$F = \{u_i \preceq \mathrm{p}(l_i) \parallel\ \rightarrow \mid i = 1, \dots, n\},$$
$$G = \{u_i \preceq \mathrm{p}(l_{i+1}) \parallel\ \rightarrow \mid i = 1, \dots, n-1\},$$

where the u_i's and l_j's are constants. The $\mathbb{Z}$-clauses in E state that array a is constant between bounds l_i and u_i, for $i = 1, \dots, n$; the $\mathbb{Z}$-clauses in F state that each interval has at least 1 element, and the $\mathbb{Z}$-clauses in G state that all the intervals have a nonempty intersection. Thus, the union of these sets entails that a is constant between bounds l_1 and u_n. Let b denote the array obtained from a by writing element e_1 at position u_{n+1}. If $u_{n+1} = \mathrm{s}(u_n)$, then b is constant between bounds l_1 and $\mathrm{s}(u_n)$. Let

$$\begin{aligned} H = &\{x \simeq u_{n+1} \parallel\ \rightarrow b \simeq \mathtt{store}(a, x, e_1),\ \parallel\ \rightarrow u' \simeq \mathrm{s}(u_n)\} \\ &\cup \{k \preceq \mathrm{p}(l_1) \parallel\ \rightarrow, u_n \preceq \mathrm{p}(k) \parallel\ \rightarrow\} \\ &\cup \{\parallel \mathtt{select}(b, k) \simeq e_1 \rightarrow\}, \text{ and} \\ S_0 = &\ E \cup F \cup G \cup H, \end{aligned}$$

then $\mathcal{A}_\mathbb{Z} \cup S_0$ is unsatisfiable. By applying the definition of B_S from the previous section, we obtain $B_S = \{u_1, \dots, u_n, u_{n+1}, \mathrm{p}(u_{n+1}), k\}$. In the first step, all variables in E are instantiated with the elements of $B'_S = B_S \cup \{\chi\}$, yielding[3] a ground set E'. The inequality variables in the axioms of $\mathcal{A}_\mathbb{Z}$ are also instantiated with the elements of B'_S, yielding a set of clauses A. Then, in the second step, the clauses in A are instantiated using the term $\mathtt{store}(a, u_{n+1}, e_1)$, and we obtain a set A' containing clauses of the form

$$\begin{aligned} x \simeq u_{n+1} &\parallel \mathtt{select}(\mathtt{store}(a, x, e_1), x) \simeq e_1, \\ x \simeq u_{n+1}, s \preceq \mathrm{p}(x) &\parallel \mathtt{select}(\mathtt{store}(a, x, e_1), s) \simeq \mathtt{select}(a, s), \\ x \simeq u_{n+1}, \mathrm{s}(x) \preceq s &\parallel \mathtt{select}(\mathtt{store}(a, x, e_1), s) \simeq \mathtt{select}(a, s), \end{aligned}$$

where $s \in B'_S$. Then an SMT solver is invoked on the ground set of clauses $A' \cup E' \cup F \cup G \cup H$. The size of this set is to be compared with the one obtained by the procedure of [10], clauses are instantiated using an index set $\mathcal{I} = \{l_i, u_i \mid i = 1, \dots, n\} \cup \{u_{n+1}, \mathrm{p}(u_{n+1}), \mathrm{s}(u_{n+1}), \mathrm{s}(u_n), k\}$. There are twice as many terms in this instantiation set. It is simple to check that our procedure always generates less instances than the one of [10]. In fact, there are cases in which our method is exponentially better. For example, for $i = 1, \dots, n$, let A_i denote the atom $\mathtt{select}(a, x_i) \simeq c_i$, and let $S = \mathcal{A}_\mathbb{Z} \cup S_0$, where $S_0 = \{i \preceq x_1, \dots, i \preceq x_n, j \preceq y \parallel A_1, \dots, A_n \rightarrow \mathtt{select}(a, y) \simeq e\}$. With this set, our instantiation scheme generates only a *unique* clause, whereas the one in [10] instantiates every x_i with i and j, yielding 2^n clauses.

Stratified Classes. To show the wide range of applicability of our results, we provide another example of a domain where they can be applied. The results in this section concern decidable subclasses of first-order logic with sorts, which are investigated in [1]. We briefly review some definitions.

[3] In an actual implementation, the variables in E would not be instantiated with χ.

Definition 16. *A set of function symbols Σ is* stratified *if there exists a function* level *mapping every sort* $\mathtt{s}$ *to a natural number such that for every function symbol $f \in \Sigma$ of profile $\mathtt{s}_1 \times \dots \mathtt{s}_n \to \mathtt{s}$ and for every $i \in [1..n]$,* $\text{level}(\mathtt{s}) > \text{level}(\mathtt{s}_i)$*. We denote by T_Σ (resp. $T_\Sigma^{\mathtt{s}}$) the set of ground terms built on the set of function symbols Σ (resp. the set of terms of sort $\mathtt{s}$ built on Σ).*

Proposition 1. *Let Σ be a finite stratified set of function symbols. Then the set T_Σ is finite.*

A set of clauses is in St_0 if its signature is stratified. In particular, any formula in the Bernays-Schönfinkel class is in St_0. By Proposition 1, St_0 admits a trivial instantiation scheme: it suffices to replace each variable by every ground term of the same sort, defined on the set of function symbols occurring in St_0[4]. This instantiation scheme is obviously admissible (see Definition 13).

This instantiation scheme can be applied also to the class St_2 defined in [1] as an extension of the class St_0 with atoms of the form $t \in \text{Im}[f]$, where f is a function symbol of profile $\mathtt{s}_1 \times \dots \times \mathtt{s}_n \to \mathtt{s}$, meaning that t is in the image of the function f. From a semantic point of view, the atom $t \in \text{Im}[f]$ is a shorthand for $\exists x_1, \dots, x_n . t \simeq f(x_1, \dots, x_n)$. To ensure decidability, for every atom of the form $t \in \text{Im}[f]$ and for every function symbol g of the same range as f, the following properties have to be satisfied: (i) g must have the same profile as f; (ii) the formula $f(x_1, \dots, x_n) \simeq g(y_1, \dots, y_n) \Rightarrow \bigwedge_{i=1}^n x_i \simeq y_i$, where n denotes the arity of f and g, must hold in every model of the considered formula. In [1] it is shown that every satisfiable set in St_2 admits a finite model, hence, St_2 is decidable. It turns out that any formula in St_2 can be reduced to a clause set in St_0, thus reducing satisfiability problems in St_2 to satisfiability problems in St_0. This is done by getting rid of all occurrences of atoms of the form $t \in \text{Im}[f]$. One such transformation is obvious: by definition, every occurrence of the form $t \notin \text{Im}[f]$ can be replaced by $t \not\simeq f(x_1, \dots, x_n)$, where the x_i are fresh variables. We now focus on the other occurrences of the atoms.

Definition 17. *Let S denote a set of clauses. We denote by S' the set of clauses obtained from S by applying the following transformation rule (using a nondeterministic strategy):*

$$\Gamma \to \Delta, x \in Im[f] \rightsquigarrow \{x \simeq g(x_1, \dots, x_n), \Gamma \to \Delta, g(x_1, \dots, x_n) \in Im[f] \mid g \in \Sigma_f\}$$

where x is a variable, f is of arity n, Σ_f denotes the set of function symbols with the same profile as f and $x_1, \dots, x_n$ are fresh variables that are pairwise distinct. We denote by $S{\downarrow_0}$ the set of clauses obtained from S' by applying the following transformation rule: $g(x_1, \dots, x_n) \in Im[f] \rightsquigarrow g(x_1, \dots, x_n) \simeq f(x_1, \dots, x_n)$.

The first rule gets rids of atoms of the form $x \in \text{Im}[f]$ by replacing them with atoms of the form $t \in \text{Im}[f]$ where t is a complex term, and the second rule gets rid of these atoms. The rules terminate and preserve satisfiability (see the technical report at `http://membres-lig.imag.fr/peltier/inst_la.pdf` for details). We therefore have the following result:

[4] Possibly enriched with some constant symbols in order to ensure that each sort is nonempty.

Theorem 5. *Consider a set of $\mathbb{Z}$-clauses $S = \{\Lambda_i \,\|\, C_i \mid i \in [1..n]\}$ in St_2 such that every $\Lambda_i \,\|\, C_i$ is preconstrained, and for every occurrence of an atom $t \in Im[f]$, the range of f is not of sort $\mathbb{Z}$. The set S is $\mathbb{Z}$-satisfiable if and only if $\gamma(S{\downarrow_0}\, \Omega)$ is $\mathbb{Z}$-satisfiable, where:*

– *Ω is the set of substitutions of domain $V(S)$ whose codomain is a set B such that $\Lambda_i \,\|\, C_i \trianglelefteq B$ for all $i = 1, \ldots, n$;*
– *γ denotes an instantiation scheme for St_0 satisfying the conditions of page 57 (e.g. $\gamma(S) = S\Omega'$ where Ω' is the set of substitutions of domain $V(S)$ and of codomain T_Σ).*

Examples of specifications in the classes St_0 and St_2 are presented in [1]. Our results allow the integration of integer constraints into these specifications.

8 Discussion

In this paper we presented a way of defining an instantiation scheme for SMT problems based on a combination of linear arithmetic with another theory. The scheme consists in getting rid of the integer variables in the problem, and applying another instantiation procedure to the resulting problem. Provided the integer variables essentially occur in inequality constraints, this scheme is complete for the combination of linear arithmetic with several theories of interest to the SMT community, but also for the combination of linear arithmetic with other decidable theories such as the class St_2 from [1]. The application of this scheme to the theory of arrays with integer indices shows that it can produce considerably fewer ground instances than other state-of-the-art procedures, such as that of [10]. The instantiation scheme of [11] is currently being implemented, and will be followed by a comparison with other tools on concrete examples from SMT-LIB[5].

As far as further research is concerned, we intend to investigate how to generalize this procedure, in which it is imposed that functions of range $\mathbb{Z}$ can only have integer arguments. We intend to determine how to allow other functions of range $\mathbb{Z}$ while preserving completeness. It is shown in [10] that considering arrays with integer elements, for which nested reads can be allowed, gives rise to undecidable problems, but we expect to define decidable subclasses, that may generalize those in [13]. Dealing with more general functions of range $\mathbb{Z}$ should also allow us to devise a new decision procedure for the class of arrays with dimension that is considered in [14]. We also intend to generalize our approach to other combinations of theories that do not necessarily involve linear arithmetic, by determining conditions that guarantee *combinations* of instantiation schemes can safely be employed to eliminate all variables from a formula. Another interesting line of research would be to avoid a systematic grounding of integer variables and to use decision procedures for non-ground systems of arithmetic formulae. The main difficulty is of course that with our current approach, instantiating integer variables is required to determine how to instantiate the remaining variables.

[5] http://www.smt-lib.org/

References

1. Abadi, A., Rabinovich, A., Sagiv, M.: Decidable fragments of many-sorted logic. Journal of Symbolic Computation 45(2), 153–172 (2010)
2. Althaus, E., Kruglov, E., Weidenbach, C.: Superposition modulo linear arithmetic sup(la). In: Ghilardi, S., Sebastiani, R. (eds.) FroCoS 2009. LNCS, vol. 5749, pp. 84–99. Springer, Heidelberg (2009)
3. Armando, A., Bonacina, M.P., Ranise, S., Schulz, S.: New results on rewrite-based satisfiability procedures. ACM Transactions on Computational Logic 10(1), 129–179 (2009)
4. Armando, A., Ranise, S., Rusinowitch, M.: A rewriting approach to satisfiability procedures. Information and Computation 183(2), 140–164 (2003)
5. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. Journal of Logic and Computation 3(4), 217–247 (1994)
6. Bachmair, L., Ganzinger, H., Waldmann, U.: Refutational theorem proving for hierachic first-order theories. Appl. Algebra. Eng. Commun. Comput. 5, 193–212 (1994)
7. Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability modulo theories. In: Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, ch. 26, vol. 185, pp. 825–885. IOS Press, Amsterdam (2009)
8. Bonacina, M.P., Echenim, M.: On variable-inactivity and polynomial T-satisfiability procedures. J. of Logic and Computation 18(1), 77–96 (2008)
9. Bonacina, M.P., Echenim, M.: Theory decision by decomposition. Journal of Symbolic Computation 45(2), 229–260 (2010)
10. Bradley, A.R., Manna, Z.: The Calculus of Computation: Decision Procedures with Applications to Verification. Springer, New York (2007)
11. Echenim, M., Peltier, N.: A new instantiation scheme for Satisfiability Modulo Theories. Technical report, LIG, CAPP group (2009), http://membres-lig.imag.fr/peltier/rr-smt.pdf
12. Ganzinger, H., Korovin, K.: Integrating equational reasoning into instantiation-based theorem proving. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 71–84. Springer, Heidelberg (2004)
13. Ge, Y., de Moura, L.M.: Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification. LNCS, vol. 5643, pp. 306–320. Springer, Heidelberg (2009)
14. Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Decision procedures for extensions of the theory of arrays. Ann. Math. Artif. Intell. 50(3-4), 231–254 (2007)
15. Korovin, K., Voronkov, A.: Integrating linear arithmetic into superposition calculus. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 223–237. Springer, Heidelberg (2007)
16. Lee, S., Plaisted, D.A.: Eliminating duplication with the hyper-linking strategy. Journal of Automated Reasoning 9, 25–42 (1992)
17. Plaisted, D.A., Zhu, Y.: Ordered semantic hyperlinking. Journal of Automated Reasoning 25(3), 167–217 (2000)

On Krawtchouk Transforms

Philip Feinsilver[1] and René Schott[2]

[1] Southern Illinois University, Carbondale, IL. 62901, U.S.A.
pfeinsil@math.siu.edu
[2] IECN and LORIA, Nancy-Université, Université Henri Poincaré
54506 Vandoeuvre-lès-Nancy, France
schott@loria.fr

Abstract. Krawtchouk polynomials appear in a variety of contexts, most notably as orthogonal polynomials and in coding theory via the Krawtchouk transform. We present an operator calculus formulation of the Krawtchouk transform that is suitable for computer implementation. A positivity result for the Krawtchouk transform is shown. Then our approach is compared with the use of the Krawtchouk transform in coding theory where it appears in MacWilliams' and Delsarte's theorems on weight enumerators. We conclude with a construction of Krawtchouk polynomials in an arbitrary finite number of variables, orthogonal with respect to the multinomial distribution.

1 Introduction

Krawtchouk polynomials appear originally as orthogonal polynomials for the binomial distribution [4,10], and in coding theory via the Krawtchouk transform in the context of MacWilliams' theorem on weight enumerators as well in Delsarte's extension to association schemes [5,8]. They play a role in discrete formulations of quantum mechanics [2,6,7,9], transforms in optics [1], as well as in recent developments in image analysis [11].

We present an operator calculus formulation of the Krawtchouk transform that not only is theoretically elucidating, it is highly suitable for computer implementation. A consequence of our formulation is a positivity theorem for the Krawtchouk transform of polynomials. We indicate connections with the transform appearing in coding theory.

2 Krawtchouk Polynomials as a Canonical Appell System

2.1 Generating Function

Consider a Bernoulli random walk starting at the origin, jumping to the left with probability q, to the right with probability p, $p+q=1$, $pq \neq 0$. After N steps, the position is x, with $j=(N-x)/2$ denoting the number of jumps to the left.

Start with the generating function

$$\begin{aligned}G(v) &= (1+2qv)^{(N+x)/2}(1-2pv)^{(N-x)/2}\\ &= (1+2qv)^{N-j}(1-2pv)^j = \sum_{n=0}^{N} \frac{v^n}{n!} K_n\end{aligned}$$

where we consider K_n as a function of x or of j according to context.

2.2 Orthogonality

We check orthogonality with respect to the binomial distribution, j running from 0 to N, with corresponding probabilities $\binom{N}{j} q^j p^{N-j}$. Using angle brackets to denote expected value, we wish to show that $\langle G(v)G(w)\rangle$ is a function of the product vw.

$$\begin{aligned}\langle G(v)G(w)\rangle &= \sum \binom{N}{j} q^j p^{N-j}(1+2qv)^{N-j}(1-2pv)^j(1+2qw)^{N-j}(1-2pw)^j\\ &= \sum \binom{N}{j} q^j p^{N-j}(1+2q(v+w)+4q^2vw)^{N-j}(1-2p(v+w)+4p^2vw)^j\\ &= (p+2pq(v+w)+4pq^2vw+q-2pq(v+w)+4p^2qvw)^N\\ &= (1+4pqvw)^N\end{aligned}$$

which immediately gives the squared norms $\|K_n\|^2 = (n!)^2\binom{N}{n}(4pq)^n$.

2.3 Canonical Appell System

For an Appell system with generating function $\exp[zx - tH(z)]$, a corresponding canonical Appell system has a generating function of the form

$$\exp[xU(v) - tH(U(v))]$$

where $U(v)$ is analytic about the origin in $\mathbb{C}$, with analytic inverse $V(z)$, and $H(z)$ is the logarithm of the Fourier-Laplace transform of the distribution of x at time 1. Here we have N replacing t, and write

$$G(v) = (1+2(q-p)v-4pqv^2)^{N/2}\left(\frac{1+2qv}{1-2pv}\right)^{x/2}$$

identifying

$$U(v) = \frac{1}{2}\log\frac{1+2qv}{1-2pv}, \quad \text{and} \quad H(z) = \log(pe^z + qe^{-z})$$

One checks that

$$\log(pe^{U(v)} + qe^{-U(v)}) = p\sqrt{\frac{1+2qv}{1-2pv}} + q\sqrt{\frac{1-2pv}{1+2qv}} = (1+2(q-p)v-4pqv^2)^{-1/2}$$

which verifies the form $\exp[xU(v) - NH(U(v))]$.

Solving $z = U(v) = U(V(z))$, we find

$$V(z) = \frac{(e^2 - e^{-z})/2)}{pe^z + qe^{-z}} = \frac{\sinh z}{pe^z + qe^{-z}}$$

3 Krawtchouk Expansions

The generating function, with $z = U(v)$, is

$$G(v) = e^{zx - NH(z)} = \sum_{n \geq 0} \frac{V(z)^n}{n!} K_n(x, N)$$

Rearrange to get,

$$e^{zx} = (pe^z + qe^{-z})^N \sum_{n \geq 0} \left(\frac{\sinh z}{pe^z + qe^{-z}} \right)^n \frac{K_n(x, N)}{n!}$$

We want to write the coefficients of the expansion in terms of $D = d/dx$ acting on a function of x. We cannot substitute $z \leftrightarrow D$ directly, since the D and x do not commute. Introduce another variable s. Replacing z by $D_s = d/ds$, apply both sides to a function $f(s)$:

$$e^{xD_s} f(s) = f(s + x) = (pe^{D_s} + qe^{-D_s})^N \sum_{n \geq 0} \left(\frac{\sinh D_s}{pe^{D_s} + qe^{-D_s}} \right)^n \frac{K_n(x, N)}{n!} f(s)$$

Now we move the operators involving D_s past $K_n(x, N)$. Letting $s = 0$, thinking of f as a function of x instead of s, we can replace D_s by our usual $D = d/dx$, to get

$$f(x) = \sum_{0 \leq n \leq N} \frac{K_n(x, N)}{n!} (pe^D + qe^{-D})^N \left(\frac{\sinh D}{pe^D + qe^{-D}} \right)^n f(0)$$

In other words, the coefficients of the Krawtchouk expansion of $f(x)$ are given by

$$\tilde{f}(n) = \frac{1}{n!} (pe^D + qe^{-D})^{N-n} (\sinh D)^n f(0) \tag{1}$$

Theorem 1. *For $p > q$, if a polynomial has positive coefficients, then the coefficients of its Krawtchouk expansion are positive. For $p = q = 1/2$, we have nonnegativity of the Krawtchouk coefficients.*

Proof. Note that if $\psi(D)$ is any formal power series $\sum c_n D^n$ in D, then

$$\psi(D) x^m \Big|_0 = m! \, c_m$$

is positive if c_m is. Now, write $pe^z + qe^{-z} = \cosh z + (p - q) \sinh z$. If $p > q$, then the power series expansion has positive coefficients so that the coefficients in the Krawtchouk expansion are positive as well. For $p = q$, we have nonnegativity.

3.1 Matrix Formulation

As shown in [3], for Krawtchouk polynomials of degree at most N, one can use the matrix of D, denoted $\hat{D}$, acting on the standard basis $\{1, x, x^2, \ldots, x^N\}$ replacing the operator D in eq. (1).

For example, take $N = 4$. We have

$$\hat{D} = \begin{pmatrix} 0&1&0&0&0 \\ 0&0&2&0&0 \\ 0&0&0&3&0 \\ 0&0&0&0&4 \\ 0&0&0&0&0 \end{pmatrix}$$

The basic matrices needed are computed directly from the power series for the exponential function. Thus,

$$pe^{\hat{D}} + qe^{-\hat{D}} = \begin{pmatrix} 1 & 2p-1 & 1 & 2p-1 & 1 \\ 0 & 1 & 4p-2 & 3 & 8p-4 \\ 0 & 0 & 1 & 6p-3 & 6 \\ 0 & 0 & 0 & 1 & 8p-4 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

where the relation $p + q = 1$ has been used and

$$\sinh \hat{D} = \begin{pmatrix} 0&1&0&1&0 \\ 0&0&2&0&4 \\ 0&0&0&3&0 \\ 0&0&0&0&4 \\ 0&0&0&0&0 \end{pmatrix}$$

The nilpotence of $\hat{D}$ reduces the exponentials to polynomials in $\hat{D}$, making computations with these matrices convenient and fast.

4 Krawtchouk Polynomials in Coding Theory

The Krawtchouk transform appears in coding theory in a different variant as an essential component of MacWilliams' theorem on weight enumerators [5,8]. It appears in Delsarte's formulation in terms of association schemes as well.

Fix $N > 0$. For Krawtchouk transforms on functions defined on $\{-N, 2-N, \ldots, N-2, N\}$ we use the Fourier-Krawtchouk matrices, Φ, which we call "Kravchuk matrices". The entries are the values of the Krawtchouk polynomials as functions on $\{-N, \ldots, N\}$. Thus, via the mapping $x = N - 2j$, $0 \le j \le N$, the columns correspond to values of x and we write

$$G(v) = (1 + 2qv)^{N-j}(1 - 2pv)^j = \sum_i v^i \Phi_{ij}$$

In [5], the Krawtchouk polynomials are defined via a slightly different generating function, changing the notation to fit our context,

$$G_s(v) = (1 + (s-1)v)^{N-j}(1 - v)^j = \sum_i v^i K_i(j; N, s)$$

The difference is primarily one of scaling. Comparing G with G_s, replacing v by $v/(2p)$, we find

$$K_i(j; N, s) = (2p)^{-i} \Phi_{ij}$$

with $s - 1 = \frac{q}{p} = \frac{1-p}{p} = \frac{1}{p} - 1$, or

$$s = \frac{1}{p}$$

The condition $s \ge 2$ thus corresponds to $p \le q$, complementary to the condition required for positivity in the previous section.

Following [5, p. 132], we have:

- MacWilliams' Theorem: If A is a linear code over $\mathbb{F}_s$ and $B = A^\perp$, its dual, then the weight distribution of B is, up to a factor, the Krawtchouk transform of the weight distribution of A.
- Delsarte's Theorem: If A is a code over an alphabet of size s, then the values of the Krawtchouk transform of the coefficients of the distance enumerator are nonnegative.

In this context, the components of the Krawtchouk transform of a vector $\mathbf{v}$ are defined by

$$\hat{v}_i = \sum_j K_i(j; N, s) v_j$$

We show how to invert the transform.

4.1 Inverse Transform

The calculation of §2.2 can be recast in matrix form. Set B equal to the diagonal matrix

$$B = \operatorname{diag}\left(p^N, Np^{N-1}q, \ldots, \binom{N}{i} p^{N-i}q^i, \ldots, q^N\right)$$

Let Γ denote the diagonal matrix of squared norms, here without the factors of $n!$,

$$\Gamma = \operatorname{diag}\left(1, N(4pq), \ldots, \binom{N}{i} (4pq)^i, \ldots, (4pq)^N\right)$$

With $G(v)=\sum v^i\Phi_{ij}$, we have, following the calculation of §2.2,

$$\begin{aligned}\sum_{i,j} v^i w^j (\Phi B \Phi^T)_{ij} &= \sum_{i,j,k} v^i w^j \Phi_{ik} B_{kk} \Phi_{jk} \\ &= \langle G(v)G(w)\rangle = \sum_i (vw)^i \Gamma_{ii}\end{aligned}$$

In other words, the orthogonality relation takes the form

$$\Phi B \Phi^T = \Gamma$$

from which the inverse is immediate

$$\Phi^{-1} = B\Phi^T \Gamma^{-1}$$

A consequence of this formula is that $\det \Phi$ is independent of p and q:

$$(\det \Phi)^2 = \frac{\det \Gamma}{\det B} = 2^{N(N+1)}$$

The sign can be checked for the symmetric case $p=q=1/2$ with the result

$$\det \Phi = \pm\, 2^{N(N+1)/2}$$

with the + sign for $N \equiv 0, 3 \pmod 4$.

We illustrate with an example.

Example. For $N=4$,

$$\Phi = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 8\,q & 6\,q-2\,p & 4\,q-4\,p & 2\,q-6\,p & -8\,p \\ 24\,q^2 & 12\,q^2-12\,pq & 4\,q^2-16\,pq+4\,p^2 & -12\,pq+12\,p^2 & 24\,p^2 \\ 32\,q^3 & 8\,q^3-24\,pq^2 & -16\,pq^2+16\,p^2q & 24\,p^2q-8\,p^3 & -32\,p^3 \\ 16\,q^4 & -16\,pq^3 & 16\,p^2q^2 & -16\,p^3q & 16\,p^4 \end{pmatrix}$$

where we keep p and q to show the symmetry.

$$Q = 2^4\, B\Phi^T\Gamma^{-1} = \begin{pmatrix} 16\,p^4 & 8\,p^3 & 4\,p^2 & 2\,p & 1 \\ 64\,p^3q & 8\,p^2\,(3\,q-p) & 8\,p\,(q-p) & 2\,q-6\,p & -4 \\ 96\,p^2q^2 & 24\,qp\,(q-p) & 4\,q^2-16\,pq+4\,p^2 & -6\,q+6\,p & 6 \\ 64\,pq^3 & 8\,q^2\,(q-3\,p) & -8\,q\,(q-p) & 6\,q-2\,p & -4 \\ 16\,q^4 & -8\,q^3 & 4\,q^2 & -2\,q & 1 \end{pmatrix}$$

satisfies $Q\Phi = \Phi Q = 2^4\, I$.

4.2 Modified Involution Property

For $p = 1/2$, there is a simpler approach to inversion. Namely the identity

$$\Phi^2 = 2^N\, I$$

We will see how this is modified for $p \neq 1/2$, which will result in a very simple form for Φ^{-1}.

Proposition 1. *Let P be the diagonal matrix*

$$P = \operatorname{diag}\left((2p)^N, \ldots, (2p)^{N-j}, \ldots, 1\right)$$

Let P' be the diagonal matrix

$$P' = \operatorname{diag}\left(1, \ldots, (2p)^j, \ldots, (2p)^N\right)$$

Then

$$\Phi P \Phi = 2^N P'$$

We just sketch the proof as it is similar to that for orthogonality.

Proof. The matrix equation is the same as the corresponding identity via generating functions. Namely,

$$\sum_{i,j,k} v^i \Phi_{ik} (2p)^{N-k} \Phi_{kj} w^j \binom{N}{j} = 2^N (1 + 2pvw)^N$$

First, sum over i, using the generating function $G(v)$, with j replaced by k. Then sum over k, again using the generating function. Finally, summing over j using the binomial theorem yields the desired result, via $p + q = 1$.

Thus,

Corollary 1

$$\Phi^{-1} = 2^{-N}\, P \Phi P'^{-1}$$

5 Krawtchouk Polynomials in 2 or More Variables

Now we will show a general construction of Krawtchouk polynomials in variables $(j_1, j_2, \ldots, j_d)$, running from 0 to N according to the level N. These systems are analogous to wavelets in that they have a dimension, d, and a resolution N.

5.1 Symmetric Representation of a Matrix

Given a $d \times d$ matrix A, we will find the "symmetric representation" of A, the action on the symmetric tensor algebra of the underlying vector space. This is effectively the action of the matrix A on vectors extended to polynomials in d variables.

Introduce commuting variables $x_1, \ldots, x_d$. Map

$$y_i = \sum_j A_{ij} x_j$$

We use multi-indices, $m = m_1, \ldots, m_d$, $m_i \geq 0$, similarly for n. Then a monomial

$$x^m = x_1^{m_1} x_2^{m_2} \cdots x_d^{m_d}$$

and similarly for y^n. The induced map at level N has matrix elements $\bar{A}_{nm}$ determined by the expansion

$$y^n = \sum_m \bar{A}_{nm} x^m$$

One can order the matrix entries lexicographically corresponding to the monomials x^m. We call this the *induced matrix* at level N. Often the level is called the *degree* since the induced matrix maps monomials of homogeneous degree N to polynomials of homogeneous degree N.

We introduce the special matrix B which is a diagonal matrix with multinomial coefficients as entries.

$$B_{nm} = \delta_{nm} \binom{N}{n} = \frac{N!}{n_1!\, n_2! \cdots n_d!}$$

For simplicity, for B we will not explicitly denote the level or dimension, as it must be consistent with the context.

The main feature of the map $A \to \bar{A}$ is that at each level it is a multiplicative homomorphism, that is,

$$\overline{AB} = \bar{A}\,\bar{B}$$

as follows by applying the definition to $y_i = \sum (AB)_{ij} x_j$ first to A, then to B. Observe, then, that the identity map is preserved and that inverses map to inverses. However, transposes require a separate treatment.

Transposed Matrix. The basic lemma is the relation between the induced matrix of A with that of its transpose. We denote the transpose of A, e. g., by A^T.

Lemma 1. *The induced matrices at each level satisfy*

$$\overline{A^T} = B^{-1} \bar{A}^T B$$

Proof. Start with the bilinear form $F = \sum_{i,j} x_i A_{ij} y_j$. Then, from the definition of $\bar{A}$,

$$F^N = \sum_{n,m} x^n \bar{A}_{nm} y^m \binom{N}{n}$$

Now write $F = \sum_{i,j} x_i (A^T)_{ji}\, y_j$ with

$$F^N = \sum_{n,m} \binom{N}{m} y^m \overline{A^T}_{mn}\, x^n$$

Matching the above expressions, we have, switching indices appropriately,

$$(\bar{A}^T B)_{mn} = \bar{A}_{nm} \binom{N}{n} = \binom{N}{m} \overline{A^T}_{mn} = (B\, \overline{A^T})_{mn}$$

which is the required relation.

5.2 General Construction of Orthogonal Polynomials with Respect to a Multinomial Distribution

For the remainder of the article, we work in $d+1$ dimensions, with the first coordinate subscripted with 0.

The multinomial distribution extends the binomial distribution to a sequence of independent random variables where one of d choices occurs at each step, choice i occurring with probability p_i. The probability of none of the d choices is $p_0 = 1 - p_1 - p_2 - \cdots - p_d$. The probabilities for a multinomial distribution at step $N$are given by

$$p(j_1, j_2, \ldots, j_d) = \binom{N}{j_0, j_1, j_2, \ldots, j_d} p_0^{j_0} p_1^{j_1} \cdots p_d^{j_d}$$

this being the joint probability distribution that after N trials, choice i has occurred j_i times, with none of them occurring $j_0 = N - j_1 - \cdots - j_d$ times. Let P denote the diagonal matrix with diagonal $(p_0, p_1, \ldots, p_d)$. Then with

$$y_i = \sum_j P_{ij} x_j = p_i x_i$$

we see that $y^n = p^n x^n$ which implies that, at each level N, $\bar{P}$ is diagonal with entries

$$\bar{P}_{nm} = \delta_{nm} p^n = p_0^{N-n_1-\cdots-n_d} p_1^{n_1} \cdots p_d^{n_d}$$

In other words, the multinomial distribution comprises the entries of the diagonal matrix $B\bar{P}$.

Now for the construction. Start with an orthogonal matrix W, a diagonal matrix of probabilities P, and a diagonal matrix D with positive entries on the diagonal. Let

$$A = P^{-1/2}\, W\, D^{1/2} \tag{2}$$

where the power on a diagonal matrix is applied entrywise. Then it is readily checked that

$$A^T P A = D$$

which will generate the squared norms of the polynomials, to be seen shortly. Using the homomorphism property, we have, via the Lemma,

$$\overline{A^T}\bar{P}\bar{A} = B^{-1}\bar{A}^T B \bar{P} \bar{A} = \bar{D}$$

Or, setting $\Phi = \bar{A}^T$,

$$\Phi\, B\bar{P}\, \Phi^T = B\bar{D} \tag{3}$$

with both $B\bar{P}$ and $B\bar{D}$ diagonal.

Taking as variables the column indices, writing j for m, we have the n^{th} Krawtchouk polynomial

$$K_n(j; N, p) = \Phi_{nj}$$

at level N. The relation given by equation (3) is the statement that

the Krawtchouk polynomials are orthogonal with respect to the multinomial distribution, $B\bar{P}$, with squared norms given by the entries of the induced matrix $B\bar{D}$.

To summarize, starting with the matrix A as in equation (2), form the induced matrix at level N. Then the corresponding Krawtchouk polynomials are functions of the column labels of the transpose of the induced matrix. Apart from the labelling conventions, then, in fact they are polynomials in the row labels of the original induced matrix.

Finally, note that the basic case arises from the choice

$$A = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

for $d = 1$ with $p = 1 - p = 1/2$, $D = I$.

Remark 1. A useful way to get a symmetric orthogonal matrix W is to start with any vector, v, form the rank-one projection, $V = vv^T/v^Tv$ and take for W the corresponding reflection $2V - I$.

5.3 Examples

Two Variables. For two variables, start with the 3×3 matrices

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 1 & -2 \end{pmatrix}, \qquad P = \begin{pmatrix} 1/3 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1/6 \end{pmatrix}$$

and D the identity. We find for the level two induced matrix

$$\Phi^{(2)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 0 & 2 & -2 & 0 & 2 \\ 2 & 1 & -1 & 0 & -2 & -4 \\ 1 & -1 & 1 & 1 & -1 & 1 \\ 2 & -1 & -1 & 0 & 2 & -4 \\ 1 & 0 & -2 & 0 & 0 & 4 \end{pmatrix}$$

indicating the level explicitly. These are the values of the polynomials evaluated at integer values of the variables $(j_1, j_2, \ldots)$. So multiplying a data vector on either side will give the corresponding Krawtchouk transform of that vector.

Three variables. This example is very close to the basic case for $d = 1$. Start with the vector $v^T = (1, -1, -1, -1)$. Form the corresponding rank-one projection and the associated reflection, as indicated in the remark above. We find

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

and take the uniform distribution $p_i = 1/4$, and $D = I$. We find

$$\Phi^{(2)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 0 & 0 & 2 & 0 & 0 & -2 & -2 & -2 \\ 2 & 0 & 2 & 0 & -2 & 0 & -2 & 2 & 0 & -2 \\ 2 & 0 & 0 & 2 & -2 & -2 & 0 & -2 & 0 & 2 \\ 1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 & 1 & 1 \\ 2 & 0 & 0 & -2 & -2 & 2 & 0 & -2 & 0 & 2 \\ 2 & 0 & -2 & 0 & -2 & 0 & 2 & 2 & 0 & -2 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 \\ 2 & -2 & 0 & 0 & 2 & 0 & 0 & -2 & 2 & -2 \\ 1 & -1 & -1 & 1 & 1 & 1 & -1 & 1 & -1 & 1 \end{pmatrix}$$

With $A^2 = 4I$, we have, in addition to the orthogonality relation, as in equation (3), that $(\Phi^{(2)})^2 = 16I$.

6 Conclusion

Using matrix methods allows for a clear formulation of the properties of Krawtchouk polynomials and Krawtchouk transforms. Working with polynomials or with vectors, computations can be done very efficiently. We have shown how to construct Krawtchouk polynomials in an arbitrary (finite) number of variables, with enough flexibility in the parameters to allow for a wide range of potential applications.

References

1. Atakishiyev, N.M., Wolf, K.B.: Fractional Fourier-Kravchuk Transform. J. Opt. Soc. Am. A. 14, 1467–1477 (1997)
2. Atakishiyev, N.M., Pogosyan, G.S., Wolf, K.B.: Finite Models of the Oscillator. Physics of Particles and Nuclei 36(Suppl. 3), 521–555 (2005)

3. Feinsilver, P., Schott, R.: Finite-Dimensional Calculus. Journal of Physics A: Math.Theor. 42, 375214 (2009)
4. Feinsilver, P., Schott, R.: Algebraic Structures and Operator Calculus. In: Representations and Probability Theory, vol. I-III. Kluwer Academic Publishers, Dordrecht (1993-1995)
5. Hall, J.I.: Notes on coding theory, `http://www.mth.msu.edu/~jhall/classes/codenotes/coding-notes.html`
6. Lorente, M.: Orthogonal polynomials, special functions and mathematical physics. Journal of Computational and Applied Mathematics 153, 543–545 (2003)
7. Lorente, M.: Quantum Mechanics on discrete space and time. In: Ferrero, M., van der Merwe, A. (eds.) New Developments on Fundamental Problems in Quantum Physics, pp. 213–224. Kluwer, Dordrecht (1997) arXiv:quant-ph/0401004v1
8. MacWilliams, F.J., Sloane, N.J.A.: Theory of error-correcting codes. North-Holland, Amsterdam (1977)
9. Santhanam, T.S.: Finite-Space Quantum Mechanics and Krawtchuk Functions. In: Proc. of the Workshop on Special Functions and Differential Equations, Madras, India, vol. 192. Allied Publishers, Delhi (1997)
10. Szëgo, Orthogonal Polynomials. AMS, Providence (1955)
11. Yap, P.-T., Paramesran, R.: Image analysis by Krawtchouk moments. IEEE Transactions on Image Processing 12, 1367–1377 (2003)

Appendix

Here is maple code for producing the symmetric powers of a matrix. The arguments are the matrix X and the level N, denoted dg in the code, for “degree”.

```
SYMPOWER := proc(X, dg)
local nd, ND, XX, x, y, strt, i, vv, yy, j, ww,kx,kk;
  nd := (linalg:-rowdim)(X);
  ND := (combinat:-binomial)(nd + dg - 1, dg);
  XX := matrix(ND, ND, 0);
  x := vector(nd);
  y := (linalg:-multiply)(X, x);
  strt := (combinat:-binomial)(nd + dg - 1, dg - 1) - 1;
  for i to ND do vv := (combinat:-inttovec)(strt + i, nd);
    yy := product(y[kk]^vv[kk], kk = 1 .. nd);
    for j to ND do ww := (combinat:-inttovec)(strt + j, nd);
      XX[i, j] := coeftayl(yy,
      [seq(x[kx], kx = 1 .. nd)] = [seq(0, kx = 1 .. nd)], ww);
    end do;
  end do;
  evalm(XX);
end:"outputs the symmetric power of X";
```

A Mathematical Model of the Competition between Acquired Immunity and Virus

Mikhail K. Kolev

Department of Mathematics and Computer Science,
University of Warmia and Mazury,
Olsztyn, Zolnierska 14, 10-561, Poland
kolev@matman.uwm.edu.pl

Abstract. A mathematical model describing the interactions between viral infection and acquired immunity is proposed. The model is formulated in terms of a system of partial integro-differential bilinear equations. Results of numerical experiments are presented.

Keywords: numerical modelling, kinetic theory, active particles, partial integro-differential equations, nonlinear dynamics, virus, acquired immune system.

1 Introduction

Various foreign substances and microorganisms are able to invade plant's, animal's and human's bodies. Examples of such invaders are viruses, bacteria, microbes, eukaryotes like malaria and larger parasites such as worms. Sometimes they are called foreign antigens. The term antigen may be used in two different senses. The first meaning is a substance able to induce immune response. Secondly, the term antigen may be used to denote a substance that can be specifically recognized by the immune system. Some of the invading organisms can be useful for their hosts, some of them can be harmless while other species can be very dangerous. For example, helpful microorganisms are gut flora in human digestive tract as well as Escherichia coli in the intestines. On the contrary, one of the most pathogenic invaders is the human immunodeficiency virus (HIV) causing AIDS [1]. Other sources of danger are internal invaders such as cancers.

In order to protect the organisms against foreign antigens and altered self-substances, various mechanisms have evolved. They form the immune system, which can be observed in some forms even in very simple species (like the enzyme system of bacteria protecting them against viruses). The literature devoted to the description of the main functions and properties of the immune system is enormous. We refer the readers to the books [1,21,23,28] for general immunological knowledge used in this paper as well as to [8,20,24,25,27] for reference on mathematical models of immunological processes.

The immune system is one of the most complicated systems of higher organisms. It involves various populations of molecules, cells, organs and tissues [1].

The reaction of the immune system (called the immune response) may or may not be protective [28]. Sometimes disorders of the organism lead to very weak function of the immune system when it is not able to clean the host of pathogens. Such insufficient behaviours of immune system are called immunodeficiencies. In some instances, in its attempt to combat the infection the immune response can injure the body of the host and undermine its own survival. Such situation may arise for example when the immune system "over-reacts" against pathogens and causes tissue damage. In addition, the immune response is a significant barrier to successful transplantation of organs [28].

The immune system may be classified into two main components: (i) innate immune system and (ii) acquired immune system [1,21,23,27,28]. Innate immunity refers to the basic resistance of the host to foreign agents, that the host possesses since birth. Innate defense mechanisms provide the initial protection against infections. The acquired immune system is a remarkably adaptive defense system that has evolved in vertebrates. It needs more time to develop than the innate immunity and mediates the later defenses against pathogens. Often acquired immunity is more effective than the protection performed by innate immunity [1,28].

Innate immunity, also called natural or native immunity, is the basic defense system of almost all multicellular organisms [1,21,23,28]. The natural immunity is always present in healthy individuals and changes little throughout their life. The innate defense mechanisms exist before the invasion of foreign substances to take place. These mechanisms are not intrinsically affected by prior contact with the infectious agents. Innate immunity blocks the invasion of foreign antigens and responds rapidly to pathogens that do succeed in entering host organism. In general, most of the microorganisms encountered by a healthy individual are readily cleared within a few days by innate defense mechanisms. The natural immunity is capable of sparing tissues of the host body. Usually, the host cells are not recognized by the components of innate immunity or the host cells express regulatory molecules that prevent innate response against own cells. This property of the immune system is called immune (immunological) tolerance with respect to self, or self-tolerance. If the controlling mechanisms preventing the immune system to combat self-tissues are impaired it can lead to autoimmune disease.

The mechanisms of native immunity have low level of specificity [1,21,23,28]. They are not able to very well discriminate between different kinds of foreign pathogens and thus respond to them in a generic way. Many types of cells of natural immunity can recognize and respond to the same pathogen. That is why the innate immunity is also called nonspecific immunity. Nevertheless, in many cases the nonspecific immune response is strong enough to clean the infection.

Innate immunity possesses no immunological memory: it reacts in the same, unimproved way to foreign pathogens when the host encounters them repeatedly [1,28]. Therefore, the innate immunity is non-adaptive.

Many organisms possess only innate immunity. These natural immune mechanisms have been developed and refined during very long periods of time in the process of evolution of living species. In this way organisms ensure their defense against many traumata, infections and diseases. Innate defense mechanisms are the most universal and rapidly acting immune responses. They are quite efficient insofar as severe or prolonged infections are very rare. However, various microbes, viruses and tumors which are very dangerous for humans and other vertebrates have tremendous opportunities through mutation to evolve strategies which evade the innate immune defenses. Therefore, higher organisms needed to "devise" immune mechanisms which could be dovetailed individually to such pathogens. Thus, vertebrates have evolved additional system for antigen recognition and destruction, called acquired (and also adaptive or specific) immune system. For example, its specific components called lymphocytes and antigen receptors can be observed in jawed fishes. They are better developed and more efficient in reptiles, birds, and mammals. The acquired immunity is the second line of defense in higher organisms that acts after the initial response performed by the innate immunity when invading pathogens or cancer cells resist the innate host defense mechanisms.

The reaction of acquired immunity is highly adaptive [1,28]. It is induced by certain antigens and changes the nature or quality of the response after repeated encounters with the same antigens. This capability of the acquired immune system to "remember" previous interactions with particular antigens and to enhance the response during subsequent encounters with them is called immunological memory. This property of acquired immunity is of chief importance. Cells that are able to develop faster, stronger and qualitatively better response to pathogens when they are encountered again are named memory cells. Such improved reactions to subsequent interactions with already known antigens are called secondary or recall immune responses. The nature of adaptive immunity is not constant as generally the nature of the native immunity is. Acquired immunity develops during the life course of individuals. Its response to pathogens is not as rapid as the response of the innate immunity but often is much more effective. The acquired immunity is highly specific because it is able to distinguish different antigens and reacts in a specific way to every particular pathogen. Lymphocytes of the acquired immunity use molecules expressed on their surface (the so-called antigen receptors) or secreted (the so-called antibodies or immunoglobulins) for recognition of pathogens. The specific receptors of every particular lymphocyte are able to recognize only one particular antigen through physical contact with a part of complex antigenic molecules. Such part of an antigen is called antigenic determinant or epitope [27,28]. In this way, the ability of an antigen receptor of a particular lymphocyte to uniquely recognize an epitope of a given antigen determines the specificity of this lymphocyte.

The natural immunity and the acquired immunity of higher organisms function in close cooperation. For example, some cytokines such as interleukins and IFN-γ, that are soluble proteins participating in natural immunity, stimulate the development and activity of specific T and B lymphocytes and therefore are involved in acquired immunity as well. Some nonspecific complement components participate in the development of humoral immune response, one of the major mechanisms of the specific immunity. Specific antibodies are able to coat pathogens and to bind to nonspecific phagocytes, which engulf and destroy the coated microorganisms. In such a way, antibodies play the role of opsonins in the process of phagocytosis [1,28].

The acquired immunity may be subdivided into two main types, called cell-mediated (or cellular) immunity and humoral immunity [1,21,23,28]. The main immune cells involved in the cell-mediated immunity are cells called T lymphocytes. They include cytotoxic T lymphocytes (CTLs) and T helper (T_h) cells. The cytotoxic T lymphocytes can destroy infected cells. T helper cells produce cytokines and signals inducing the proliferation and activation of immune cells. The humoral responses are performed by proteins called antibodies, which are produced by cells called B lymphocytes. As it is mentioned before, innate immunity also uses cellular and humoral components for defense. The term humoral refers to soluble substances found in body fluids (or humors), for example cytokines, antibodies etc. [1,21,23,28].

The need of development of cell-mediated and humoral parts of acquired immunity in higher organisms is a consequence of the existence of two different types of pathogen which are able to infect the host. These two types of invaders are the intracellular and the extracellular pathogens [1]. The intracellular invaders are able to enter the cells of the host and use them, for example in order to reproduce. Such pathogens are viruses and intracellular bacteria. Other invaders such as extracellular microorganisms, toxins, extraneous chemicals etc. are located in the extracellular space, i.e. outside the cells of the host. The cell-mediated acquired immunity is responsible primarily to fight off intracellular pathogens. The main function of the humoral adaptive immunity is to combat extracellular antigens [1,21,23,28].

Various viruses cause diseases, some of which like AIDS, hepatitis etc. are very dangerous. Over the past several decades various methods have been used in the field of virology for studying the nature and features of viruses. Serious advance in understanding the mechanisms of the interactions between the viruses and the immunological system has been achieved by the use of *in vitro* and *in vivo* experiments. The clinical and experimental investigations have been successfully complemented by mathematical models. The numerical and mathematical modelling of immunological phenomena provide an essential tool for description and prediction of the complex and highly nonlinear dynamics of the competition between the viruses and the immune system [4,22,26,27].

Viruses are intracellular pathogens. In order to reproduce, they must enter susceptible cells and use the metabolic machinery of the host cells. The viruses can replicate inside the infected cells, thus producing new virus particles that may leave the infected cells. The virus can destroy some of the host cells [31].

The immune system can apply innate and adaptive responses against the viruses. The specific acquired humoral response performed mainly by antibodies helps in the eradication of the free virus particles. On the other hand, the cellular defense mechanisms lead to the destruction of infected host cells by T lymphocytes.

A mathematical model of the interactions between the virus and the humoral immunity has been recently proposed and analysed in [17,18]. Furthermore, a model describing the cellular immune response to viruses has been proposed in [19]. The goal of the present paper is to extend these two models in order to describe and study the acquired immune response to viral infection.

The organization of the paper is as follows. In Section 2 we present the interacting populations and the mathematical model. Numerical approximations to the solutions of the model are constructed in Section 3. Results of our numerical experiments are presented in Section 4. Finally, Section 5 includes our concluding remarks and future directions.

2 Mathematical Model

Following the idea of Wodarz et al. [32,33] and generalizing the models proposed in [17,18] and [19] we consider the following five interacting populations, each denoted by the corresponding subscript i (see Table 1).

Table 1. Virus-acquired immune system dynamics variables

Variable i	Abbreviation	Population	Activation state $u \in [0,1]$
1	Uninfected T_h	Uninfected helper T cells	not relevant
2	Infected T_h	Infected helper T cells	virus replication, T_h destruction
3	Virus	Free virus particles	rate of infection of T_h
4	AB	Antibodies	destruction, deactivation of virus
5	CTLs	Cytotoxic T lymphocytes	destruction of infected T_h

The interacting individuals (cells or particles) are characterized by a microscopic state variable $u \in [0,1]$, which describes the specific biological function (activity) of each individual. For convenience, the values of variable u are chosen to belong to the interval $[0,1]$, describing in this way all possible values between the lowest and the highest activity of corresponding individuals. In some papers the variable u is chosen to belong to different intervals, e.g. $[-1,1]$ or $[0,\infty]$ without significant difference.

In our model, the state of activity of the infected helper T cells denotes the virus mediated killing rate of the infected cells as well as the rate of viral reproduction inside the host cell. We assume that the T helper cells infected by cytopathic viruses (i.e. viruses able to shorten the life-span of the host cells at a higher rate) possess higher activation states. Moreover, the infected cells with higher states of activity are assumed to produce larger amount of virus particles.

Here, the state of activity of free viruses denotes their ability to infect the susceptible T_h cells. The higher the ability of a virus to enter a cell, the higher the activation state of the virus.

The activation state of the population of antibodies is supposed to denote their ability to destroy viral particles and to lower their states of activity.

Further, we assume that the state of activity of the CTLs denotes their ability to destroy the infected T_h cells.

Here, the presence of internal degrees of freedom of the population of the uninfected helper T cells is neglected. For the sake of simplicity, we assume that the population denoted by $i = 1$ is independent of their activation states.

The meaning of the states of activity of the interacting populations is presented in Table 1.

We denote by

$$f_i(t,u), \quad f_i : [0,\infty) \times [0,1] \to R_+, \quad i = 1,\dots,5,$$

the distribution density of the i-th population with activation state $u \in [0,1]$ at time $t \geq 0$. Moreover, let

$$n_i(t) = \int_0^1 f_i(t,u)du, \quad n_i : [0,\infty) \to R_+, \quad i = 1,\dots,5, \tag{1}$$

be the concentration of the i-th individuals at time $t \geq 0$.

Due to the assumed independency of the distribution function $f_1(t,u)$ of the activation state u

$$f_1(t,u) = n_1(t), \quad \forall u \in [0,1], \quad t \geq 0.$$

Further, we present a mathematical model describing the dynamics of the distribution densities of the interacting populations. Respective gain, loss and conservative terms corresponding to the most important processes of production, destruction and change of activity of the individuals are included in the system (2)-(6) of partial integro-differential equations. The modelling approach utilizes the variable u that describes the biological activity of the interacting populations. This is the reason to use the term "kinetic theory for active particles" for this approach. Its application to immunology has been introduced by Bellomo and Forni for modelling the growth of cancer [7] and has been later developed in a series of papers, cf. [11,12], as well as the special issues [9,10]. We refer the readers also to the recent review papers [5,6] as well as to the book [3] devoted to description and critical analysis of the mathematical kinetic theory of active particles applied to the modelling of large living systems made up of interacting entities. The mathematical properties of the corresponding models have been extensively investigated, see [2] for complete bibliography.

Our model of the interactions between the virus and the acquired immune system is a generalization of the models of humoral response [17,18] and of the model of cellular immune response to virus [19]. It is given by the following system of partial integro-differential equations.

$$\tfrac{d}{dt}n_1(t) = S_1(t) - d_{11}n_1(t) - d_{13}n_1(t)\textstyle\int_0^1 vf_3(t,v)dv, \tag{2}$$

$$\begin{aligned}\tfrac{\partial f_2}{\partial t}(t,u) =& \, p_{13}^{(2)}(1-u)n_1(t)\textstyle\int_0^1 vf_3(t,v)dv - d_{25}f_2(t,u)\textstyle\int_0^1 vf_5(t,v)dv \\ &-d_{22}uf_2(t,u) + c_{22}\left(2\textstyle\int_0^u (u-v)f_2(t,v)dv - (1-u)^2 f_2(t,u)\right),\end{aligned} \tag{3}$$

$$\frac{\partial f_3}{\partial t}(t,u)=p_{22}^{(3)}\int_0^1 vf_2(t,v)dv - d_{33}f_3(t,u) - d_{34}f_3(t,u)\int_0^1 vf_4(t,v)dv, \tag{4}$$

$$\frac{\partial f_4}{\partial t}(t,u)=p_{34}^{(4)}(1-u)\int_0^1 f_3(t,v)dv\int_0^1 f_4(t,v)dv - d_{44}f_4(t,u), \tag{5}$$

$$\frac{\partial f_5}{\partial t}(t,u)=p_{13}^{(5)}(1-u)n_1(t)\int_0^1 f_3(t,v)dv - d_{55}f_5(t,u), \tag{6}$$

with nonnegative initial conditions

$$n_1(0) = n_1^{(0)}, \quad f_i(0,u) = f_i^{(0)}(u), \quad i = 2,3,4,5.$$

All parameters denoted by $p_{ij}^{(k)}$, d_{ij} and c_{ij} are assumed to be nonnegative and $p_{13}^{(2)} = 2d_{13}$.

The function $S_1(t)$ denotes the rate of production of uninfected T helper cells. The parameter d_{11} characterizes the natural death of the uninfected cells, which become infected by the virus with a rate proportional to their concentration as well as to the activation state of the virus (see equation (2)).

The factor $(1-u)$ in the gain term of equation (3) describes our assumption that the activity of the newly infected T helper cells is low. This is connected with the experimental observations showing that the virus needs some time after entering the host cell in order to replicate. During this period the virus particle uncoats and the viral genome is exposed. Subsequently, the viral genome is replicated and viral proteins are made. New virus particles are produced after the association of the newly generated viral proteins with the viral genomes [31]. The rate of destruction of the infected cells by the virus is assumed to be higher for cells with higher states of activity. It is described by the loss term

$$d_{22}uf_2(t,u).$$

The parameter d_{25} characterizes the rate of destruction of infected cells by CTLs which is assumed to be proportional to the activation state of CTLs. The replication of the virus particles inside the infected cells leads to increase in the probability of the destruction of the infected cells by the virus. We describe this by the conservative term

$$c_{22}\left(2\int_0^u (u-v)f_2(t,v)dv - (1-u)^2 f_2(t,u)\right)$$

corresponding to raising the activation states of the infected cells (see equation (3)).

The parameter $p_{22}^{(3)}$ characterizes the rate of reproduction of the virus inside the host cells, which is assumed to be proportional to the activation state of the infected cells (see equation (4)). The parameter d_{33} characterizes the natural death of viruses. The parameter d_{34} characterizes the rate of destruction of free viruses by antibodies.

The parameter $p_{34}^{(4)}$ characterizes the rate of production of AB, while the parameter d_{44} describes the natural death of AB.

There is experimental evidence that the newly produced CTLs and AB need time for their development and activation [21]. The factor $(1-u)$ in the gain terms of equation (5) and equation (6) describes our assumption that the activity of the newly generated CTLs and AB is low. The rate of generation of the CTLs is assumed to be proportional to the concentrations of the uninfected helper T cells and of the virus, both of which stimulate the proliferation of cytotoxic T lymphocytes [23]. The parameter d_{55} characterizes the natural death of CTLs.

3 Approximate Solution of the Model

Here, we construct a numerical solution to the concentrations of individuals $n_i(t)$, $i = 1, \dots, 5$, at any time variable $t > 0$. The concentrations $n_2(t)$, $n_3(t)$, $n_4(t)$ and $n_5(t)$ can be computed from (1) by the use of the functions $f_2(t,u)$, $f_3(t,u)$, $f_4(t,u)$ and $f_5(t,u)$. To compute numerical approximations to the functions $n_1(t)$, $f_2(t,u)$, $f_3(t,u)$, $f_4(t,u)$ and $f_5(t,u)$, we perform a discretization of the system (2)-(6) with respect to the state of activity $u \in [0,1]$ by applying the uniform grid-points

$$u_i = i\varDelta u, \quad i = 0, \dots, N,$$

where N is a positive integer and $\varDelta u = 1/N$. Then the values $f_2(t,u)$, $f_3(t,u)$, $f_4(t,u)$ and $f_5(t,u)$ in (2)-(6) can be replaced by their approximations

$$f_j(t,u_i) \approx f_{j,i}(t), \quad j = 2,3,4,5 \tag{7}$$

at the state grid-points $u_i \in [0,1]$.

For every $t > 0$ and every $u_i \in [0,1]$ with $i = 0, \dots, N$, we apply the approximations (7) for quadrature formulae to approximate the integrals:

$$\begin{aligned} &\textstyle\int_0^1 f_j(t,v)dv &&\approx Q_0^N\Big[f_j(t,v)\Big], \quad j = 3, \\ &\textstyle\int_0^1 v f_j(t,v)dv &&\approx Q_0^N\Big[v f_j(t,v)\Big], \quad j = 2,3,4,5 \\ &\textstyle\int_0^{u_i} (u_i - v) f_j(t,v)dv &&\approx Q_0^i\Big[(u_i - v) f_j(t,v)\Big], \quad j = 2. \end{aligned} \tag{8}$$

The approximations in (8) represent arbitrary quadratures. For example, in Section 4, the values $Q_0^N\Big[f_j(t,v)\Big]$, $Q_0^N\Big[v f_j(t,v)\Big]$ and $Q_0^i\Big[(u_i - v) f_j(t,v)\Big]$ are computed by the use of the composite Simpson's rule [14,30].

The approximations (7) and (8) applied to the partial integro-differential system (2)-(6) yield a system of ordinary differential equations.

This system is solved in Section 4. Its numerical solutions $f_{j,i}(t)$, with $j = 2,3,4,5$ and $i = 0, \dots, N$, are then used to compute the approximations to the functions $n_2(t)$, $n_3(t)$, $n_4(t)$ and $n_5(t)$. The approximations are computed from

$$n_j(t) \approx Q_0^N\Big[f_j(t,v)\Big], \quad j = 2,3,4,5. \tag{9}$$

4 Numerical Experiments and Discussion

The system of ordinary differential equations corresponding to the discretized model (2)-(6) is solved by using the code `ode15s` from the Matlab ODE suite [29] with $RelTol = 10^{-3}$ and $AbsTol = 10^{-4}$.

The obtained approximate solutions for the functions $f_{2,i}(t)$, $f_{3,i}(t)$, $f_{4,i}(t)$ and $f_{5,i}(t)$, with $i = 0, \dots, N$, are applied to (9) to compute the concentrations $n_2(t)$, $n_3(t)$, $n_4(t)$ and $n_5(t)$.

As initial conditions we assume the presence of uninfected T helper cells, antibodies, free virus particles, and the absence of infected T helper cells and CTLs, setting for every $i = 0, \dots, N$:

$$n_1(0) = 1, \quad f_{2,i}(0) = 0, \quad f_{3,i}(0) = 0.1, \quad f_{4,i}(0) = 0.1, \quad f_{5,i}(0) = 0.$$

In the first part of our simulation we study the interactions between viral particles and acquired immunity when only the humoral response is activated. We model this case by setting

$$d_{25} = d_{55} = p_{13}^{(5)} = 0.$$

The remaining values of the parameters of the model are set as follows:

$$S_1(t) = 100, \quad t \geq 0, \quad c_{22} = 15,$$

$$d_{11} = d_{13} = d_{33} = d_{34} = d_{44} = p_{22}^{(3)} = p_{34}^{(4)} = p_{13}^{(5)} = 100.$$

In this case the humoral response is unable to control alone the infection when the values of the parameter d_{22} are low (e.g. for $d_{22} = 49$) but clean the virus for higher values of d_{22} (for example, for $d_{22} = 55$), see Fig. 1. This parameter characterizes the rate of destruction of the infected T helper cells by the

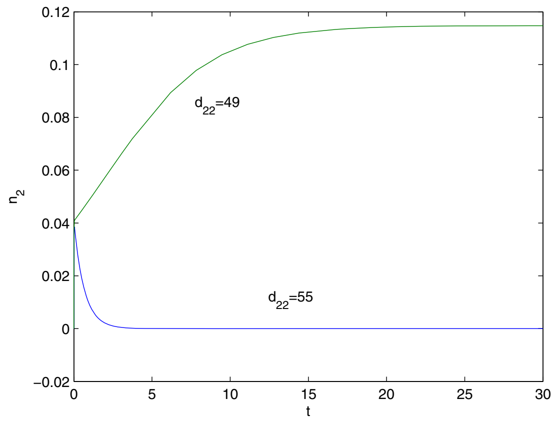


Fig. 1. Dynamics of the infected cells for $d_{22} = 49$ and $d_{22} = 55$ in the case of "humoral-only-response"

viral particles. The lower values of d_{22} correspond to non-cytopathic viruses; the higher - to cytopathic viruses.

The results of our numerical simulations confirm some experimental observations that in cases of cytopathic viral infections the humoral immunity can be sufficient to fight the infection while in cases of non-cytopathic virus the humoral immune response alone is insufficient to control the infection and additional cellular immune response performed by CTL is necessary for the successful clearance of the virus [31,34]. An example of non-cytopathic virus is lymphocytic choriomeningitis (LCMV) [15,34].

A possible explanation of these observations is the following. The cytopathic viruses are destructive enough to eradicate the majority of the infected cells. Due to the low amount of the remaining infected cells, the development of the virus is limited. This affords better opportunities to the ABs to clean the infection. On the contrary, when the organism is infected by a non-cytopathic virus, large amount of infected cells remains alive and the virus is able to replicate inside them. The newly generated viral particles can be released and then are able to infect new susceptible cells. Thus, the virus can grow and the humoral response fails.

In such cases of non-cytopathic viruses additional cellular immune response is necessary for the successful clearance of the virus. The second part of our numerical analysis is devoted to the cooperative use of humoral and cell-mediated acquired response by the immune system. As initial conditions we assume additionally the presence of CTLs:

$$f_{5,i}(0) = 0.1, \quad i = 0, \ldots, N$$

and change the values of the following parameters:

$$d_{25} = 500, d_{55} = 100, p_{13}^{(5)} = 200.$$

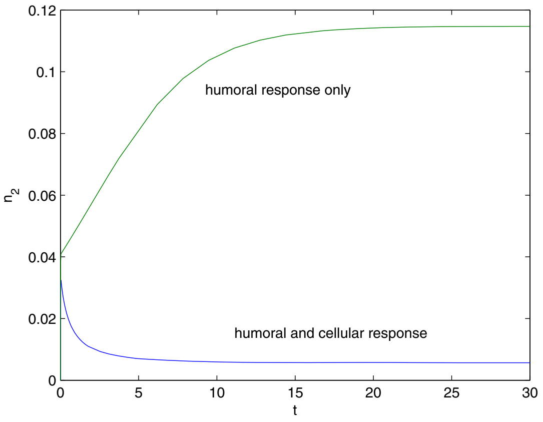


Fig. 2. Dynamics of the infected cells in cases of humoral-only vs humoral-and-cellular response for $d_{22} = 49$

The illustration of the successful cooperation of humoral and cell-mediated is presented on Fig. 2 where it is compared with the unsuccessful humoral-only response to non-cytopathic virus (in both cases $d_{22} = 49$).

The results show that while the humoral-only response is unable to fight off the infection, the cell-mediated defense leads to an additional destruction of infected cells and therefore decreases the ability of the virus to replicate. This results in a successful eradication of the infection.

5 Concluding Remarks and Future Directions

We proposed a new mathematical model of the interactions between the acquired immune system and the viruses. It describes both the humoral and the cell-mediated immune mechanisms. The results of the numerical simulations are related to cytopathic and non-cytopathic viruses and are compared with known experimental observations.

Numerical simulations utilizing mathematical models may lead to a reduction in the quantity of experimental studies performed in virology. Our future work will address the influence of other parameters of the model (2)-(6) on the competition between the viral infections and the adaptive immunity. It may lead to better understanding of these complex and highly nonlinear interactions.

The model (2)-(6) can be seen as an expert system for the problem of competition between a viral infection and an immune system. The generalized model proposed and studied numerically in this paper describes the knowledge that has been gathered in virology in specific cases described above.

We plan to further extend the model taking into account other cells and molecules participating in the interactions between the immune system and foreign pathogens. A corresponding general modelling system based on qualitative process theory [13] will be developed. Model heuristics will be mapped to a simulation system based on the computational framework of cellular automata [16].

Acknowledgement

The author wish to express his gratitude to anonymous referees for their useful comments which led to the improvements in the presentation of results.

References

1. Abbas, A.K., Lichtman, A.H.: Basic Immunology. In: Functions and Disorders of the Immune System. Elsevier, Philadelphia (2004)
2. Arlotti, L., Bellomo, N., De Angelis, E., Lachowicz, M.: Generalized Kinetic Models in Applied Sciences. World Sci., New Jersey (2003)
3. Bellomo, N.: Modelling Complex Living Systems. Birkhäuser, Boston (2007)
4. Bellomo, N., Bellouquid, A.: On the onset of nonlinearity for diffusion models of binary mixtures of biological materials by asymptotic analysis. Internat. J. Nonlinear Mech. 41(2), 281–293 (2006)

5. Bellomo, N., Bianca, C., Delitala, M.: Complexity analysis and mathematical tools towards the modelling of living systems. Physics of Life Reviews 6, 144–175 (2009)
6. Bellomo, N., Delitala, M.: From the mathematical kinetic, and stochastic game theory to modelling mutations, onset, progression and immune competition of cancer cells. Physics of Life Reviews 5, 183–206 (2008)
7. Bellomo, N., Forni, G.: Dynamics of tumor interaction with the host immune system. Math. Comput. Modelling 20(1), 107–122 (1994)
8. Bellomo, N., Li, N.K., Maini, P.K.: On the foundations of cancer modelling: Selected topics, speculations, and perspectives. Math. Models Methods Appl. Sci. 18, 593–646 (2008)
9. Bellomo, N., Maini, P.: Preface in: Cancer Modelling (II). In: Math. Models Methods Appl. Sci. 16(7b) (special issue), iii–vii (2006)
10. Bellomo, N., Sleeman, B.: Preface in: Multiscale Cancer Modelling. Comput. Math. Meth. Med. 20(2-3) (special issue), 67–70 (2006)
11. De Angelis, E., Lodz, B.: On the kinetic theory for active particles: A model for tumor-immune system competition. Math. Comput. Modelling 47(1-2), 196–209 (2008)
12. De Lillo, S., Salvatori, M.C., Bellomo, N.: Mathematical tools of the kinetic theory of active particles with some reasoning on the modelling progression and heterogeneity. Math. Comput. Modelling 45(5-6), 564–578 (2007)
13. Forbus, K.D.: Qualitative Modeling. In: van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Represantation, Foundations of Artificial Intelligence, vol. 3, pp. 361–393. Elsevier, Amsterdam (2008)
14. Gautschi, W.: Numerical Analysis: An Introduction. Birkhäuser, Boston (1997)
15. Kagi, D., Seiler, P., Pavlovic, J., Ledermann, B., Burki, K., Zinkernagel, R.M., Hengartner, H.: The roles of perforin-dependent and Fas-dependent cytotoxicity in protection against cytopathic and noncytopathic viruses. Eur. J. Immunol. 25, 3256 (1995)
16. Klienstein, S.H., Seiden, P.E.: Simulating the immune system. Computer Simulation, 69–77 (July-August 2000)
17. Kolev, M.: Mathematical modelling of the interactions between antibodies and virus. In: Proc. of the IEEE Conf. on Human System Interactions, Krakow, pp. 365–368 (2008)
18. Kolev, M.: Mathematical modelling of the humoral immune response to virus. In: Proc. of the Fourteenth National Conf. on Application of Mathematics in Biology and Medicine, Leszno, Poland, pp. 63–68 (2008)
19. Kolev, M.: Numerical modelling of cellular immune response to virus. In: Margenov, S., Vulkov, L.G., Waśniewski, J. (eds.) NAA 2008. LNCS, vol. 5434, pp. 361–368. Springer, Heidelberg (2009)
20. Kolev, M.: The immune system and its mathematical modelling. In: De Gaetano, A., Palumbo, P. (eds.) Mathematical Physiology Encyclopedia of Life Support Systems (EOLSS). Developed under the Auspices of the UNESCO, Eolss Publ., Oxford (2010), `http://www.eolss.net`
21. Kuby, J.: Immunology. W.H. Freeman, New York (1997)
22. Lollini, P.L., Motta, S., Pappalardo, P.: Modeling tumor immunology. Math. Models Methods Appl. Sci 16(7b), 1091–1125 (2006)
23. Lydyard, P.M., Whelan, A., Fanger, M.W.: Instant Notes in Immunology. BIOS Sci. Publ. Ltd., Oxford (2000)
24. Marchuk, G.I.: Mathematical Modeling of Immune Response in Infectious Diseases. Kluwer Academic Publ., Dodrecht (1997)

25. Nowak, M.A., May, R.M.: Virus Dynamics: Mathematical Principles of Immunology and Virology. Oxford Univ. Press, Oxford (2000)
26. d'Onofrio, A.: Tumor–immune system interaction and immunotherapy: Modelling the tumor–stimulated proliferation of effectros. Math. Models Methods Appl. Sci. 16(8), 1375–1402 (2006)
27. Perelson, A.S., Weisbuch, G.: Immunology for physicists. Rev. Mod. Phys. 69(4), 1219–1267 (1997)
28. Pinchuk, G.: Schaum's Outline of Theory and Problems of Immunology. McGraw-Hill, New York (2002)
29. Shampine, M.W., Reichelt, M.W.: The Matlab ODE suite. SIAM J. Sci. Comput. 18, 1–22 (1997)
30. Volkov, E.A.: Numerical Methods. Hemisphere/Mir, New York/Moscow (1990)
31. Wodarz, D.: Killer Cell Dynamics. Springer, Heidelberg (2007)
32. Wodarz, D., Bangham, C.R.: Evolutionary dynamics of HTLV-I. J. Mol. Evol. 50, 448–455 (2000)
33. Wodarz, D., Krakauer, D.C.: Defining CTL-induced pathology: implication for HIV. Virology 274, 94–104 (2000)
34. Wodarz, D., May, R., Nowak, M.: The role of antigen-independent persistence of memory cytotoxic T lymphocytes. Intern. Immun. 12, 467–477 (2000)

Some Notes upon "When Does $< \mathbb{T} >$ Equal Sat($\mathbb{T}$)?"

Yongbin Li

School of Applied Mathematic
University of Electronic Science and Technology of China
Chengdu, Sichuan 610054, China
yongbinli@uestc.edu.cn

Abstract. Given a regular set $\mathbb{T}$ in $\mathbf{K}[\mathbf{x}]$, Lemaire *et al.* in ISSAC'08 give a nice algebraic property: the regular set $\mathbb{T}$ generates its saturated ideal if and only if it is primitive. We firstly aim at giving a more direct proof of the above result, generalizing the concept of primitivity of polynomials and regular sets and presenting a new result which is equivalent to the above property. On the other hand, based upon correcting an error of the definition of U-set in AISC'06, we further develop some geometric properties of triangular sets. To a certain extent, the relation between the primitivity of $\mathbb{T}$ and its U-set is also revealed in this paper.

Keywords: Regular sets, saturated ideal, weakly primitive, C-primitive, U-set.

1 Introduction

The development and computer implementation of classical and modern elimination methods in solving *polynomial systems* in Symbolic Computation have provoked their wide applications in many areas including Artificial Intelligence. *Triangular sets* of polynomials have a fine structure suitable for representing field extensions and zeros of polynomial systems of any dimension. Methods based on triangular sets are particularly efficient for geometric problems, where the geometric properties may be inherited in the algebraic relations and some degenerate cases may be automatically ruled out in Automated Theorem Provers. Triangular decompositions are one of the studied techniques for solving polynomial systems symbolically. Invented by J.F. Ritt for systems of differential polynomials, their stride started with the method of Wu in [17,18]. The notion of a *regular set* (or *regular chain*) introduced independently by Kalkbrener in [10] and by Yang and Zhang in [19], led to important discoveries in algebraic and geometric meanings.

The theories and methods of computing regular sets of polynomial systems have significantly contributed to the theoretical and practical development of triangular decompositions such as the MMP(http://www.mmrc.iss.ac.cn/mmp/), the package Epsilon (http://www-calfor.lip6.fr/ wang/epsilon) in Maple, the

RegularChains library shipped with Maple, and others. However, there exist a common difficulty of removing redundant components.

Lemaire *et al.* in ISSAC'08, through generalizing the notion of primitivity from univariate polynomials to regular sets, establish a nice necessary and sufficient condition for a regular set to generate its saturated ideal, which reveals some algebraic meanings of regular sets. A generalization of Gauss lemma over an arbitrary commutative ring with unity is the foundation of the proof in [6]. Applying some well-known facts without generalizing Gauss lemma, we present a rather direct and alternative proof of the necessary of the above result. The new approach of proof is beneficial to understand the primitivity of regular sets. On the other hands, we also give a new equivalent condition for a regular set to generate its saturated ideal. The new condition is helpful for practical development of checking if a regular set is not primitive, such as the algorithm IsPrimitive in the RegularChains library.

Based upon the theory of the *weakly nondegenerate condition* of regular sets established by Zhang *et al.* in [21], we develop the related theory and generalize it to triangular sets, referring to [12,13,14]. These results have been applied to improve some algorithms of triangular decompositions. Specially the author defines the notion of U-set of triangular sets which has some interesting properties. The practical applications in [9,14] show that one can remove most redundant components of the algorithm CharSer by virtue of the method of Wu. But there exists an error (or flaw) in the notion of U-set in theoretical sense which affects the correctness of some results in some cases. Based upon correcting the error, we give some further properties of triangular sets considered in the complex number field. To be different from the primitivity of regular sets, these properties reveal some geometric meanings of triangular sets. The relationship of the two aspects of regular sets also is presented in this paper. These new results might contribute to develop better algorithms, and solve partly the problem of removing redundant components in most of practical developments of triangular decompositions in Computer Algebra Systems.

Let $\mathbf{K}$ be a field and $\mathbf{K}[x_1, \ldots, x_n]$ (or $\mathbf{K}[\mathbf{x}]$ for short) be the ring of polynomials in the variables $x_1, \ldots, x_n$ with coefficients in $\mathbf{K}$. The extension field $\tilde{\mathbf{K}}$ of $\mathbf{K}$ is algebraically closed in this paper. For any polynomial $p \notin \mathbf{K}$, the biggest index k such that $\deg(p, x_k) > 0$ is called the *class*, x_k the *leading variable*, $\deg(p, x_k)$ the *leading degree* of p, and $\mathrm{lcoeff}(p, x_k)$ the *leading coefficient* of p, denoted by $\mathrm{cls}(p)$, $\mathrm{lv}(p)$, $\mathrm{ldeg}(p)$ and $\mathrm{lc}(p)$, respectively. In addition, we denote $\mathrm{red}(p)$ by the polynomial $p - \mathrm{lc}(p)\mathrm{lv}(p)^{\mathrm{ldeg}(p)}$.

A *polynomial set* is a finite set $\mathbb{P}$ of nonzero polynomials in $\mathbf{K}[\mathbf{x}]$. The ideal of $\mathbf{K}[\mathbf{x}]$ generated by all elements of $\mathbb{P}$ is denoted by $<\mathbb{P}>$. The zero set of $\mathbb{P}$ in $\tilde{\mathbf{K}}^n$, denoted by $\mathrm{Zero}(\mathbb{P})$, is called the *affine variety* defined by $\mathbb{P}$. It is obvious that $\mathrm{Zero}(\mathbb{P}) = \mathrm{Zero}(<\mathbb{P}>)$.

Referring to [16], a polynomial set $\mathbb{T} \subset \mathbf{K}[\mathbf{x}] \setminus \mathbf{K}$, called triangular set, is written as the following form

$$\mathbb{T} = [f_1(\mathbf{u}, y_1), \ldots, f_s(\mathbf{u}, y_1, \ldots, y_s)], \tag{1}$$

where $\mathbf{u} = u_1, \ldots, u_r$ and $(u_1, \ldots, u_r, y_1, \ldots, y_s)$ is a permutation of $(x_1, \ldots, x_n)$, and we always assume $n - 1 \geq r \geq 1$ throughout this paper. We denote $\mathbb{T}^{\{k-1\}}$ by the triangular set $[f_1, \ldots, f_{k-1}]$ for $s > k \geq 2$. In particular, $\mathbb{T}^{\{0\}}$ stands for $\{0\}$.

The saturation of $\mathbb{T}$ is the ideal

$$< \mathbb{T} >: J^\infty = \{g \in \mathbf{K}[\mathbf{x}] \mid J^q g \in < \mathbb{T} > \text{ for some integer } q \geq 0\},$$

where $J = \prod_{c \in \mathrm{ini}(\mathbb{T})} c$ with $\mathrm{ini}(\mathbb{T}) = \{\mathrm{lc}(f_1), \ldots, \mathrm{lc}(f_s)\}$.

For any polynomial p, $\mathrm{prem}(p, \mathbb{T})$ stands for the *pseudo-remainder* of p with respect to $\mathbb{T}$ which is defined by

$$\mathrm{prem}(p, \mathbb{T}) \triangleq \mathrm{prem}(\ldots \mathrm{prem}(p, f_s, y_s), \ldots, f_1, y_1).$$

Similarly $\mathrm{res}(p, \mathbb{T})$ stands for the *resultant* of p with respect to $\mathbb{T}$. It is easy to deduce the following *pseudo-remainder formula*

$$(\prod_{i=1}^{s} \mathrm{lc}(f_i)^{d_i})p = \sum_{i=1}^{s} q_i f_i + \mathrm{prem}(p, \mathbb{T}),$$

where each d_i is a nonnegative integer and $q_i \in \mathbf{K}[\mathbf{x}]$ for $1 \leq i \leq s$.

While speaking about a polynomial system, we refer to a pair $[\mathbb{P}, \mathbb{Q}]$ of polynomial sets. The zero set of $[\mathbb{P}, \mathbb{Q}]$ defined as

$$\mathrm{Zero}(\mathbb{P}/\mathbb{Q}) \triangleq \{\mathbf{z} \in \tilde{\mathbf{K}}^n : \ p(\mathbf{z}) = 0, \ q(\mathbf{z}) \neq 0, \ \forall p \in \mathbb{P}, \ q \in \mathbb{Q}\}$$

is called the *quasi-affine variety* defined by $[\mathbb{P}, \mathbb{Q}]$.

Let R be a commutative ring with unity. We say that a non-constant polynomial $p = a_e x^e + \ldots + a_0 \in R[x]$ is *weakly primitive* if for any $\beta \in R$ such that a_e divides $\beta a_{e-1}, \ldots, \beta a_0$ then a_e divides β as well. We say the regular set $\mathbb{T}$ as (1) is *primitive* if for all $1 \leq k \leq s$, the polynomial f_k is weakly primitive in $R[y_k]$ where $R = \mathbf{K}[\mathbf{u}, y_1, \ldots, y_{k-1}]/ < \mathbb{T}^{\{k-1\}} >$. In ISSAC'08, Lemaire *et al.* give a very beautiful result: the regular set $\mathbb{T}$ generates its saturated ideal if and only if it is primitive. In Section 2, based upon a direct approach, we give a simpler proof of the above result. On the other hand, we try to generalize the concept of primitivity of polynomials and regular sets and present a new result which is equivalent to the above one.

In geometric aspect,applying the analytic method, Zhang *et al.* in [21] first establish the theory of the weakly nondegenerate condition of regular sets in $\mathbf{K}[\mathbf{x}]$. Referring to [12], we have $\mathrm{Zero}(\mathbb{T}) = \mathrm{Zero}(\mathrm{sat}(\mathbb{T}))$ if $\mathbb{T}$ is a strong regular set. In AISC'06 in [14], the above theory is extended to triangular sets in general but there is an error about the definition of $\mathbb{U}_\mathbb{T}$. In this paper, we correct the error and present some further properties of triangular sets. On the other hand, we also present a similar result: the regular set $\mathbb{T}$ generates its saturated ideal if and only if $\mathbb{U}_\mathbb{T} = \emptyset$ when $\tilde{\mathbf{K}}$ is the complex number field. Throughout the presentation, one fail to discuss algorithmic applications. We hope that these results are helpful to develop better algorithms in triangular decompositions for solving polynomial systems in Symbolic Computation.

2 Some Algebraic Properties of Triangular Sets

2.1 Preliminaries

The remarkable item (1) in the following theorem presented by Aubry *et al.* in [1], and two different proofs of it are also given by Wang in [15,16] and by Yang *et al.* in [20].

Theorem 2.1. For a regular set $\mathbb{T}$ and a polynomial p we have:
(1) $p \in \mathrm{sat}(\mathbb{T})$ if and only if $\mathrm{prem}(p, \mathbb{T}) = 0$,
(2) p is regular modulo $\mathrm{sat}(\mathbb{T})$ if and only if $\mathrm{res}(p, \mathbb{T}) \neq 0$,
(3) p is a zerodivisor modulo $\mathrm{sat}(\mathbb{T})$ if and only if $\mathrm{res}(p, \mathbb{T}) = 0$ and $\mathrm{prem}(p, \mathbb{T}) \neq 0$.

Refer to [3,6] for the proof of items (2) and (3). An ideal in $\mathbf{K}[\mathbf{x}]$ is *unmixed* if all its associated primes have the same dimension. In particular, an unmixed ideal has no embedded associated primes.

Theorem 2.2. Let $\mathbb{T} = [f_1, \ldots, f_s]$ with $s > 1$ be a regular set in $\mathbf{K}[\mathbf{x}]$. The following properties hold:
(1) $\mathrm{sat}(\mathbb{T})$ is an unmixed ideal with dimension $n - s$,
(2) $\mathrm{sat}(\mathbb{T} \cap \mathbf{K}[x_1, \ldots, x_i]) = \mathrm{sat}(\mathbb{T}) \cap \mathbf{K}[x_1, \ldots, x_i]$,
(3) $\mathrm{sat}(\mathbb{T}) =< \mathrm{sat}(\mathbb{T}^{\{s-1\}}) \cup \{f_s\} >: \mathrm{lc}(f_s)^\infty$.

For the proofs, one can refer to Boulier *et al.* in [2] and Gao and Chou in [7,8] for item (1), to Aubry *et al.* in [1] for item (2), and to Kalkbrener in [10] for item (3).

Let R be a commutative Noetherian ring with unity. One can easily give the proofs of the following Lemmas 2.1, 2.2. and 2.3 which can be found in [6].

Lemma 2.1. Let I be a proper ideal of R and let h be an element of R. Then h is regular modulo I if and only if $I = I : h^\infty$ holds.

Lemma 2.2. Let I be a proper ideal of R and $G = \{h \in R|\ h \text{ is regular modulo } I\}$. Then G endowed with the multiplication operation in R is a semi-group.

Lemma 2.3 (Mc Coy Lemma). A non-zero polynomial $f \in R[x]$ is a zero-divisor if and only if there exists a non-zero element $a \in R$ such that $af = 0$ holds.

For $p \in R[x]\backslash\{0\}$, $\mathbb{C}_p$ stands for the set of all the nonzero coefficients of p in x. The next notions of weak primitivity of polynomials and regular sets are presented in [6].

Definition 2.1. Let $p = a_0 + \ldots + a_e x^e \in R[x]$ with $e \geq 1$. The polynomial p is *weakly primitive* if either condition holds:
(i) a_e is invertible in R if $|\mathbb{C}_p| = 1$;
(ii) for any $\beta \in R$ such that $a_e|\beta b$ for all $b \in \mathbb{C}_{\mathrm{red}(p)}$, we have $a_e|\beta$ as well.

Remark: The above description, which is somewhat different from the one in [6], is helpful for illustrating the weak primitivity when $p = a_e x^e$ and a_e is invertible without citing the strong primitivity in [6].

Definition 2.2. Let $\mathbb{T} = [f_1, \ldots, f_s] \subset \mathbf{K}[x_1, \ldots, x_n]$ be a regular set. We say $\mathbb{T}$ is *primitive* if for all $1 \leq k \leq s$, f_k is weakly primitive in $R[y_k]$ where $y_k = \mathrm{lv}(f_k)$ and

$$R = \mathbf{K}[\mathbf{u}, y_1, \ldots, y_{k-1}]/ < \mathbb{T}^{\{k-1\}} > .$$

The next nice result is presented by Lemaire *et al.* in [6].

Theorem 2.3. Let $\mathbb{T}$ be a regular set. Then $\mathbb{T}$ is primitive if and only if $< \mathbb{T} >= \mathrm{sat}(\mathbb{T})$.

Lemma 2.4. Let R be a Noetherian commutative ring with unity. Consider $g \in R[x]$ with $\deg(g, x) > 0$ and $b \in R$. If g is a zerodivisor modulo $< b >$, then there exists an element $a \notin < b >$ such that $ac \in < b >$ for any $c \in \mathbb{C}_g$.

Proof: Consider the ring $R/ < b >$, it is easy to check that $R/ < b >$ is also a Noetherian commutative ring with unity. And g is also a zerodivisor in $R/ < b > [x]$. According to Lemma 2.3, there exists a non-zero element $\bar{a} \in R/ < b >$ such that $\bar{a}g = 0$ holds in $R/ < b > [x]$. Now set $a \in R$ such that $a - \bar{a} \in < b >$. It implies $ag \in < b >$. Thus $ac \in < b >$ for any $c \in \mathbb{C}_g$. This completes the proof. □

The next lemma is Proposition 6.5 in [6].

Lemma 2.5. Let R be a Noetherian commutative ring with unity. Consider a polynomial $f = \sum_{i=0}^{n} a_i x^i \in R[x]$. Assume that n is at least 1 and a_n is regular in R. Then $< f >=< f >: a_n^\infty$ holds if and only if a_n is invertible in R, or $\mathrm{red}(f)$ is regular modulo $< a_n >$.

2.2 Main Results

The following results are helpful to understand any primitive regular set $\mathbb{T}$ with $|\mathbb{T}| = 1$.

Proposition 2.1. Let R be a Noetherian commutative ring with unity and $p = a_0 + \ldots + a_e x^e \in R[x]$ with $e \geq 1$. If a_e is regular in R and p is weakly primitive, then $< p >=< p >: a_e^\infty$.

Proof: When a_e is invertible in R, it is easy to see that $< p >=< p >: a_e^\infty$. Now we only consider the case that regular element a_e is not invertible in R. We claim that $\mathrm{red}(p)$ is regular modulo $< a_e >$.

In order to prove our claim, we suppose $\mathrm{red}(p)$ is not regular modulo $< a_e >$, it means that $\mathrm{red}(p)$ is a zerodivisor modulo $< a_e >$. We proceed to show p is not weakly primitive. Applying Lemma 2.4, there exists $\breve{a} \in R$ with $\breve{a} \notin < a_e >$ such that $a\breve{a} \in < a_e >$ for each $a \in \mathbb{C}_{\mathrm{red}(p)}$.

Since p is weakly primitive, for any $\beta \in R$, if $a_e | \beta a$ for all $a \in \mathbb{C}_{\mathrm{red}(p)}$, then we have $a_e | \beta$. We let $\beta = \breve{a}$, it is obvious that $a_e | \beta a$ since $a\breve{a} \in < a_e >$ for any $a \in \mathbb{C}_{\mathrm{red}(p)}$.

Thus we have $a_e | \beta = \breve{a}$. This contradicts the fact $\breve{a} \notin < a_e >$. Therefore, our assumption that $\mathrm{red}(p)$ is not regular modulo $< a_e >$ is impossible. So $\mathrm{red}(p)$

is regular modulo $< a_e >$. By virtue of Lemma 2.5, we get $< p >=< p >: a_e^\infty$. This completes the proof. $\square$

Corollary 2.1. Let $\mathbb{T}$ be a regular set with $|\mathbb{T}| = 1$ in $\mathbf{K}[\mathbf{x}]$. If $\mathbb{T}$ is primitive, then $< \mathbb{T} >= \mathrm{sat}(\mathbb{T})$.

We proceed to present a direct and simpler proof of the half of Theorem 2.3, which is described by the following theorem.

Theorem 2.4. Let a regular set $\mathbb{T} = [f_1, \ldots, f_s]$ in $\mathbf{K}[\mathbf{x}]$ be primitive. Then $< \mathbb{T} >= \mathrm{sat}(\mathbb{T})$.

Proof: When $s = 1$, our result holds true by virtue of Corollary 2.1. Now assume $s \geq 2$. By induction,

$$\mathrm{sat}(\mathbb{T}^{\{k-1\}}) =< \mathbb{T}^{\{k-1\}} >$$

holds, we proceed to show $\mathrm{sat}(\mathbb{T}^{\{k\}}) =< \mathbb{T}^{\{k\}} >$ holds too. To do so, we consider $p \in \mathrm{sat}(\mathbb{T}^{\{k\}})$ and prove $p \in< \mathbb{T}^{\{k\}} >$. If $\mathrm{lv}(p) = y_i$ with $i > k$, then $p \in \mathrm{sat}(\mathbb{T}^{\{k\}})$ if and only if all coefficients of p w.r.t y_i are in $\mathrm{sat}(\mathbb{T}^{\{k\}})$. So we can concentrate on the case $p \in \mathbf{K}[\mathbf{u}, y_1, \ldots, y_k]$.

Consider in $\mathbf{K}[\mathbf{u}, y_1, \ldots, y_k]$, by virtue of Theorem 2.2 we have

$$\mathrm{sat}(\mathbb{T}^{\{k\}}) =< \mathrm{sat}(\mathbb{T}^{\{k-1\}}), f_k >: I_k^\infty =< \mathbb{T}^{\{k-1\}}, f_k >: I_k^\infty =< \mathbb{T}^{\{k\}} >: I_k^\infty$$

where $I_k = \mathrm{lc}(f_k)$.

We claim that I_k is regular modulo $< \mathbb{T}^{\{k\}} >$ in $\mathbf{K}[\mathbf{u}, y_1, \ldots, y_k]$. In order to show our claim, we consider $f_k \in R[y_k]$ where $R = \mathbf{K}[\mathbf{u}, y_1, \ldots, y_{k-1}]/ < \mathbb{T}^{\{k-1\}} >$.

One can easily check that R is a Noetherian commutative ring with unity. Since $\mathbb{T}$ is primitive, we know that f_k is weakly primitive in $R[y_k]$. Then we have $< f_k >=< f_k >: I_k^\infty$ in $R[y_k]$ according to Proposition 2.1. By virtue of Lemma 2.1, I_k is regular modulo $< f_k >$ in $R[y_k]$.

In order to prove that I_k is regular modulo $< \mathbb{T}^{\{k\}} >=< \mathbb{T}^{\{k-1\}} > + < f_k >$ in $\mathbf{K}[\mathbf{u}, y_1, \ldots, y_k]$, we first claim that $I_k \neq 0$ modulo $< \mathbb{T}^{\{k\}} >$. Indeed,if $I_k \in< \mathbb{T}^{\{k\}} >$, then $\mathrm{prem}(I_k, \mathbb{T}^{\{k\}}) = \mathrm{prem}(I_k, \mathbb{T}^{\{k-1\}}) = 0$ by Theorem 2.1 and the pseudo-division formula. Consequently, $I_k \in< \mathbb{T}^{\{k-1\}} >$ in $\mathbf{K}[\mathbf{u}, y_1, \ldots, y_k]$ by virtue of $\mathrm{sat}(\mathbb{T}^{\{k-1\}}) =< \mathbb{T}^{\{k-1\}} >$ and the fact that $\mathbb{T}^{\{k-1\}}$ is a regular set. Thus $I_k = 0$ in $R[y_k]$. Note that I_k is also regular modulo $< \mathbb{T}^{\{k-1\}} >$ in $\mathbf{K}[\mathbf{u}, y_1, \ldots, y_{k-1}]$ since $\mathbb{T}^{\{k\}}$ is a regular set. This is impossible.

Secondly we are ready to show I_k is not a zerodivisor modulo $< \mathbb{T}^{\{k\}} >$ in $R[y_k]$. Indeed, we suppose there exists some $g \in \mathbf{K}[\mathbf{u}, y_1, \ldots, y_k]$ which is not zero modulo $< \mathbb{T}^{\{k\}} >$, so $g \neq 0$ modulo $< f_k >$ in $R[y_k]$,such that $I_k g \in< \mathbb{T}^{\{k\}} >$ in $\mathbf{K}[\mathbf{u}, y_1, \ldots, y_k]$. We proceed to show that it is impossible.

Consider $I_k g \in< \mathbb{T}^{\{k\}} >\subseteq \mathrm{sat}(\mathbb{T}^{\{k\}})$, we have $\mathrm{prem}(I_k g, \mathbb{T}^{\{k\}}) = 0$ by Theorem 2.1. By the pseudo-division formula, there exit a non-negative integer $t \geq 1$ and $q \in \mathbf{K}[\mathbf{u}, y_1, \ldots, y_k]$ such that

$$I_k^t g = q f_k + \mathrm{prem}(I_k g, f_k); \;\; \mathrm{prem}(\mathrm{prem}(I_k g, f_k), \mathbb{T}^{\{k-1\}}) = 0.$$

It follows from $\mathrm{sat}(\mathbb{T}^{\{k-1\}}) =< \mathbb{T}^{\{k-1\}} >$ that $\mathrm{prem}(I_k g, f_k) = 0$ in $R[y_k]$. Thus, we have $I_k^t g = q f_k$ in $R[y_k]$. Hence $I_k^t g$ is zero modulo $< f_k >$ in $R[y_k]$. Note that I_k is regular modulo $< f_k >$ in $R[y_k]$. By virtue of Lemma 2.2, I_s^t is also regular modulo $< f_k >$. This contradicts the fact $g \neq 0$ modulo $< f_k >$ in $R[y_k]$.

Thus I_k is regular modulo $< \mathbb{T}^{\{k\}} >$. It implies $\mathrm{sat}(\mathbb{T}^{\{k\}}) =< \mathbb{T}^{\{k\}} >$ in $\mathbf{K}[\mathbf{u}, y_1, \ldots, y_k]$ by lemma 2.1. Consequently, $\mathrm{sat}(\mathbb{T}^{\{k\}}) =< \mathbb{T}^{\{k\}} >$. This completes the proof. □

In the following definitions, we try to give a stronger notion of primitivity of polynomials and triangular sets. By the similar argument in other part proof of Theorem 4.4 in [6], we will prove a generalizing result.

Definition 2.3. Let $p \in R[x]$ with $\deg(p, x) \geq 1$. The polynomial p is *C- weakly primitive* if either condition holds:

(i) a_e is invertible in R if $|\mathbb{C}_p| = 1$;

(ii) for any regular element $a \in \mathbb{C}_p$ and any $\beta \in R$, if $a|\beta b$ for all $b \in \mathbb{C}_p \setminus \{a\}$, we have $a|\beta$ as well.

Definition 2.4. Let $\mathbb{T} = [f_1, \ldots, f_s] \subset \mathbf{K}[x_1, \ldots, x_n]$ be a regular set. We say $\mathbb{T}$ is *C-primitive* if for all $1 \leq k \leq s$, f_k is C-weakly primitive in $R[y_k]$ where $y_k = \mathrm{lv}(f_k)$ and

$$R = \mathbf{K}[\mathbf{u}, y_1, \ldots, y_{k-1}]/ < \mathbb{T}^{\{k-1\}} > .$$

Remark: It is obvious that $\mathbb{T}$ is primitive if it is C-primitive. But the inverse does not hold true.

Proposition 2.2. Let R be a Noetherian commutative ring with unity and $p = a_0 + \ldots + a_e x^e \in R[x]$ with $e \geq 1$. If a_e is regular in R and $< p >=< p >: a_e^\infty$, then p is C-weakly primitive.

Proof: We suppose that p is not C-weakly primitive. Then there exist some regular element $a_i \in \mathbb{C}_p$ and $\beta \in R$ such that

$$\forall b \in \mathbb{C}_p \backslash \{a_i\}, \;\; a_i | \beta b \;\; \textit{and} \;\; a_i \nmid \beta.$$

Consequently, there exist $d_k \in R$ such that $\beta a_k = a_i d_k$ for any $k \neq i$. Let

$$q = d_e x^e + \ldots + d_{i+1} x^{i+1} + \beta x^i + d_{i-1} x^{i-1} + \ldots + d_0,$$

it is obvious that $\beta p = a_i q$. We claim $a_e d_k = a_k d_e$ for any $k \neq i$. In fact, one can see that $a_i a_e d_k = a_i d_k a_e = \beta a_k a_e$ and $a_i a_k d_e = a_i d_e a_k = \beta a_e a_k$. This implies $a_i(a_e d_k - a_k d_e) = 0$ in R. Since a_i is a regular element in R, we have $a_e d_k = a_k d_e$ for any $k \neq i$.

We proceed to show that $q \in < p >: a_e^\infty$ in $R[x]$. Indeed, it follows from

$$\begin{aligned} a_e q &= a_e d_e x^e + \ldots + a_e d_{i+1} x^{i+1} + a_e \beta x^i + a_e d_{i-1} x^{i-1} + \ldots + a_e d_0 \\ &= a_e d_e x^e + \ldots + a_{i+1} d_e x^{i+1} + d_e a_i x^i + d_e a_{i-1} x^{i-1} + \ldots + d_e a_0 = d_e p \end{aligned}$$

that $q \in < p >: a_e^\infty$. Thus, $q \in < p >$.

It means there exists $\alpha \in R[x]$ such that $q = \alpha p$ in $R[x]$. By the construction of q, $\deg(q, x) = \deg(p, x)$. Hence $\alpha \in R$ and $\beta = \alpha a_i$. This contradicts $a_i \nmid \beta$. Thus p is C-weakly primitive. This completes the proof. $\square$

Theorem 2.5. Let a regular set $\mathbb{T} \subset \mathbf{K}[x_1, \ldots, x_n]$ such that $< \mathbb{T} >= \mathrm{sat}(\mathbb{T})$. Then $\mathbb{T}$ is C-primitive.

Proof: The theorem holds true when $|\mathbb{T}| = s = 1$ by virtue of Proposition 2.2. We only need to show that $< \mathbb{T} >$ is C-primitive when $|\mathbb{T}| = s > 1$. By induction, $\mathbb{T}^{\{k-1\}}$ is C-primitive, we proceed to prove that $\mathbb{T}^{\{k\}}$ is also C-primitive.

Let $f_k = a_e y_k^e + \ldots + a_0$. We know $\mathrm{sat}(\mathbb{T}^{\{k-1\}}) =< \mathbb{T}^{\{k-1\}} >$. Consider $f_k \in R[y_k]$ with $R = \mathbf{K}[\mathbf{u}, y_1, \ldots, y_{k-1}]/ < \mathbb{T}^{\{k-1\}} >$. Note that a_e is regular in R. Furthermore, we have

$$< \mathbb{T}^{\{k\}} >= \mathrm{sat}(\mathbb{T}^{\{k\}}) =< \mathrm{sat}(\mathbb{T}^{\{k-1\}}), f_k >: a_e^{\infty} =< \mathbb{T}^{\{k-1\}}, f_k >: a_e^{\infty}.$$

So $< f_k >=< f_k >: a_e^{\infty}$ in $R[y_k]$. Thus f_k is C-weakly primitive by virtue of Proposition 2.2. Consequently, $\mathbb{T}^{\{k\}}$ is also C-primitive. This completes the proof. $\square$

The following assertion which is helpful for checking if $< \mathbb{T} >$ is not primitive, is obvious by virtue of Theorem 2.3.

Corollary 2.2. Let a regular set $\mathbb{T} \subset \mathbf{K}[x_1, \ldots, x_n]$. Then $< \mathbb{T} >$ is primitive if and only if it is C-primitive.

3 Geometric Properties

In this section, we consider $\tilde{\mathbf{K}}$ as the complex number field, $\tilde{\mathbf{K}}^n$ is also considered as the topological space $\tilde{\mathbf{K}}^n$ induced by the following metric, $|\mathbf{z} - \mathbf{z}^*| = \max\{|x_1 - x_1^*|, |x_2 - x_2^*|, \ldots, |x_n - x_n^*|\}$ for any $\mathbf{z}, \mathbf{z}^* \in \tilde{\mathbf{K}}^n$. Let $\mathbf{S}$ be a nonempty subset in $\tilde{\mathbf{K}}^n$, we write $\overline{\mathbf{S}}^E$ as the topological closure of $\mathbf{S}$. The Zariski closure of $\mathbf{S}$ is denoted by $\overline{\mathbf{S}}$. Given a triangular set $\mathbb{T}$ as (1) in $\mathbf{K}[\mathbf{x}]$, for any $\bar{\mathbf{z}} = (\bar{\mathbf{u}}, \bar{y}_1, \ldots, \bar{y}_s) \in \mathrm{Zero}(\mathbb{T})$, we write $\bar{\mathbf{z}}^{\{j\}}$ for $\bar{\mathbf{u}}, \bar{y}_1, \ldots, \bar{y}_j$ or $(\bar{\mathbf{u}}, \bar{y}_1, \ldots, \bar{y}_j)$ with $\bar{\mathbf{u}} = \bar{\mathbf{z}}^{\{0\}}$ and $\bar{\mathbf{z}} = \bar{\mathbf{z}}^{\{s\}}$.

3.1 Preliminaries

Applying the analytic method, Zhang et al. in [21] establish the theory of the weakly nondegenerate condition of regular sets in $\mathbf{K}[\mathbf{x}]$. Let $\mathbb{T}$ be a regular set, a zero $\mathbf{z}_0 \in \mathrm{Zero}(\mathbb{T})$ is called a *quasi-normal zero* if $\mathbf{z}_0^{\{i-1\}} \notin \mathrm{Zero}(\mathbb{C}_{f_i})$ for any $1 \leq i \leq s$, also said to be satisfying the *nondegenerate condition* (see [21] for details). $\mathbb{T}$ is called a *strong regular set* if every zero of $\mathbb{T}$ is also a quasi-normal zero. Referring to [12,13], we have $\mathrm{Zero}(\mathbb{T}) = \mathrm{Zero}(\mathrm{sat}(\mathbb{T}))$ if $\mathbb{T}$ is a strong regular set.

In AISC'06, the above theory is extended to triangular sets in general. The next definition in [14] is an extension of the concept of quasi-normal zero of regular sets.

Definition 3.1. Let $\mathbb{T} = [f_1, \ldots, f_s]$ be a triangular set in $\mathbf{K}[\mathbf{x}]$. A zero $\mathbf{z}_0 \in \mathrm{Zero}(\mathbb{T})$ is called a *quasi-normal zero* of $\mathbb{T}$ if for any $1 \leq k \leq s$, either condition holds:

a. $\mathrm{lc}(f_k)(\mathbf{z}_0^{\{k-1\}}) \neq 0$;

b. $\mathbf{z}_0^{\{k-1\}} \notin \mathrm{Zero}(\mathbb{C}_{f_k})$ if $\mathrm{res}(\mathrm{lc}(f_k), \mathbb{T}) \neq 0$.

Definition 3.2. Let $\mathbb{T} = [f_1, \ldots, f_s]$ be a triangular set in $\mathbf{K}[\mathbf{x}]$. We call the following set U-set of $\mathbb{T}$, denoted by $\mathbb{U}_\mathbb{T}$,

$$\begin{aligned}\mathbb{U}_\mathbb{T} \triangleq \{c|\ &\mathrm{res}(c, \mathbb{T}) = 0,\ \text{for } c \in \mathrm{ini}(\mathbb{T})\}\\ &\cup\{\text{one } c \in \mathbb{C}_{f_k} |\ \mathrm{res}(\mathrm{lc}(f_k), \mathbb{T}^{\{k-1\}}) \neq 0, \mathrm{Zero}(\mathbb{T}^{\{k-1\}} \cup \mathbb{C}_{f_k}) \neq \emptyset,\ \text{for } f_k \in \mathbb{T}\}.\end{aligned}$$

Remark: The definition is different from the one in [14] which has a theoretic error which will lead to a mistaken result in some cases. In our improved definition of $\mathbb{U}_\mathbb{T}$, the algorithms and results except Proposition 13 presented in [14] hold true.

Example 3.1. Let a triangular set $\mathbb{T} = [f_1, f_2, f_3]$ in $\mathbf{K}[x_1, x_2, x_3, x_4]$ under $x_1 \prec x_2 \prec x_3 \prec x_4$, where

$$\begin{aligned}f_1 &= -x_2^2 + x_1,\\ f_2 &= -x_2 x_3^3 + (2x_1 - 1)x_3^2 - x_2(x_1 - 2)x_3 - x_1,\\ f_3 &= (x_2 x_3 + 1)x_4 + x_1 x_3 + x_2.\end{aligned}$$

By the above notations, we know

$$\mathbb{C}_{f_1}=\{-1, x_1\},\ \mathbb{C}_{f_2}=\{-x_2, 2x_1-1, -x_2(x_1-2), -x_1\},\ \mathbb{C}_{f_3}=\{x_2x_3+1, x_1x_3+x_2\}.$$

Since $\mathrm{res}(\mathrm{lc}(f_3), \mathbb{T}) = 0$, we have

$$\mathbb{U}_\mathbb{T} = \{\mathrm{lc}(f_3)\} = \{x_2 x_3 + 1\}.$$

For any triangular set $\mathbb{T} \subset \mathbf{K}[\mathbf{x}]$, we denote $\mathrm{QnZero}(\mathbb{T})$ by the set of all quasi-normal zeros of $\mathbb{T}$. The following results are presented in [14].

Theorem 3.1. For any triangular set $\mathbb{T}$ in $\mathbf{K}[\mathbf{x}]$, we have

$$\mathrm{Zero}(\mathbb{T}/\mathbb{U}_\mathbb{T}) \subseteq \overline{\mathrm{QnZero}(\mathbb{T})}^E \subseteq \mathrm{Zero}(\mathrm{sat}(\mathbb{T})).$$

3.2 Main Results

Lemma 3.1. Let a nonempty subset $\mathbf{S} \subseteq \tilde{\mathbf{K}}^n$. Then, $\overline{\mathbf{S}} = \overline{\overline{\mathbf{S}}^E}$.

Proof: We denote the ideal $\mathrm{Ideal}(\mathbf{S})$ by the following set

$$\{f \in \tilde{\mathbf{K}}[\mathbf{x}] :\ f(\mathbf{z}) = 0 \text{ for all } \mathbf{z} \in \mathbf{S}\}.$$

We claim that $\mathrm{Ideal}(\mathbf{S}) = \mathrm{Ideal}(\overline{\mathbf{S}}^E)$. It is obvious that $\mathrm{Ideal}(\mathbf{S}) \supseteq \mathrm{Ideal}(\overline{\mathbf{S}}^E)$.

To get the reverse containment relation, suppose $f \in \mathrm{Ideal}(\mathbf{S})$. By the continuity of the function determined by f, we know that $f(\mathbf{z}) = 0$ for any $\mathbf{z} \in \overline{\mathbf{S}}^E$. Thus, $f \in \mathrm{Ideal}(\overline{\mathbf{S}}^E)$. Since $f \in \mathrm{Ideal}(\mathbf{S})$ is arbitrary, we have $\mathrm{Ideal}(\mathbf{S}) \subseteq \mathrm{Ideal}(\overline{\mathbf{S}}^E)$. This completes the proof of the assertion. □

By the similar argument in the proof of Theorem 3.1 in [14], we can prove easily the following result.

Lemma 3.2. For any triangular set $\mathbb{T}$, we have

$$\mathrm{Zero}(\mathbb{T}/\mathrm{ini}(\mathbb{T})) \subseteq \overline{\mathrm{Zero}(\mathbb{T}/\mathbb{U}_{\mathbb{T}})}^E.$$

Lemma 3.3. Let $\mathbb{T}$ be a triangular set and g a polynomial in $\mathbf{K}[\mathbf{x}]$, we have

$$\mathrm{Zero}(<\mathbb{T}>: g^{\infty}) = \overline{\mathrm{Zero}(\mathbb{T}/\{g\})}.$$

Proof: By the Ascending Chain Condition, one can easily see that there is an integers $m > 0$ such that

$$<\mathbb{T}>: g^{\infty} =<\mathbb{T}>: g^m =<\mathbb{T}>: g^{m+l},$$

for any integer l.

Applying Theorem 7 in [4], we have

$$\mathrm{Zero}(<\mathbb{T}>: g^m) \supset \overline{\mathrm{Zero}(<\mathbb{T}>) \setminus \mathrm{Zero}(\{g^m\})}.$$

Note that $\mathrm{Zero}(\mathbb{T}) = \mathrm{Zero}(<\mathbb{T}>)$. It implies that

$$\mathrm{Zero}(<\mathbb{T}>: g^m) \supset \overline{\mathrm{Zero}(\mathbb{T}/\{g\})}.$$

To get the reverse containment relation, suppose $\mathbf{z} \in \mathrm{Zero}(<\mathbb{T}>: g^m)$. Equivalently, if $Hg^t \in <\mathbb{T}>$ for some integer $t \geq m$, then $H(\mathbf{z}) = 0$.

Now let $H \in \mathrm{Ideal}(\mathrm{Zero}(\mathbb{T}/\{g\}))$. It is easy to see that Hg^m vanishes on $\mathrm{Zero}(\mathbb{T})$ or $\mathrm{Zero}(<\mathbb{T}>)$. Thus, by Hilbert's Nullstellensatz, $Hg^m \in \sqrt{<\mathbb{T}>}$. That is to say that there is some integer $k_0 > 0$ such that

$$H^{k_0} g^{mk_0} \in <\mathbb{T}>.$$

Thus $H^{k_0}(\mathbf{z}) = 0$, so $\mathbf{z} \in \overline{\mathrm{Zero}(\mathbb{T}/\{g\})}$. This means that

$$\mathrm{Zero}(<\mathbb{T}>: g^m) \subset \overline{\mathrm{Zero}(\mathbb{T}/\{g\})}.$$

This completes the proof of the lemma. □

We state the main result in this paper as the follows. From Lemmas 3.1 and 3.2 and Theorem 3.1 follow easily that the next result.

Theorem 3.2. For any triangular set $\mathbb{T}$, we have

$$\mathrm{Zero}(\mathrm{sat}(\mathbb{T})) = \mathrm{Zero}(<\mathbb{T}>: U^{\infty})$$

where $U = \prod_{u \in \mathbb{U}_{\mathbb{T}}} u$.

Proof: From Lemmas 3.1 and 3.2 and Theorem 3.1 imply that

$$\mathrm{Zero}(\mathbb{T}/\mathrm{ini}(\mathbb{T})) \subseteq \overline{\mathrm{Zero}(\mathbb{T}/\mathbb{U}_{\mathbb{T}})}^{E} = \overline{\mathrm{Zero}(\mathbb{T}/\mathbb{U}_{\mathbb{T}})} \subseteq \mathrm{Zero}(\mathrm{sat}(\mathbb{T})) = \overline{\mathrm{Zero}(\mathbb{T}/\mathrm{ini}(\mathbb{T}))}.$$

Thus,

$$\overline{\mathrm{Zero}(\mathbb{T}/\mathbb{U}_{\mathbb{T}})} = \mathrm{Zero}(\mathrm{sat}(\mathbb{T})).$$

Consequently, by virtue of Lemma 3.3, we complete the proof. □

Corollary 3.1. For any triangular set $\mathbb{T}$ as the above,

$$\mathrm{Zero}(\mathrm{sat}(\mathbb{T})) = \overline{\mathrm{Zero}(\mathbb{T}/\mathbb{U}_{\mathbb{T}})} = \overline{\mathrm{QnZero}(\mathbb{T})}.$$

We are ready to show, to a certain extent, some relation between the primitivity of $\mathbb{T}$ and $\mathbb{U}_{\mathbb{T}}$. In order to prove the relation, the following lemma which is easy to prove is useful.

Lemma 3.4. Let $\mathbb{P}$ be any polynomial set in $\mathbf{K}[\mathbf{x}]$. If $< \mathbb{P} >=< 1 >$ in $\tilde{\mathbf{K}}[\mathbf{x}]$, then $< \mathbb{P} >=< 1 >$ in $\mathbf{K}[\mathbf{x}]$.

Theorem 3.5. Let $\mathbb{T} = [f_1, \ldots, f_s]$ be a regular set in $\mathbf{K}[\mathbf{x}]$. If $\mathbb{U}_{\mathbb{T}} = \emptyset$, then $\mathbb{T}$ is primitive.

Proof: Let $\mathbb{T} = [f_1, \ldots, f_s]$. Since $\mathbb{T}$ is a regular set in $\mathbf{K}[\mathbf{x}]$, by the definition of $\mathbb{U}_{\mathbb{T}}$, we have $\mathrm{Zero}(\mathbb{C}_{f_1}) = \emptyset$ and $\mathrm{Zero}(\mathbb{C}_{f_k} \cup \mathbb{T}^{\{k-1\}}) = \emptyset$ for any $k = 2, \ldots, s$ if $s > 1$. By virtue of Hilbert's Nullstellensatz and lemma 3.4, we have $< \mathbb{C}_{f_1} >=< 1 >$ and $< \mathbb{T}^{\{k-1\}} \cup \mathbb{C}_{f_k} >=< 1 >$ in $\mathbf{K}[\mathbf{x}]$ for any $k = 2, \ldots, s$ if $s > 1$.

When $s = 1$, let $\mathbb{T} = [f_1(\mathbf{u}, y_1)]$ with $f_1 = a_0 + \ldots + a_e y_1^e \in R[y_1]$ with $R = \mathbf{K}[\mathbf{u}]$, we know a_e is a regular element in $\mathbf{K}[\mathbf{u}]$ since $\mathbb{T}$ is a regular set. It follows from $< \mathbb{C}_{f_1} >=< 1 >$ that there exist $b_i \in R[y_1]$ for $i = 0, \ldots, e$.

$$a_0 b_0 + a_1 b_1 + \ldots + a_{e-1} b_{e-1} + a_e b_e = 1.$$

For any $\beta \in R$ such that $a_e | \beta a_j$ for $j = 0, 1, \ldots, e-1$, it follows from $a_0 b_0 \beta + a_1 b_1 \beta + \ldots + a_{e-1} b_{e-1} \beta + a_e b_e \beta = \beta$ that $a_e | \beta$. So $\mathbb{T}$ is primitive.

Now assume $s \geq 2$, we prove the theorem by induction. Suppose $\mathbb{T}^{\{k-1\}}$ is primitive. Similarly suppose $f_k = c_0 + \ldots + c_e y_k^e \in R^{\star}[y_k]$ with $R^{\star} = \mathbf{K}[\mathbf{u}, y_1, \ldots, y_{k-1}]$. We know c_e is regular element in $R^{\star}$ since $\mathbb{T}^{\{k\}}$ is a regular set. It follows from $< \mathbb{T}^{\{k-1\}} \cup \mathbb{C}_{f_k} >=< 1 >$ that there exist $d_i \in R^{\star}[y_k]$ for $i = 0, \ldots, e$.

$$c_0 d_0 + c_1 d_1 + \ldots + c_{e-1} d_{e-1} + c_e d_e - 1 \in < \mathbb{T}^{\{k-1\}} > .$$

Now consider in $R[y_k]$ with $R = R^{\star} / < \mathbb{T}^{\{k-1\}}$, so $c_0 d_0 + c_1 d_1 + \ldots + c_{e-1} d_{e-1} + c_e d_e = 1$. For any $\beta \in R$ such that we have $c_e | \beta c_j$ for $j = 0, 1, \ldots, e-1$, it follows from $c_0 d_0 \beta + c_1 d_1 \beta + \ldots + c_{e-1} d_{e-1} \beta + c_e d_e \beta = \beta$ that $c_e | \beta$. So f_k is primitive in $R[y_k]$. Hence $\mathbb{T}$ is primitive. This completes the proof of the theorem. □

Acknowledgments. The author would like to thank Dr. Marc Moreno Maza and Dr. Yuzhen Xie for interesting discussions on the related subject, Prof. Xiao-Shan Gao for his help to correct the error in the definition of $\mathbb{U}_{\mathbb{T}}$ and the referees for their helpful comments and suggestions on an earlier version of this paper. This work is supported by a National Key Basic Research Project of China (2004CB318000).

References

1. Aubry, P., Lazard, D., Moreno Maza, M.: On the theories of triangular sets. J. Symb. Comput. 28, 105–124 (1999)
2. Boulier, F., Lemaire, F., Moreno Maza, M.: Well known theorems on triangular systems and the D5 principle. In: Proc. of Transgressive Computing 2006 (2006)
3. Chen, C., Golubitsky, O., Lemaire, F., Maza, M.M., Pan, W.: Comprehensive triangular decomposition. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2007. LNCS, vol. 4770, pp. 73–101. Springer, Heidelberg (2007)
4. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and algorithms. Springer, Heidelberg (1992)
5. Dahan, X., Maza, M.M., Schost, E., Wu, W., Xie, Y.: Lifting techniques for triangular decompositions. In: Proceedings ISSAC 2005, Beijing, China (2005)
6. Lemaire, F., Moreno Maza, M., Pan, W., Xie, Y.: When does $<\mathbb{T}> = \mathrm{sat}(\mathbb{T})$? In: Proceedings ISSAC 2008, pp. 207–214. ACM Press, New York (2008)
7. Gao, X.-S., Chou, S.-C.: Computations with parametric equations. In: Proceedings ISAAC 1991, pp. 122–127 (1991)
8. Gao, X.-S., Chou, S.-C.: On the dimension of an arbitray ascending chain. Chin. Sci. Bull. 38, 799–804 (1993)
9. Huang, F.-J.: Researches on Algorithms of Decomposition of Polynomial System. Ph.D. Thesis. Chengdu Institute of Computer Applications, China (2007)
10. Kalkbrener, M.: A generalized Euclidean algorithm for computing triangular representations of algebraic varieties. J. Symb. Comput. 15, 143–167 (1993)
11. Lazard, D.: A new method for solving algebraic systems of positive dimension. Discrete Appl. Math. 33, 147–160 (1991)
12. Li, Y.-B., Zhang, J.-Z., Yang, L.: Decomposing polynomial systems into strong regular sets. In: Cohen, A.M., Gao, X.-S., Takayama, N. (eds.) Proceedings ICMS 2002, pp. 361–371 (2002)
13. Li, Y.-B.: Applications of the theory of weakly nondegenerate conditions to zero decomposition for polynomial systems. J. Symb. Comput. 38, 815–832 (2004)
14. Li, Y.-B.: Some Properties of Triangular Sets and Improvement Upon Algorithm CharSer. In: Calmet, J., Ida, T., Wang, D. (eds.) AISC 2006. LNCS (LNAI), vol. 4120, pp. 82–93. Springer, Heidelberg (2006)
15. Wang, D.: Computing triangular systems and regular systems. J. Symb. Comput. 30, 221–236 (2000)
16. Wang, D.: Elimination Methods. Springer, New York (2001)
17. Wu, W.-T.: On the decision problem and the mechanization of theorem-proving in elementary geometry. Scientia Sinica 21, 159–172 (1978)
18. Wu, W.-T.: On zeros of algebraic equations — An application of Ritt principle. Kexue Tongbao 31, 1–5 (1986)
19. Yang, L., Zhang, J.-Z.: Search dependency between algebraic equations: An algorithm applied to automated reasoning. Technical Report ICTP/91/6, International Center For Theoretical Physics, International Atomic Energy Agency, Miramare, Trieste (1991)
20. Yang, L., Zhang, J.-Z., Hou, X.-R.: Non-Linear Equation Systems and Automated Theorem Proving. Shanghai Sci. & Tech. Education Publ. House, Shanghai (1996) (in Chinese)
21. Zhang, J.-Z., Yang, L., Hou, X.-R.: A note on Wu Wen-Tsün's nondegenerate condition. Technical Report ICTP/91/160, International Center For Theoretical Physics, International Atomic Energy Agency, Miramare, Trieste (1991); Also in Chinese Science Bulletin, 38(1), 86–87 (1993)

How to Correctly Prune Tropical Trees[⋆]

Jean-Vincent Loddo and Luca Saiu

Laboratoire d'Informatique de l'Université Paris Nord - UMR 7030
Université Paris 13 - CNRS
99, avenue Jean-Baptiste Clément - F-93430 Villetaneuse
{loddo,saiu}@lipn.univ-paris13.fr

Abstract. We present *tropical games*, a generalization of combinatorial *min-max* games based on tropical algebras. Our model breaks the traditional symmetry of rational zero-sum games where players have exactly opposed goals (*min* vs. *max*), is more widely applicable than *min-max* and also supports a form of pruning, despite it being less effective than α-β. Actually, *min-max* games may be seen as particular cases where both the game and its dual are tropical: when the dual of a tropical game is also tropical, the power of α-β is completely recovered. We formally develop the model and prove that the tropical pruning strategy is correct, then conclude by showing how the problem of approximated parsing can be modeled as a tropical game, profiting from pruning.

Keywords: combinatorial game, search, alpha-beta pruning, rational game, tropical algebra, tropical game, term, rewriting, logic, parsing.

1 Introduction

We are all familiar with games such as Chess or Checkers. Such games are purely *rational* as they do not involve any element of chance; they are also *zero-sum*, as the players' interests are dual: what one "wins", the other "loses" — which is the origin of the *min-max* evaluation mechanism. The two fundamental questions to be asked in a rational game are *"Who will win?"* and *"How much will she win?"*. Answering such questions involves *searching for a strategy* trough a (typically large) *game tree*. Some optimized search techniques were developed, which in the case of combinatorial two-player games include the *α-β pruning* technique [1,2]. α-β is not an approximated algorithm: its correctness relies on the mutual distributive properties of *min* and *max*. In this work we explore the implications of assuming only *one* player to be rational, breaking the symmetry of the traditional "double-sided" rationality. Quite unsurprisingly our *tropical α-pruning* depends on just *one* distributive property, a requirement satisfied by *tropical algebras* (Section 3).

Following the style introduced by [3] and [4], we will distinguish two aspects of two-player combinatorial games: a first one that we call *syntactic*, consisting

⋆ This work is partially financed by Marie Curie action n. 29849 Websicola and ANR-06-JCJC-0122.

in a description of the possible game positions and the valid moves leading from a position to another; the game syntax is the formal equivalent of the intuitive notion of the "game rules". By contrast the *semantic* aspect is concerned about the interpretation of the game according to the interests of the players, and ultimately about the answer to the two fundamental questions above. Our semantics will be based on tropical algebras, and as a consequence our technique is widely applicable, relying as it does only on their comparatively weak hypotheses.

We formally define tropical α-pruning and prove its soundness, as our main contribution (Section 4). A further contribution consists in our formalization of game evaluation and tropical (and α-β) cuts as a small-step semantics, so that proofs can reuse the results of term-rewriting theory.

Actually, our soundness result subsumes other works proving α-β's soundness over distributive lattices such as [4] and (later) [5], since distributive lattices are bi-tropical structures (Definition 8).

We conclude by proposing the algorithm design style *Choose-How-To-Divide and Conquer* meant for attacking even apparently unrelated search problems as tropical games; we develop approximated parsing as one such problem by showing how it profits from α-pruning (Section 5).

2 Combinatorial Game Syntax and Semantics

2.1 Syntax

We speak about "syntax", hinting at formal grammars, in that some initial game positions are given, together with some "rule" allowing to derive successive positions from those: in this way a game can be seen as the tree of all the possibilities of playing it — the tree of all the possible matches.

Definition 1 (Syntax). *A game syntax or arena is a triple $S = (\mathbb{P}, \lambda, succ)$, where:*

- *$\mathbb{P}$ is the set of all game positions.*
- *the turn function $\lambda : \mathbb{P} \rightarrow \{\mathcal{P}, \mathcal{O}\}$, says whose turn it is: $\mathcal{P}$ for "player" or $\mathcal{O}$ for "opponent".*
- *the successor function succ, taking a game position and returning all the positions reachable with valid moves from there; $succ : \mathbb{P} \rightarrow \mathbb{P}^*$.*

Given $S = (\mathbb{P}, \lambda, succ)$, we define:

- *the set of terminal positions $\mathbb{P}_T = \{\pi \in \mathbb{P} \mid succ(\pi) = \langle\rangle\}$.*
- *the dual arena $S^\perp = (\mathbb{P}, \lambda^\perp, succ)$, of course with $\lambda^\perp : \mathbb{P} \rightarrow \{\mathcal{P}, \mathcal{O}\}$, where for any $\pi \in \mathbb{P}$ we have $\lambda^\perp(\pi) \neq \lambda(\pi)$.*
- *the move relation is the binary version of the succ relation: for all $\pi, \pi' \in \mathbb{P}$, $move(\pi, \pi')$ iff $\pi' = \pi_i$ for some i, where $succ(\pi) = \langle \pi_1 ... \pi_n \rangle$.*

The arena is called alternate-turn iff $move(\pi, \pi')$ implies $\lambda(\pi) \neq \lambda(\pi')$.

If move is Nötherian we speak about Nötherian or finite arena.

Remark 1 (Alternate-turn arenas). It is possible to systematically make a game alternate-turn by "collapsing" all the *sequences of consecutive moves of the same player* into single moves.

One of the most important ideas in Game Theory is the *strategy*, containing a plan to win the game — a player saying to herself "if this happens I should do that, but if this other thing happens I should do that, and so on". It should be noticed that a strategy is only related to the syntactic part of a game, being independent, *per se*, from the game evaluation. In particular, a strategy may very well not be winning.

Definition 2 (Strategy). *Let $S = (\mathbb{P}, \lambda, succ)$ be an arena, and $\pi \in \mathbb{P}$ be a position. We define:*

- *the reachable positions from π as the right elements of the reflexive-transitive closure of the relation succ: $\pi{\downarrow} = succ^*(\pi)$;*
- *a global strategy σ, as a subset of the relation succ which is:*
  - *deterministic in $\mathcal{P}$ positions:*
    for all $\pi \in \mathbb{P}$ where $\lambda(\pi) = \mathcal{P}$, if $succ(\pi) = \langle \pi_1 ... \pi_n \rangle$ then $\sigma(\pi) = \langle \pi_i \rangle$, for some i such that $1 \leq i \leq n$.
  - *complete in $\mathcal{O}$ positions:*
    for all $\pi \in \mathbb{P}$ where $\lambda(\pi) = \mathcal{O}$, $\sigma(\pi) = succ(\pi)$.
- *a strategy for the initial position π is a global strategy for the restricted arena $S_\pi = (\pi{\downarrow}, \lambda|_{\pi\downarrow}, succ|_{\pi\downarrow})$, where we indicate with $f|_D$ the restriction of a function f to the set D.*

2.2 Semantics

Let us assume a finite game with syntax $S = (\mathbb{P}, \lambda, succ)$. Traditionally the two players have exactly opposed interests and we assume, by convention, that the player $\mathcal{P}$ will try to *minimize* the *payoff* of the final position while the opponent $\mathcal{O}$ will try to *maximize* it.

The ordinary way of *evaluating* such a finite game consists in labeling non-terminal nodes with the functions *min* and *max* (according to the turn), and terminal nodes with the *payoff* of the terminal position $p(\pi)$. Such values are then "propagated" back, applying the function at each node to its children's values. The final value at the root is called the *game value*: it says *who* wins and *how much*, supposing *both* players to be rational.

Hence, assuming $p : \mathbb{P}_T \to \mathbb{Z}$ in accord to the tradition, the game value $v_p : \mathbb{P} \to \mathbb{Z}$ could be simply defined as a function of the initial position:

$$v_p(\pi) = \begin{cases} p(\pi), & \pi \in \mathbb{P}_T \\ min_{i=1}^{n}\ v_p(\pi_i), & succ(\pi) = \langle \pi_1 ... \pi_n \rangle, \lambda(\pi) = \mathcal{P} \\ max_{i=1}^{n}\ v_p(\pi_i), & succ(\pi) = \langle \pi_1 ... \pi_n \rangle, \lambda(\pi) = \mathcal{O} \end{cases}$$

This classical definition has the obvious defect of only supporting the function *min* and *max*; often for resolving actual games the preferred structure is $\mathbb{Z}$, $\mathbb{Q}$ (possibly extended with $-\infty$ and $+\infty$), floating point numbers, or some sort of tuples containing such structures on which a topological order is defined. Hence, in order to be more general, let us define $\mathbb{U}$ to be any set closed over two associative binary operations $\oplus$ and $\odot$, where $\oplus$ will be associated to the player and $\odot$ to the opponent. Assuming $p : \mathbb{P} \to \mathbb{U}$, the definition above would become:

$$v_p(\pi) = \begin{cases} p(\pi), & \pi \in \mathbb{P}_T \\ \bigoplus_{i=1}^{n} v_p(\pi_i), & succ(\pi) = \langle \pi_1 ... \pi_n \rangle, \lambda(\pi) = \mathcal{P} \\ \bigodot_{i=1}^{n} v_p(\pi_i), & succ(\pi) = \langle \pi_1 ... \pi_n \rangle, \lambda(\pi) = \mathcal{O} \end{cases}$$

The extended v_p above is a step forward, but it still has the problem of only being well-defined on finite games. We solve this problem by abandoning the functional definition of v_p altogether, and giving a *small-step semantics* instead. Actually, this style will also be useful in Section 4.1 to prove the soundness of our pruning technique.

Remark 2 (Invariance under alternate-turn transformation). It is easy to see that the transformation hinted at in Remark 1 does not alter semantics, because of the two associative properties.

Definition 3 (Game). *A game is the triple $G=(S, \mathcal{A}, p)$, where $S=(\mathbb{P}, \lambda, succ)$ is the syntax, $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$ is an algebra with associative operations $\oplus$ and $\odot$, and where $p : \mathbb{P}_T \to \mathbb{U}$ is the payoff function.*

Sometimes we informally refer to syntactic or semantic properties as if they belonged to a game, for example by speaking about "Nötherian game" instead of "Game with Nötherian syntax".

Small-step Operational Semantics. In the following, we assume a game $G = (S, \mathcal{A}, p)$, where $S = (\mathbb{P}, \lambda, succ)$ and $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$.
The configurations of our system consist of (ground) *terms* of G, recursively defined as: $Ter(G) = \mathbb{P} \uplus \mathbb{U} \uplus (\{\sum, \prod\} \times Ter(G)^+)$:

– positions in $\mathbb{P}$ indicate game positions still to be expanded (if not terminal) and evaluated (otherwise).
– values in $\mathbb{U}$ denote the value, already fully computed, of some sub-terms.
– a complex term such as $\sum\langle t_1 ... t_n \rangle$ or $\prod\langle t_1 ... t_n \rangle$ indicates a position at some state of its evaluation; $\sum$ or $\prod$ holding the turn information, and $t_1 ... t_n$ representing the game subterms from that state on.

It is crucial not to mistake *terms of G*, which represent partially expanded game trees, for *game positions*, which in practice will also tend to be structured symbolic terms, but can be considered atomic at a high level: the rewrite rules shown in the following work on $Ter(G)$, not on $\mathbb{P}$.

Syntactic conventions. We use (possibly with subscripts or primes) π to indicate positions in $\mathbb{P}$, s and t for generic terms, v for values in $\mathbb{U}$, $\boldsymbol{t}$ and $\boldsymbol{z}$ for of terms in $Ter(G)$. Sequences are allowed to be empty, if not specified otherwise in a side condition. Just to make the notation more compact we will write $\sum \boldsymbol{t}$ instead of

$(\sum, \boldsymbol{t})$ and $\prod \boldsymbol{t}$ for $(\prod, \boldsymbol{t})$. We write Λ instead of either $\sum$ or $\prod$, just to avoid duplicating otherwise identical rules. Sequences are written with no commas, and parentheses or brackets are used to group when needed.

$$[\text{Payoff}]\ \frac{\pi \in \mathbb{P}_T \qquad p(\pi) = v}{\pi \to v}$$

$$[\mathcal{P}\text{-expand}]\ \frac{succ(\pi) = \boldsymbol{t} \qquad \lambda(\pi) = \mathcal{P}}{\pi \to \sum \boldsymbol{t}}\ \#\boldsymbol{t} \geq 1$$

$$[\mathcal{O}\text{-expand}]\ \frac{succ(\pi) = \boldsymbol{t} \qquad \lambda(\pi) = \mathcal{O}}{\pi \to \prod \boldsymbol{t}}\ \#\boldsymbol{t} \geq 1$$

$$[\mathcal{P}\text{-reduce}]\ \frac{}{\sum \boldsymbol{t}\langle v_1\ v_2\rangle \boldsymbol{z} \to \sum \boldsymbol{t}\langle v\rangle \boldsymbol{z}}\ v_1 \oplus v_2 = v$$

$$[\mathcal{O}\text{-reduce}]\ \frac{}{\prod \boldsymbol{t}\langle v_1\ v_2\rangle \boldsymbol{z} \to \prod \boldsymbol{t}\langle v\rangle \boldsymbol{z}}\ v_1 \odot v_2 = v$$

$$[\text{Return}]\ \frac{}{\Lambda\langle v\rangle \to v}$$

$$[\text{Context}]\ \frac{t \to t'}{C[t] \to_c C[t']}\ \text{for all contexts } C$$

[Payoff] simply replaces a terminal position with its value in $\mathbb{U}$, by means of the payoff function. [$\mathcal{P}$-expand] and [$\mathcal{O}$-expand] expand a position, generating its successors and keeping track of the turn, which will be important at reduction time. [$\mathcal{P}$-reduce] and [$\mathcal{O}$-reduce] combine two values into one, using $\oplus$ for the player and $\odot$ for the opponent. Notice that these two rules are sources of non-determinism. [Return] unwraps a completely evaluated term containing a single value. [Context] allows to use the other rules within nested terms (also introducing non-determinism).

Notice that keeping the relation $\to_c$ distinct from $\to$ allows us, when needed, to see our semantics as a *term rewriting system* (TRS) [6].

Proposition 1. *$\to_c$ is strongly confluent.*

Proof. For the purposes of this proof, we consider the small-step semantics as a pure term-rewriting system, expressed in a slightly sugared notation. The system does *not* need to be conditional (CTRS), since all the rule premises can in fact be seen as structural constraints on syntactic constructors. $\oplus$ and $\odot$ should also be read as syntactic constructors, with their associative properties written as rewrite rules. What is a variable in the rules becomes a (syntactic) variable in the TRS; however, we will not exploit the full power of the formal system: reductions will only be applied to ground terms[1].

[1] We do not need the full power of unification: from a programming point of view, *pattern matching* as used in ML or Haskell is enough for our purposes.

Our TRS is trivially *left-* and *right-linear*, as no variable occurs more than once in each side of a rule. By showing that our system is also *strongly closed*, strong confluence follows by Huet's Lemma 3.2 in [7]: "If $\mathscr{R}$ is a left- and right-linear strongly closed term rewriting system, $\to_{\mathscr{R}}$ is strongly confluent".

In order to show that the system is strongly-closed, we have to show that for every critical pair s, t there exist s', t' such that $s \to^* t' \leftarrow^{\equiv} t$ and $t \to^* s' \leftarrow^{\equiv} s$ (as in [7] and [6]), where $\leftarrow^{\equiv}$ is the reflexive closure of $\leftarrow$.

The left-hand side of [$\mathcal{P}$-reduce] is $\sum \boldsymbol{t}\langle v_1 \ v_2\rangle \boldsymbol{z}$. When this rule is used to generate a critical pair with any other rule, only a variable in $\boldsymbol{t}$ or in $\boldsymbol{z}$ can match, with the whole left-hand side of the other rule. The resulting critical pair s, t reaches confluence (to $s' = t'$) in one step because redexes are non-overlapping. The same holds for [$\mathcal{O}$-reduce].

The only rule pairs candidate for overlapping are [$\mathcal{P}$-reduce] with itself, and [$\mathcal{O}$-reduce] with itself; we only show the first one. The only interesting case of overlapping is the term family $\sum \boldsymbol{t}\langle v_1 \ v_2 \ v_3\rangle \boldsymbol{z}$, generating the critical pair s, t. Notice that $s' \to t'$ and vice-versa because of the associativity of $\oplus$:

$$\begin{tikzcd}
 & \sum \boldsymbol{t}\langle v_1 \ v_2 \ v_3\rangle \boldsymbol{z} \arrow[dl] \arrow[dr] & \\
s = \sum \boldsymbol{t}\langle (v_1 \oplus v_2) v_3\rangle \boldsymbol{z} \arrow[d] & & \sum \boldsymbol{t}\langle v_1 (v_2 \oplus v_3)\rangle \boldsymbol{z} = t \arrow[d] \\
s' = \sum \boldsymbol{t}\langle (v_1 \oplus v_2) \oplus v_3\rangle \boldsymbol{z} \arrow[rr, shift left] & & \sum \boldsymbol{t}\langle v_1 \oplus (v_2 \oplus v_3)\rangle \boldsymbol{z} = t' \arrow[ll, shift left]
\end{tikzcd}$$

□

Definition 4 (Game tree). *Let $\twoheadrightarrow$ be the sub-rewrite system of $\to_c$, made only by the rules [$\mathcal{P}$-expand], [$\mathcal{O}$-expand] and [Context]: given an initial position $\pi_0 \in \mathbb{P}$, the set of game tree prefixes from π_0 is the set $T_{\pi_0} = \{t \mid \pi_0 \twoheadrightarrow^* t\}$. The game tree, if it exists, is the tree $t_{\pi_0} \in T_{\pi_0}$ whose positions are all terminal.*

The game tree t_{π_0} is well-defined: when it exists it is unique. Actually, the TRS defining $\twoheadrightarrow^*$ is non-ambiguous (there is no *overlap* among any reduction rules) and left-linear: such a TRS is called *orthogonal*, and any orthogonal TRS is confluent [8].

Proposition 2. *$\to_c$ is normalizing for any Nötherian game.*

Proof. Let a game $G = (S, \mathcal{A}, p)$ where $S = (\mathbb{P}, \lambda, succ)$ and $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$ be given. We prove normalization by exhibiting a *reduction order* $<$ compatible with our rules [6].

Let us define a *weight* function $w : \mathbb{P} \to \mathbb{N}$ to be a particular instance of the higher-order function $v_{\bar{p}} : \mathbb{P} \to \mathbb{U}$, where $\bar{p}(\pi) = 2$ for any $\pi \in \mathbb{P}_T$ and $\bigoplus_{i=1}^n x_i = \bigodot_{i=1}^n x_i = 2 + \sum_{i=1}^n x_i$ for any $x \in \mathbb{N}^*$. Intuitively, w returns 2 times the number of nodes in the game tree for Nötherian games.

Let $f : Ter(G) \to \mathbb{N}$ be:

$$\begin{aligned}
&f(\pi) = w(\pi), && \pi \in \mathbb{P}\\
&f(v) = 1, && v \in \mathbb{U}\\
&f(\Lambda\langle t_1 ... t_n\rangle) = 1 + \textstyle\sum_{i=1}^n f(t_i) &&
\end{aligned}$$

In the formula above and in the rest of this proof $\sum$ represents the sum operation over $\mathbb{N}$. We define our order on terms by using the interpretation f on $>_{\mathbb{N}}$: by definition, let $t_0 > t_1$ iff $f(t_0) >_{\mathbb{N}} f(t_1)$. The order $>$ is trivially *stable*, as our terms do not contain variables. $>$ is also *monotonic* (f is increasing because $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is increasing), *strict* ($>_{\mathbb{N}}$ is strict) and *well-founded* ($>_{\mathbb{N}}$ is well-founded). Hence, $>$ is a reduction order.

In order to prove compatibility we show that for every rule $l \to r$ we have $l > r$, which by definition is equivalent to $f(l) >_{\mathbb{N}} f(r)$. All equalities follow from definitions or trivial algebraic manipulations:

– [Payoff]: $f(\pi) = w(\pi) = \bar{p}(\pi) = 2 >_{\mathbb{N}} 1 = f(v)$.
– [$\mathcal{P}$-expand], [$\mathcal{O}$-expand]: $f(\pi) = w(\pi) = 2+\sum_{i=1}^{n} w(\pi_i) >_{\mathbb{N}} 1+\sum_{i=1}^{n} w(\pi_i) = 1 + \sum_{i=1}^{n} f(t_i) = f(\Lambda\ \boldsymbol{t})$.
– [$\mathcal{P}$-reduce], [$\mathcal{O}$-reduce]: $f(\Lambda\ \boldsymbol{t}\langle v_1 v_2\rangle \boldsymbol{z}) = \sum_{i=1}^{\#\boldsymbol{t}} f(t_i) + f(v_1) + f(v_2) + \sum_{i=1}^{\#\boldsymbol{z}} f(z_i) = \sum_{i=1}^{\#\boldsymbol{t}} f(t_i) + 1 + 1 + \sum_{i=1}^{\#\boldsymbol{z}} f(z_i) >_{\mathbb{N}} \sum_{i=1}^{\#\boldsymbol{t}} f(t_i) + 1 + \sum_{i=1}^{\#\boldsymbol{z}} f(z_i) = f(\Lambda\ \boldsymbol{t}\langle v\rangle \boldsymbol{z})$.
– [Return]: $f(\Lambda\langle v\rangle) = 1 + 1 >_{\mathbb{N}} 1 = f(v)$. □

Intuitively, if a term converges then its sub-terms also converge; said otherwise if a term converges in a context, then it must also converge in the trivial (empty) context. This is true because of the *non-erasing* nature of our system, different from, for example, the λ-calculus having actual *reduction steps* [8]. More formally:

Lemma 1 (Sub-term normalization). *Given a game $G = (S, \mathcal{A}, p)$ where $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$, for any term $t \in Ter(G)$ and any context C, if there exists $v \in \mathbb{U}$ such that $C[t] \to_c^* v$ then there exists $v' \in \mathbb{U}$ such that $t \to_c^* v'$.*

Proof. By induction over the derivation length n of $C[t] \to_c^* v$. We look at the possible shape of the premise of the [Context] rule, $s \to s'$.

– Base case, $n = 1$: $C[t] \to_c v$. The only applicable rules are [Payoff] and [Return]: in the case of [Payoff], $C[t] = t$; in the case of [Return], $t = v$. In either case, $t \to_c^* v = v'$.
– Recursive case $n \Rightarrow n + 1$: $t_0 = C[t] \to_c t_1 \to_c^* v$. The inductive hypothesis is that for any term t' and context C' if $C'[t'] \to_c^* v$ in n or fewer steps, then $t' \to_c^* v'$. Three cases:
  • s and t are disjoint sub-terms within C. Since the system is non-erasing t has not been erased, i.e. $t_1 = C'[t]$; for inductive hypothesis $t \to_c^* v'$.
  • s contains t. $s \to s'$ may have as its premise [Return], in which case $s = \Lambda\langle v\rangle$ and $t = v$. Otherwise the premise may be [$\mathcal{P}$-Reduce] or [$\mathcal{O}$-Reduce]: either t is one of the values, or it matches one of the variables, in which case there exists a context C' such that $t_1 = C'[t]$; then the inductive hypothesis applies.
  • t contains s. $t = C'[s]$, hence by definition of $\to_c$ we have that t can turn into $C'[s'] = t'$. There exists a context C'' where $C[s] = C''[C'[s]]$, hence $t_1 = C''[C'[s']]$. By induction hypothesis $t' = C'[s'] \to_c^* v'$. □

Normalization and confluence justify our re-definition of the game value v_p as the transitive closure of the transition relation $\to_c$:

Definition 5 (Game Value). *Let a game, an initial position π and a value v be given; we say that the game value from π is v (and we write $v_p(\pi) = v$) if and only if $\pi \to_c^* v$.*

3 α-β Pruning

The α-β *algorithm* [1,2] is a method for computing the exact value of a *min-max* combinatorial game without exhaustively visiting *all* game positions.

The α-β algorithm is traditionally presented as a recursive function written in imperative style (see Figure 1): the function `alpha_beta` analyzes a game position $\pi \in \mathbb{P}$ with two additional parameters, α and β, each one denoting a sort of *threshold* not to be overstepped during the incremental computation of the value of $\mathbb{P}$. Whenever the threshold is past the evaluation of an entire subtree is aborted, as it can be proven that it will not contribute to the result.

The correctness of α-β relies on the algebraic properties of the *min* and *max* functions, notably their *mutual* distributive laws — something we can *not* count on under our weaker hypotheses on $\oplus$ and $\odot$ [4,5,3].

```
function alpha_beta($\pi : \mathbb{P}$; $\alpha, \beta : \mathbb{Z}$) : $\mathbb{Z}$
  if $\pi \in \mathbb{P}_T$ then
    return $p(\pi)$
  $\pi_1 ... \pi_n := succ(\pi)$  # $n \geq 1$
  if $\lambda(\pi) = \mathcal{P}$ then
    $v := \alpha$
    for $i$ from 1 to $n$
        and while $\beta <_{\mathbb{Z}} v$ do
      $v := min\{v, \text{alpha\_beta}(\pi_i, v, \beta)\}$
  else # $\lambda(\pi) = \mathcal{O}$
    $v := \beta$
    for $i$ from 1 to $n$
        and while $v <_{\mathbb{Z}} \alpha$ do
      $v := max\{v, \text{alpha\_beta}(\pi_i, \alpha, v)\}$
  return $v$
```

```
function tropical($\pi : \mathbb{P}$; $\alpha : \mathbb{U}$) : $\mathbb{U}$
  if $\pi \in \mathbb{P}_T$ then
    return $p(\pi)$
  $\pi_1 ... \pi_n := succ(\pi)$  # $n \geq 1$
  if $\lambda(\pi) = \mathcal{P}$ then
    $v := \alpha$
    for $i$ from 1 to $n$ do
      # do not prune at $\mathcal{P}$'s level
      $v := v \oplus \text{tropical}(\pi_i, v)$
  else # $\lambda(\pi) = \mathcal{O}$
    $v := \text{tropical}(\pi_1, \alpha)$  # No $\mathbf{1}_{\mathbb{U}}$
    for $i$ from 2 to $n$
        and while $\alpha \oplus v \neq \alpha$ do
      $v := v \odot \text{tropical}(\pi_i, \alpha)$
  return $v$
```

Fig. 1. Pruning algorithms: traditional α-β pruning vs. tropical α-pruning. Notice that the tropical version has the first iteration of the second loop unrolled, in order not to depend on the existence of a neutral element for $\odot$.

Going back to our game semantics presentation we can model the α-β's behavior by adding four more rules — two per player:

$$[\mathcal{P}\text{-will}]\ \frac{}{\sum\langle\alpha\ [\prod\langle\beta\ (\sum \boldsymbol{t}_1)\rangle\ \boldsymbol{t}_2]\rangle\ \boldsymbol{t}_3\ \rightarrow\ \sum\langle\alpha\ [\prod\langle\beta\ (\sum\langle\alpha\rangle\boldsymbol{t}_1)\rangle\ \boldsymbol{t}_2]\rangle\ \boldsymbol{t}_3}$$

$$[\mathcal{O}\text{-will}]\ \frac{}{\prod\langle\beta\ [\sum\langle\alpha\ (\prod \boldsymbol{t}_1)\rangle\ \boldsymbol{t}_2]\rangle\ \boldsymbol{t}_3\ \rightarrow\ \prod\langle\beta\ [\sum\langle\alpha\ (\prod\langle\beta\rangle\boldsymbol{t}_1)\rangle\ \boldsymbol{t}_2]\rangle\ \boldsymbol{t}_3}$$

$$[\mathcal{P}\text{-cut}]\ \frac{\alpha \oplus \beta = \alpha}{\sum\langle\alpha\ (\prod\langle\beta\rangle\boldsymbol{t}_1)\rangle\ \boldsymbol{t}_2 \rightarrow \sum\langle\alpha\rangle\boldsymbol{t}_2}\ [\mathcal{O}\text{-cut}]\ \frac{\beta \odot \alpha = \beta}{\prod\langle\beta\ (\sum\langle\alpha\rangle\boldsymbol{t}_1)\rangle\ \boldsymbol{t}_2 \rightarrow \prod\langle\beta\rangle\boldsymbol{t}_2}$$

The initialization of v at line 6 should be read as a first "virtual" move of the player, whose evaluation is the value α inherited from an ancestor (the grandparent in an alternate-turn game). This explains the rationale of [$\mathcal{P}$-will][2]: whenever subtrees are nested with turns $\mathcal{P}$-$\mathcal{O}$-$\mathcal{P}$, a grandparent may cross two levels and "give" its grandchild its current accumulator as an initialization value. Of course line 10 is the dual version for the opponent and [$\mathcal{O}$-will].

[$\mathcal{P}$-cut] and [$\mathcal{O}$-cut] are a simple reformulation of the *cut conditions* at lines 7 and 11, where the explicit order $<_{\mathbb{Z}}$ disappears[3] from the condition, now expressed as an equality constraint in the rule premise: $\alpha \oplus \beta = \alpha$ represents the fact that the player would prefer α over β. Dually, $\beta \odot \alpha = \beta$ means that the opponent would prefer β over α.

Remark 3 (Non-alternate turn games). Notice that the cut rules can just fire in alternate-turn contexts: this choice simplifies our exposition, but does not limit generality: see Remarks 1 and 2.

The presence of two exactly symmetrical behaviors is quite evident in either presentation; yet what we are interested in showing now is the fact that such duality is quite incidental: it occurs in a natural way in actual two-player games, yet many more search problems lend themselves to be modeled as games despite lacking an intrinsic symmetry.

We can see α-β as the union of two separate techniques applied at the same time, breaking the algebraic symmetry of the player/opponent operations: in the following we are going to eliminate the rules [$\mathcal{O}$-will] and [$\mathcal{O}$-cut], or equivalently to turn `alpha_beta` into `tropical` (see Figure 1), exploiting the weaker properties of *tropical algebras* which only allow *one* threshold α.

4 Tropical Games

As we are dealing with a relatively young research topic, it is not surprising that the formalization of tropical algebras has not yet crystallized into a standard form. And since several details differ among the various presentations, we have to provide our own definition:

Definition 6 (Tropical Algebra). *An algebra $(\mathbb{U}, \oplus, \odot)$ is called a tropical algebra if it satisfies the following properties for any a, b and c in $\mathbb{U}$:*

(i) Associativity of $\oplus$: $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
(ii) Associativity of $\odot$: $a \odot (b \odot c) = (a \odot b) \odot c$
(iii) Left-distributivity of $\odot$ with respect to $\oplus$: $a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$
(iv) Right-distributivity of $\odot$ with respect to $\oplus$: $(a \oplus b) \odot c = (a \odot c) \oplus (b \odot c)$

[2] "Will" should be interpreted as "bequeath", in the sense of leaving something as inheritance to a descendent.

[3] This is customary with lattices, when an order is derived from a *least-upper-bound* or *greatest-lower-bound* operation.

Some particular choices of $\mathbb{U}$, $\oplus$ and $\odot$ are widely used: the *min-plus algebra* is obtained by defining $\mathbb{U} \triangleq \mathbb{R} \cup \{+\infty\}$, $a \oplus b \triangleq min\{a, b\}$ and, a little counter-intuitively[4], $a \odot b \triangleq a + b$.

Since $\mathbb{U}$ and $\odot$ can also be usefully instantiated in other ways, we will not simply adopt a min-plus algebra; anyway *in practice* we will also choose $\oplus$ to be a minimum on $\mathbb{U}$, which *in practice* will have a total order. This seems to be the only reasonable choice for the applications[5] and helps to understand the idea, yet nothing in the theory depends on the existence of the order. Again, *in practice*, $\oplus$ will return one of its parameters, so if needed we will always be able to trivially define a total order as $x \leq y$ iff $x \oplus y = x$, for any x and y in $\mathbb{U}$. $\oplus$ and $\odot$ will also tend to be commutative *in practice*, making one of the two distributive properties trivial.

We will not make any of the supplementary hypotheses above; on the other hand, we will require the following *rationality hypothesis*[6]:

Definition 7 (Rationality). *Let* $(\mathbb{U}, \oplus, \odot)$ *be a tropical algebra such that* $\mathbf{0} \in \mathbb{U}$ *is a neutral element for* $\oplus$ *and* $\mathbf{1} \in \mathbb{U}$ *is a neutral element for* $\odot$[7]*. We call the algebra rational if, for any* $x, y, z \in \mathbb{U}$ *we have* $x \oplus (y \odot x \odot z) = x$*.*

Intuitively, the opponent accumulates costs with $\odot$, "worsening" the game value for the player: the player will always choose just x over x "worsened" by something else. Notice that the notion of rationality for two-player games in Game Theory also includes the dual condition $x \odot (y \oplus x \oplus z) = x$; such condition does *not* hold in general for tropical games.

Definition 8 (Tropical Game, Tropical Trees). *A tropical game* $G = (S, \mathcal{A}, p)$ *is simply a game based on a* rational *tropical algebra* $\mathcal{A}$*. We call tropical trees all the game trees of a tropical game, and tropical pruning the* α*-pruning of a tropical tree. A bi-tropical game is a tropical game whose dual* $G^{\perp} = (S^{\perp}, \mathcal{A}^{\perp}, p)$ *is also tropical, where* $\mathcal{A}^{\perp} = (\mathbb{U}, \odot, \oplus)$ *if* $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$*.*

4.1 Soundness of Tropical Pruning

Proposition 3 (Insertion property). *Let* $(\mathbb{U}, \oplus, \odot)$ *be a rational tropical algebra. Then for any* $x, y, \alpha, \beta \in \mathbb{U}$ *we have* $\alpha \oplus (\beta \odot x \odot y) = \alpha \oplus (\beta \odot (\alpha \oplus x) \odot y)$*.*

Proof (Using associativity implicitly). $\alpha \oplus (\beta \odot (\alpha \oplus x) \odot y) = \{\text{right-distributivity}\}$ $\alpha \oplus (\beta \odot ((\alpha \odot y) \oplus (x \odot y))) = \{\text{left-distributivity}\}$ $(\alpha \oplus (\beta \odot \alpha \odot y) \oplus (\beta \odot x \odot y) = \{\text{rationality}\}$ $\alpha \oplus (\beta \odot x \odot y)$ $\square$

[4] The particular symbols used for indicating $\oplus$ and $\odot$ are justified by the analogy with $+$ and $\cdot$ in how the distributive law works.

[5] Logic programming is an example of an interesting problem lending itself to be interpreted as a combinatorial game on a universe with no total order [3,4]. Anyway the underlying game is a symmetrical *inf-sup* rather than simply tropical.

[6] In lattice theory, the rationality hypothesis is one of the *absorption identities*.

[7] The existence of neutral elements is not strictly necessary, but it simplifies many statements and proofs; without them several results should be given in both "left" and "right" forms.

The insertion property is the semantic counterpart of the rule [$\mathcal{P}$-will]: it explains why we can "transfer" α down in the tree (or more operationally, why we can "start" from the same α when choosing with $\oplus$ two plies below), without affecting the game value.

Definition 9 ($\mathcal{P}$-irrelevance). *Let $(\mathbb{U}, \oplus, \odot)$ be a rational tropical algebra, and let $\alpha, \beta \in \mathbb{U}$. Then we call $x \in \mathbb{U}$ $\mathcal{P}$-irrelevant with respect to α and β if $\alpha \oplus (\beta \odot x) = \alpha$.*

Intuitively, as the value of an opponent-level tree, x can't affect the value of the game because the player will not give the opponent the opportunity to be in that situation: in other word, the current optimal move for the player doesn't change because of x.

Lemma 2 ($\mathcal{P}$-irrelevance). *Let $(\mathbb{U}, \oplus, \odot)$ be a rational tropical algebra, and $\alpha, \beta \in \mathbb{U}$. If $\alpha \oplus \beta = \alpha$ then any $x \in \mathbb{U}$ is $\mathcal{P}$-irrelevant with respect to α and β.*

Proof. $\alpha \oplus (\beta \odot x) = \{\text{hypothesis}\}\ (\alpha \oplus \beta) \oplus (\beta \odot x) = \{\text{associativity}\}\ \alpha \oplus (\beta \oplus (\beta \odot x)) = \{\text{rationality}\}\ \alpha \oplus \beta = \{\text{hypothesis}\}\ \alpha$ □

Definition 10 (Simulation). *Given a tropical game, we say that a term t' simulates a term t, and we write $t \le t'$, if $t \to_c^* v \Rightarrow t' \to_c^* v$.*

Lemma 3 (Tropical $\mathcal{P}$-will simulation). *Given a tropical game $G = (S, \mathcal{A}, p)$ where $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$, for any term sequence $\alpha, \beta \in \mathbb{U}$, $\boldsymbol{t_0}, \boldsymbol{t_1}, \boldsymbol{t_2} \in Ter(G)^*$*

$$\sum\langle\alpha\ [\prod\langle\beta\ (\sum \boldsymbol{t}_0)\rangle\ \boldsymbol{t}_1]\rangle\ \boldsymbol{t}_2\ \le\ \sum\langle\alpha\ [\prod\langle\beta\ (\sum\langle\alpha\rangle\ \boldsymbol{t}_0)\rangle\ \boldsymbol{t}_1]\rangle\ \boldsymbol{t}_2$$

Proof. By the Sub-term normalization Lemma, if t converges there will exist some value sequences $\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathbb{U}$ such that $\boldsymbol{t}_0 \to^* \boldsymbol{v}_0$, $\boldsymbol{t}_1 \to^* \boldsymbol{v}_1$, $\boldsymbol{t}_2 \to^* \boldsymbol{v}_2$; let us call v_0 the result of $\bigoplus \boldsymbol{v}_0$, v_1 the result of $\bigodot \boldsymbol{v}_1$ and v_2 the result of $\bigoplus \boldsymbol{v}_2$. Then,

$$\begin{tikzcd}
\sum\langle\alpha\ [\prod\langle\beta\ (\sum \boldsymbol{t}_0)\rangle\ \boldsymbol{t}_1]\rangle\ \boldsymbol{t}_2 \arrow[d, "*"] & & \sum\langle\alpha\ [\prod\langle\beta\ (\sum\langle\alpha\rangle\ \boldsymbol{t}_0)\rangle\ \boldsymbol{t}_1]\rangle\ \boldsymbol{t}_2 \arrow[d, "*"] \\
\sum\langle\alpha\ [\prod\langle\beta\ v_0\ v_1\rangle]\ v_2\rangle \arrow[d, "*"] & \{\text{Insertion}\} & \sum\langle\alpha\ [\prod\langle\beta\ (\alpha \oplus v_0)\ v_1\rangle]\ v_2\rangle \arrow[d, "*"] \\
\alpha \oplus [\beta \odot v_0 \odot v_1] \oplus v_2 & = & \alpha \oplus [\beta \odot (\alpha \oplus v_0) \odot v_1] \oplus v_2
\end{tikzcd}$$

In the reductions above we implicitly assume that some sequences are non-empty; the proof trivially generalizes to empty $\boldsymbol{t_1}$ and $\boldsymbol{t_2}$ by using neutral elements. □

Lemma 4 (Tropical cut simulation). *Given a tropical game $G = (S, \mathcal{A}, p)$ where $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$, for any term sequence $\alpha, \beta \in \mathbb{U}$, $\boldsymbol{t_0}, \boldsymbol{t_1} \in Ter(G)^*$ we have that if $\alpha \oplus \beta = \alpha$, then $\sum\langle\alpha\ (\prod\langle\beta\rangle\ \boldsymbol{t}_0)\rangle\ \boldsymbol{t}_1\ \le\ \sum\langle\alpha\rangle\ \boldsymbol{t}_1$.*

Proof. Just like Lemma 3, with $\mathcal{P}$-irrelevance at the end. □

Theorem 1 (Tropical rule soundness). *Adding the rules [$\mathcal{P}$-will] and [$\mathcal{P}$-cut] "does not alter semantics", i.e. if a term t converges to a value v in a system without the two new rules, it is guaranteed to have a reduction sequence converging to v also in the extended system.* □

5 Choose-How-To-Divide and Conquer

According to the classical Divide and Conquer technique a problem can be divided into subproblems, each of which will be solved recursively until a minimal-size instance is found; sub-solutions will then be recomposed.

In the traditional Divide and Conquer style, each division choice is final: it is considered taken once and for all, and cannot be undone. By contrast we present an alternative model based on tropical games. In the *Choose-How-To-Divide and Conquer* style we work with non-deterministic choices in a solution space, using a quality criterion to be optimized and some way of "combining" sub-solutions.

Of course many nondeterministic algorithms can be expressed this way: the challenge is finding a suitable mapping to the tropical game concepts, in term of both syntax and semantics (with the required properties). The problem must have both a suitable *syntactic* structure, and a *semantic* structure with the required properties.

The action of *choosing a division* corresponds to a player node where the $\oplus$ function (typically a minimization) returns the "best" option; the points where *sub-solutions have their cost accumulated* (often something similar to a sum, intuitively "opposed" to $\oplus$) become opponent nodes where $\odot$ combines the values of a subtree sequence into a single result.

Tropical trees have the desirable property of supporting α-pruning, with the potential of significantly cutting down the search space. The more [$\mathcal{P}$-will] and [$\mathcal{P}$-cut] can fire, the more pruning is profitable: hence the problem should be represented as a tropical game having *alternate turns* and *branching factor greater than* 2 *for* $\mathcal{O}$ at least "often enough".

Search problems abound in Artificial Intelligence, and in particular we suspect that more symbolic computation problems than one might expect can be modeled this way. We now proceed to show an unusual application of *Choose-How-To-Divide and Conquer.*

5.1 Parsing as a Tropical Game

Let $\mathscr{G}$ be a given context-free grammar, defined over an alphabet of terminals $A \ni a$ and nonterminals $N \ni X$. For simplicity[8] let it have no ϵ-production, nor any productions with two consecutive nonterminals or a single nonterminal alone in the right-hand side. Right-hand sides will hence be of the form $[a_1]X_1a_2X_2...a_nX_n[a_{n+1}]$, with $n \geq 0$ and at least one a_i. Given a string of terminals $s \in A^+$ our problem is finding the "best" parse tree of s in $\mathscr{G}$; when s contains some errors our "best" solution just ends up being *the least wrong*, according to some metric; just to keep things simple in this example out metric to minimize will be the total size of the substrings which cannot be matched, in terminals. Sometimes we may wish to have the set of *all* best parses, instead of being content with just one optimal solution.

[8] Such restrictions can be lifted at the cost of some complexity, but supporting a larger class of grammars would be quite inessential for our demonstrative purposes.

Syntax. The set of game positions is defined as $\mathbb{P} = (A^* \times N) \uplus (A^* \times N)^*$, and the turn function is $\lambda(s, X) = \mathcal{P}$, $\lambda((s_1, X_1)...(s_k, X_k)) = \mathcal{O}$. These definitions become easy to understand once the successor function *succ* is examined.

A player position has the form $\pi_{\mathcal{P}} = (s, X)$, since the player has to parse a string s with a nonterminal X. It has to choose a production $X ::= [a_1]X_1 a_2 X_2 ... a_n X_n [a_{n+1}]$, and match the terminals a_i with the terminals in s, in the right order. Each possible match of *all* terminals, for each production of X, is a valid player move generating *strictly smaller* subproblems for the opponent: the nonterminals X_i "in between" the matched terminals will have to be matched to substrings of s in the opponent position $\pi_{\mathcal{O}} = (s_1, X_1)...(s_1, X_n)$, for some $n \geq 0$. If no match exists with any production then $\pi_{\mathcal{P}}$ is terminal.

In an opponent position $\pi_{\mathcal{O}} = (s_1, X_1)...(s_1, X_n)$ the opponent has always exactly n moves: the opponent will give the player each pair (s_i, X_i) to solve "one at the time". For this reason the successor of an opponent position is equal to the position itself: it is the sequence of the elements of $\pi_{\mathcal{O}}$, itself a sequence. An opponent position $\pi_{\mathcal{O}}$ is terminal when it is empty.

Figure 2 contains a practical example.

$$
\begin{aligned}
E &::= \mathbf{n} \\
E &::= \mathbf{v} \\
E &::= \mathbf{(}\ E\ \mathbf{)} \\
E &::= \mathbf{let}\ \mathbf{v} = E\ \mathbf{in}\ E \\
E &::= \mathbf{if}\ E\ \mathbf{then}\ E\ \mathbf{else}\ E \\
E &::= E = E \\
E &::= E + E \\
E &::= E * E
\end{aligned}
$$

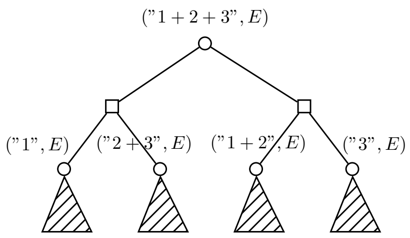


Fig. 2. We use the simple grammar $\mathscr{G}$ above, with an intentionally high level of ambiguity, to parse the string `"1 + 2 + 3"` with E as the start symbol. Circles represent $\sum$ nodes, squares are for $\prod$.

Semantics. We use a *min-plus* algebra for $\mathcal{A} = (\mathbb{U}, \oplus, \odot)$: we simply define $\mathbb{U} \triangleq \mathbb{N}$; we take $\oplus \triangleq min$, since we want as few errors as possible; and finally $\odot \triangleq +$: the number of total errors in the parse tree is equal to the sum of the number of errors in all subtrees.

The payoff $p(\pi)$ is defined as the length in characters of the input string for player positions (notice that the payoff is only defined on *terminal* positions, so such a length is actually the number of unmatched characters), and zero for opponent positions (if $\pi_{\mathcal{O}} = \langle\rangle$ then there are no errors to accumulate: at the level above, the player matched *the whole* string): $p(s, X) \triangleq \#s$, $p(\langle\rangle) \triangleq 0$.

Experiments. We implemented a prototype system[9] in ML supporting the grammar of Figure 2, which can be configured to do a simple exhaustive search

[9] The prototype is freely available under the GNU GPL license at the address `http://www-lipn.univ-paris13.fr/~loddo/aisc-2010`

or perform tropical α-pruning. The prototype supports two policies: *first-minimal (henceforth FM)* searches for only one optimal strategy at $\mathcal{P}$'s levels, and *all-minimals (henceforth AM)* generates a sequence of strategies with *non-increasing* cost.

Just as illustrative examples, we proceed to show our system behavior on some input strings belonging to different categories.

Non-ambiguous input: the input string `"let x = 42 in x + if 84=42 then 55 else 77"` is parsable in a unique way, so the FM policy is clearly the right choice. Compared to an exhaustive search the α-pruning FM version avoids 98% of the recursive calls (460 vs 28473) and its completion time is 4%. By setting the policy to AM the number of recursive call grows a little, from 460 to 671 (still avoiding 97% of the calls).

Ambiguous input: with the input string `"let x = 84 = 42 = 21 in 1 + 2 * 3"`, which is parsable in several ways, the the α-pruning FM version avoids 99% of the recursive calls (260 vs 61980), and the run time is 1% of the exhaustive-search version time. The α-pruning AM version still avoids 96% of the recursive calls (2148 vs 61980), and its run time is 3%.

"Wrong" input: with the input string `"if if if true then true else false then 10 else (1+(2+)+3)"`, containing errors, the α-pruning FM version avoids 98% of the recursive calls (9640 vs 494344) and its run time is 3%, while the AM version avoids 97% of the recursive calls (13820 vs 494344); the AM version's run time is reduced to 3%. The best strategy has value 6, corresponding to the size of the substring `"if true"` (blanks are not counted) added to the size (0) of the empty substring delimited by the tokens `"+"` and `")"`. The α-pruning algorithm has *localized* errors, guessing that the user should fix her string by replacing `"if true"` with something correct and writing something correct between `"+"` and `")"` — having the size of the unmatched substrings as the payoff function yields this "smallest-incorrect-string" heuristic. Of course other more elaborate criteria are also possible, such as "minimum number of errors".

Memoization: on a completely orthogonal axis, the implementation may be configured to perform *memoization*: when memoization is turned on all the already solved positions are cached, so that they are not computed more than once. We have compared a memoizing version of our tropical-α-pruning parser with a memoizing version performing exhaustive search. In the first case above, the string `"let x = 42 in x + if 84=42 then 55 else 77"` is now parsed with 131 calls instead of 460, again saving 98% of the calls (131 vs 7295) and cutting the run time to 1%. `"let x = 84 = 42 = 21 in 1 + 2 * 3"` is now parsed with 72 calls instead of 260, avoiding 99% of the calls (72 vs 14443) and reducing the run time to 7%. The string `"if if if true then true else false then 10 else (1+(2+)+3)"` is parsed with 1206 calls instead of 9640, avoiding 96% of calls (1206 vs 36575) and cutting the completion time to 10%.

At least in our small test cases, tropical α-pruning and memoization work well together: enabling either one does not significantly lessen the efficacy of the other.

6 Conclusions and Future Work

We have introduced and formally proved correct *tropical α-pruning*, a variant of α-β-pruning applicable to the *tropical games* underlying *Choose-How-To-Divide and Conquer* problems. As a practical example of the technique we have shown how the problem of approximated parsing and error localization can be modeled as a game, and how our pruning technique can dramatically improve its efficiency; yet an asymptotic measure of the visited node reduction would be a worthy development.

We suspect that many more problems can be formalized as tropical games, and the problem of parsing itself can also definitely be attacked in a more general way, lifting our restrictions on the grammar; tropical parsing might prove to be particularly suitable for natural language problems, with their inherent ambiguity.

The correctness and efficiency of *parallel* tropical α-pruning implementations would be particularly interesting to study.

Acknowledgments. Christophe Fouqueré first recognized tropical algebras in the properties required by our formalization.

References

1. Hart, T.P., Edwards, D.J.: The tree prune (TP) algorithm. In: Artificial Intelligence Project Memo 30. Massachusetts Institute of Technology, Cambridge (1961)
2. Knuth, D.E., Moore, R.W.: An analysis of alpha-beta pruning. Artificial Intelligence 6, 293–326 (1975)
3. Loddo, J.V.: Généralisation des Jeux Combinatoires et Applications aux Langages Logiques. PhD thesis, Université Paris VII (2002)
4. Loddo, J.V., Cosmo, R.D.: Playing logic programs with the alpha-beta algorithm. In: Parigot, M., Voronkov, A. (eds.) LPAR 2000. LNCS (LNAI), vol. 1955, pp. 207–224. Springer, Heidelberg (2000)
5. Ginsberg, M.L., Jaffray, A.: Alpha-beta pruning under partial orders. In: Games of No Chance II (2001)
6. Klop, J.W., de Vrijer, R.: First-order term rewriting systems. In: Terese (ed.) Term Rewriting Systems, pp. 24–59. Cambridge University Press, Cambridge (2003)
7. Huet, G.: Confluent reductions: Abstract properties and applications to term rewriting systems. J. ACM 27(4), 797–821 (1980)
8. Klop, J.W., Oostrom, V.V., de Vrijer, R.: Orthogonality. In: Terese (ed.) Term Rewriting Systems, pp. 88–148. Cambridge University Press, Cambridge (2003)

From Matrix Interpretations over the Rationals to Matrix Interpretations over the Naturals

Salvador Lucas

ELP Group, DSIC, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain

Abstract. Matrix interpretations generalize linear polynomial interpretations and have been proved useful in the implementation of tools for automatically proving termination of Term Rewriting Systems. In view of the successful use of rational coefficients in polynomial interpretations, we have recently generalized traditional matrix interpretations (using *natural* numbers in the matrix entries) to incorporate *real* numbers. However, existing results which formally prove that polynomials over the reals are *more powerful* than polynomials over the naturals for proving termination of rewrite systems *failed* to be extended to matrix interpretations. In this paper we get deeper into this problem. We show that, under some conditions, it is possible to transform a matrix interpretation over the rationals satisfying a set of symbolic constraints into a matrix interpretation over the naturals (using *bigger* matrices) which still satisfies the constraints.

Keywords: Matrix and Polynomial Interpretations, Program Analysis, Termination.

1 Introduction

Constraint solving is an essential technique for the implementation of automatic verification systems. Many verification problems can be expressed as sets of symbolic constraints which have to be tested for satisfaction or even solved to give some explicit solution certifying their satisfaction. For instance, termination problems are often expressed as conjunctions of *weak* or *strict symbolic constraints* like $e \succeq e'$ or $e \succ e'$ between expressions e and e' coming from (parts of) the programs [6]. Automatic termination tools have to *check* these constraints and eventually provide an appropriate certificate. A standard approach is using algebraic interpretations which translate the *symbolic* constraints into some kind of *arithmetic* constraints.

Example 1. Consider the following Term Rewriting System (TRS) $\mathcal{R}$ [1,2]:

$$f(f(X)) \to f(g(f(X))) \qquad (1)$$
$$f(g(f(X))) \to X \qquad (2)$$

A proof of termination with *dependency pairs* can be easily obtained as follows [3]: Consider the following rules (called *dependency pairs*) associated to $\mathcal{R}$:

$$F(f(X)) \to F(g(f(X))) \tag{3}$$

$$F(f(X)) \to F(X) \tag{4}$$

The following polynomial intepretation with *rational* coefficients

$$[f](x) = 2x + 2 \qquad [g](x) = \tfrac{1}{2}x + \tfrac{1}{2} \qquad [F](x) = x$$

can be used to prove termination of $\mathcal{R}$ by showing that $[l] \geq [r]$ for the rules (1), (2) (where $[l]$ and $[r]$ are the interpretations of terms l and r which is obtained by structural induction), and $[u] > [v]$ for the dependency pairs (3), (4).

A recent and fruitful approach is using *matrix interpretations* [10], where the k-ary symbols f are given *parametric* matrix functions $[f]$, e.g., $F_1x_1 + \cdots + F_kx_k + F_0$, where the F_i's are (square) matrices of some fixed dimension n and F_0 is an n-tuple. The variables $x_1, \ldots, x_k$ are intended to range on n-tuples as well. In [10], only natural numbers are used both in matrices and n-tuples.

Example 2. The following matrix interpretation *over the naturals*

$$[f](x) = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} x + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad [g](x) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x \qquad [F](x) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} x$$

can also be used for proving termination of $\mathcal{R}$ in Example 1. Both interpretations have been automatically obtained by using MU-TERM [13].

In [1,2], Endrullis et al.'s framework was extended to matrices containing *real numbers* in the entries. The adaptation was motivated by a number of recent theoretical works and experimental evaluations showing that *polynomial* interpretations over the rationals can be advantageously used instead of polynomial interpretations over the naturals [5,11,14]. In [11], syntactic conditions ensuring that, when dealing with *linear* polynomial interpretations, real coefficients *must be* used for addressing the corresponding termination problem, were given for the first time. The extension of these results to matrix interpretations over the reals failed [1,2].

Thus, the following question arises: Are rational numbers somehow *unnecessary* when dealing with matrix interpretations? By examining the proofs of termination contributed to the International Competition on Termination[1] by tools like Jambox [9], which makes extensive use of matrix interpretations over the naturals, and comparing them to the corresponding ones generated by tools like MU-TERM, which emphasizes the use of polynomials over the rationals [15], one may notice that, in many cases, proofs of termination with polynomials over the rationals somehow correspond to proofs using matrix intepretations whose matrices have specific shapes. Examples 1 and 2 illustrate this connection, which we substantiate in this paper: the bigger are the *values* of the (possibly rational) coefficients in the polynomial interpretation the more non-null entries are in the

[1] See `http://termcomp.uibk.ac.at/termcomp/`

corresponding matrix coefficients. In this paper we investigate this phenomenon. In Section 2 we develop a notion of *numeric matrix representation* which permits a representation of a natural number as a matrix of (smaller) natural numbers. This representation preserves the usual arithmetics of natural numbers (addition and product). Therefore, we can think of such matrices as having a *value*; the value of the number from which they were obtained. Then, in Section 3 we show how to extend this process to transform a matrix of natural numbers into a (bigger) matrix of (smaller) natural numbers. As a consequence, we prove that every matrix of natural numbers can be represented as a *bit* matrix with entries in $\{0, 1\}$ which still preserves its 'value' and arithmetic behavior. In Section 4, we address the problem of representing arbitrary *rational numbers* as matrices of integer numbers. We argue that this is possible only for finite subsets of rational numbers. We investigate the use of *nilpotent* matrices (i.e., square matrices which become null after a finite number of selfproducts) as suitable devices for achieving this. In Section 5, we introduce a new generalization of matrix interpretations, which we call *block-based matrix interpretations*. Essentially, we view a matrix as structured into blocks of (sums of) *constant* or *scalar* matrices. In Section 6, we investigate how the satisfaction of different kind of constraints is *preserved* under the matrix transformations investigated in the previous sections. As a consequence of our results, we prove that TRSs which can be proved terminating by using matrix interpretations over the naturals can also be proved terminating by using a matrix interpretation based on *bit* matrices. Furthermore, we show that, under some conditions, proofs of termination which are carried out by polynomial or matrix intepretations over the rationals can also be obtained by using matrix interpretations over the naturals (like in Example 2). Section 7 summarizes our contribution and concludes.

2 Numbers as Matrices

In the following, we use the standard notations and terminology for matrices [12,17,18]. Given $p, q \in \mathbb{N}_{>0}$ and a set of numbers N (usually $\mathbb{R}$, $\mathbb{Q}$, or $\mathbb{N}$), we write $A \in N^{p\times q}$ to say that A is a matrix of p rows and q columns with entries $A_{ij} \in N$ (a $p \times q$-matrix for short). If $p = q$, then A is a *square* matrix.

We investigate the representation of (rational) numbers as matrices of integer numbers satisfying some structural properties. Of course, we want to ensure that the representation preserves (part of) the algebraic structure of the considered numeric domain, in such a way that, for instance, the arithmetic operations and orderings among numbers (of the considered kind) can be implemented by using matrix operations and orderings.

We view a rational number as a product $p\frac{1}{q}$ for an integer number $p \in \mathbb{Z}$ and a positive natural number $q \in \mathbb{N}_{>0}$. First, we formalize a generic framework for representing real numbers as matrices: mappings $\mu : \mathbb{R} \to \mathbb{R}^{m\times n}$ and $\rho : \mathbb{R}^{m\times n} \to \mathbb{R}$ provide a representation of real numbers as matrices and vice versa.

Definition 1 (Numeric matrix representation). *Let $N \subseteq \mathbb{R}$ be a subset of real numbers, $p, q \in \mathbb{N}_{>0}$, and $M \subseteq \mathbb{R}^{p\times q}$ be a subset of matrices. A* numeric matrix representation *is a pair* (μ, ρ) *of mappings* $\mu : N \to M$ *(called a* numeric representation*) and* $\rho : M \to N$ *(called a* matrix valuation*) such that* $\rho \circ \mu = id$.

Let $\mathbf{1}_{p\times q}$ be the $p \times q$-matrix all whose entries contain 1. A matrix $C = c\mathbf{1}_{p\times q}$ for some $c \in \mathbb{R}$ (i.e., whose entries are settled to c) is called a *constant* matrix. The identity (square) matrix of size n is denoted I_n. A matrix $S = cI_n$ for some $c \in \mathbb{R}$ is called a *scalar* matrix. We consider the following numeric matrix representation.

Definition 2. *Let* $m, p, q \in \mathbb{N}_{>0}$ *and* $A \in \mathbb{R}^{p\times q}$. *We let*

1. $\mu_m^{p\times q}(x) = \frac{mx}{pq}\mathbf{1}_{p\times q}$, *i.e., each real number* x *is mapped to a* constant $p \times q$-*matrix with entries* $\frac{mx}{pq}$.
2. $\rho_m(A) = \frac{\sum_{i=1}^{p}\sum_{j=1}^{q} A_{ij}}{m}$, *i.e., each matrix* A *is mapped to the number which is obtained by adding all entries in* A *and then dividing this number by* m.

In the following, we prove some properties of ρ_m which are used below.

Proposition 1. *Let* $m, p, q \in \mathbb{N}_{>0}$ *and* $A, B \in \mathbb{R}^{p\times q}$. *Then,* $\rho_m(A + B) = \rho_m(A) + \rho_m(B)$ *and* $\rho_m(\alpha A) = \alpha\rho_m(A)$ *for all* $\alpha \in \mathbb{R}$.

Proposition 2. *Let* $p, q, r \in \mathbb{N}_{>0}$, $A \in \mathbb{R}^{p\times q}$ *and* $B \in \mathbb{R}^{q\times r}$. *If* B *(resp.* A*) is scalar and* $q \leq r$ *(resp.* $q \leq p$*), or* B *(resp.* A*) is a constant matrix, then* $\rho_p(AB) = \rho_p(A)\rho_q(B)$.

Propositions 1 and 2 entail the following.

Corollary 1. *Let* $p, q, r \in \mathbb{N}_{>0}$, $A \in \mathbb{R}^{p\times q}$ *and* $B \in \mathbb{R}^{q\times r}$. *If* B *(resp.* A*) is an additive combination of scalar and constant matrices, then* $\rho_p(AB) = \rho_p(A)\rho_q(B)$.

Corollary 2. *Let* $n \in \mathbb{N}_{>0}$ *and* A, B *be* n-*square matrices. If* B *is an additive combination of scalar or constant matrices, then* $\rho_n(AB) = \rho_n(A)\rho_n(B) = \rho_n(BA)$.

If we consider only n-square matrices for representations, then $\mu'_n(x) = xI_n$ could also be used with ρ_n as a numeric matrix representation.

Remark 1 (Use of vectors). Since vectors $\boldsymbol{v}$ can be seen as special matrices $\boldsymbol{v} \in \mathbb{R}^{n\times 1}$ of n rows and a single column, the numeric matrix representation in Definition 2 can also be used to represent real numbers as vectors. In particular, we get $\mu_n^{n\times 1}(x) = x\mathbf{1}_n$ and $\rho_n(\boldsymbol{v}) = \frac{\sum_{i=1}^{n} v_i}{n}$.

3 Transforming Matrices of Numbers

We can *extend* any numeric representation $\mu : \mathbb{R} \to \mathbb{R}^{p\times q}$ to a mapping $\mu : \mathbb{R}^{m\times n} \to \mathbb{R}^{mp\times nq}$ from $m \times n$-matrices A into $mp \times nq$-matrices $\mu(A)$ by just replacing the numeric entries A_{ij} in A by the corresponding matrices $\mu(A_{ij})$, i.e., $\mu(A) = (\mu(A_{ij}))_{i=1,j=1}^{i=m,j=n}$. The new matrix can be viewed as a *block* matrix whose blocks are $\mu(A_{ij})$ for $1 \leq i \leq m$ and $1 \leq j \leq n$.

Example 3. We can transform the matrix $\begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}$ by using $\mu_m^{p\times q}$ in Definition 2; with $m = p = q = 3$, we obtain:

$$\mu_3^{3\times 3}(\begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}) = \begin{pmatrix} \mu_3^{3\times 3}(3) & \mu_3^{3\times 3}(0) \\ \mu_3^{3\times 3}(0) & \mu_3^{3\times 3}(3) \end{pmatrix} = \begin{pmatrix} \mathbf{1}_{3\times 3} & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \mathbf{1}_{3\times 3} \end{pmatrix}$$

The interesting feature of the matrix obtained in Example 3 is that it is a *bit* matrix whose entries are either[2] 0 or 1. Note that this is due to the use of $\mu_3^{3\times 3}$ which permits a representation of '3' as a constant matrix $\mathbf{1}_{3\times 3}$ with 1's only. For the numeric matrix representation in Definition 2, we have the following.

Proposition 3. *Let $m, n, p, q, r, s \in \mathbb{N}_{>0}$ and $A \in \mathbb{R}^{r\times s}$. Then, $\rho_{mn}(\mu_m^{p\times q}(A)) = \rho_n(A)$.*

Theorem 1. *Let $m, n, p, q, r, s, t, u \in \mathbb{N}_{>0}$. If $A, B \in \mathbb{R}^{r\times s}$, then, $\rho_m(A) + \rho_m(B) = \rho_{mn}(\mu_n^{p\times q}(A) + \mu_n^{p\times q}(B))$ and $\rho_m(\alpha A) = \alpha\rho_{mn}(\mu_n^{p\times q}(A))$ for all $\alpha \in \mathbb{R}$. If $A \in \mathbb{R}^{s\times t}$ and $B \in \mathbb{R}^{t\times u}$, then $\rho_m(AB) = \rho_{mn}(\mu_n^{p\times q}(A)\mu_n^{q\times r}(B))$.*

Propositions 1 and 2 entail the following.

Corollary 3. *Let $m, n, p, q, r, s \in \mathbb{N}_{>0}$, $A \in \mathbb{R}^{m\times m}$ and $B \in \mathbb{R}^{m\times s}$. If A is an additive combination of scalar and constant matrices, then $\rho_m(AB) = \rho_{mn}(\mu_n^{p\times q}(A)\mu_n^{q\times r}(B)) = \rho_{mn}(\mu_n^{p\times q}(A))\rho_{mn}(\mu_n^{q\times r}(B))$.*

3.1 Representing Integer Numbers as Matrices

In the following, we use $\mu_n^{p\times q}$ in Definition 2 as a basis for the following representation mapping for *integer numbers*.

Definition 3 (Representing integer numbers as matrices). *Let $n \in \mathbb{N}$ be such that $n > 1$ and μ_n be given as follows: for all $x \in \mathbb{Z}$,*

$$\mu_n(x) = \begin{cases} \frac{x}{n}\mathbf{1}_{n\times n} & \text{if } n \text{ divides } x \\ xI_n & \text{otherwise} \end{cases}$$

We also define $\nu_n(x) = x\mathbf{1}_n$ to represent a number x as a n-dimensional vector.

Example 4. The matrix $\mu_2(4) = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$ represents 4 according to Definition 3.

Note that, for all $n \in \mathbb{N}$, (μ_n, ρ_n) (with ρ_n as in Definition 2) is a numeric matrix representation for integer numbers. We obtain the following:

Proposition 4 (Decrease of natural entries in matrices). *Every matrix $A \in \mathbb{N}^{p\times q}$ such that $n = max(A) > 1$ can be represented as a matrix $A' \in \mathbb{N}^{np\times nq}$ such that, for all $m \in \mathbb{N}_{>0}$, $\rho_m(A) = \rho_{mn}(A')$ and $max(A) > max(A')$.*

Obviously, Proposition 4 entails the following.

Corollary 4 (Natural matrices as bit matrices). *Every matrix $A \in \mathbb{N}^{p\times q}$ can be represented as a bit matrix $A' \in \{0, 1\}^{np\times nq}$ for some $n \in \mathbb{N}_{>0}$ and for all $m \in \mathbb{N}_{>0}$, $\rho_m(A) = \rho_{mn}(A')$.*

[2] Matrices with entries in $\{0, 1\}$ are called $(0, 1)$-matrices in [18, Section 8.2].

4 Representation of Rational Numbers Below 1

In this section, we investigate matrix representations (over the naturals) which can be used to deal with rational numbers $\frac{1}{q}$ for some $q \in \mathbb{N}_{>0}$. A q-square matrix $A_{\frac{1}{q}}$ which is almost null except for a single entry of value 1 can be used to represent $\frac{1}{q}$ because $\rho_q(A_{\frac{1}{q}}) = \frac{1}{q}$. By Corollary 2, for $A_p = pI_q$ we get $\rho_q(A_pA_{\frac{1}{q}}) = \rho_q(A_{\frac{1}{q}}A_p) = \rho_q(A_p)\rho_q(A_{\frac{1}{q}}) = \frac{p}{q}$. Therefore, we can represent a rational number $\frac{p}{q}$ as a q-square matrix with a single entry of value p. However, we have to change the size of the matrix if a different number $\frac{p'}{q'}$ with $q \neq q'$ is considered.

Remark 2. Note that *there is no generic representation of all rational numbers by using matrices over the naturals of a given dimension which is able to represent their values and appropriate comparisons among them.* Such a representation should be able to represent a decreasing sequence $1 > \frac{1}{2} > \frac{1}{3} > \cdots > \frac{1}{n} > \cdots$ by means of matrices $A_{\frac{1}{n}} \in \mathbb{N}^{p\times q}$ for all $n \in \mathbb{N}_{>0}$ satisfying $\rho_m(A_{\frac{1}{n}}) > \rho_m(A_{\frac{1}{n+1}})$ for all $n \in \mathbb{N}_{>0}$. Equivalently, we should have $m\rho_m(A_{\frac{1}{n}}) > m\rho_m(A_{\frac{1}{n+1}})$ for all $n \in \mathbb{N}_{>0}$. Since $m\rho_m(A_{\frac{1}{n}}), m\rho_m(A_{\frac{1}{n+1}}) \in \mathbb{N}$, this would imply the existence of an infinite decreasing sequence of *natural* numbers, which is not possible.

Furthermore, the product of rational numbers $\frac{p}{q}$ and $\frac{p'}{q}$ represented as the q-square matrices indicated above is *not* preserved by the matrix product. Thus, in the following, we look for better representations.

4.1 Use of Nilpotent Matrices

Nilpotent matrices are n-square matrices B satisfying $B^k = \mathbf{0}_{n\times n}$ for some positive integer $k \in \mathbb{N}_{>0}$ [18, Section 4.1] (which is called the *degree of nilpotency* [12, page 12] or the *index* of nilpotency of B [17, page 396]). The degree of nilpotency k of a n-square matrix A is bounded by n: $k \leq n$ [17, Exercise 7.7.1]. Given n, the following n-square matrix (called a *Jordan block* [17, Page 579]):

$$J_n = \begin{pmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{pmatrix}$$

(i.e., there are $n-1$ ones in the superdiagonal of J_n and all other entries in J_n are zeroes) is nilpotent of degree n, i.e., $J_n^n = \mathbf{0}_{n\times n}$.

Write $J_n = [0, Z_1, \ldots, Z_{n-1}]$, where Z_i is an almost null vector containing a single 1 in the i-th entry. For instance, $Z_1 = (1, 0, \ldots, 0)^T$. Then, it is not difficult to see that J_n^p is obtained by introducing $p-1$ columns of zeros from the left side of the matrix and shifting columns Z_i to the right just throwing those which exceed the n-th position, i.e, $J_n^p = [0, \ldots, 0, Z_1, \ldots, Z_{n-p}]$.

Remark 3. J_n Is also known as a *shift* matrix because, for an arbitrary matrix A, J_nA is obtained by shifting the rows of A upwards by one position and introducing a new bottom row of zeroes. Similarly AJ_n shifts the columns of A to the right and introduces a new leftmost column of zeroes.

Note that, for all $p \in \mathbb{N}$,

$$\rho_m(J_n^p) = \begin{cases} \frac{n-p}{m} & \text{if } p < n \\ 0 & \text{if } p \geq n \end{cases}$$

The following result is obvious.

Proposition 5. *Let $m, n \in \mathbb{N}_{>0}$ and $0 \leq p < q \leq n$. Then, $\rho_m(J_n^p) = \rho_m((J_n^T)^p) > \rho_m((J_n^T)^q) = \rho_m(J_n^q)$. For all $A \in \mathbb{R}^{n \times r}$, $\rho_m(J_n^p A) \geq \rho_m(J_n^q A)$. For all $A \in \mathbb{R}^{r \times n}$, $\rho_m(AJ_n^p) \geq \rho_m(AJ_n^q)$.*

In general it is *not* true that $\rho_m(J^p A) = \rho_m(AJ^p)$ for square matrices A. Due to Proposition 5 and Corollary 1, for additive combinations A of *constant* and *scalar* matrices we have the following:

Corollary 5. *Let $m, n \in \mathbb{N}_{>0}$ and $0 \leq p < q \leq n$. If $A \in \mathbb{R}^{n \times r}$ (resp. $A \in \mathbb{R}^{r \times n}$) is an additive combination of constant and scalar matrices, then $\rho_m(J_n^p A) = \rho_m(J_n^p)\rho_m(A) > \rho_m(J_n^q)\rho_m(A) = \rho_m(J_n^q A)$ (resp. $\rho_m(AJ_n^p) = \rho_m(A)\rho_m(J_n^p) > \rho_m(A)\rho_m(J_n^q) = \rho_m(AJ_n^q)$).*

Thus, the following property of rational numbers $r = \frac{1}{n}$ for some $n \in \mathbb{N}_{>0}$ is simulated by the representation of integer numbers as (additive combinations of) constant or scalar matrices, and rational numbers $\frac{1}{n}$ as powers of Jordan blocks: for all $n \in \mathbb{N}$ and positive rational number r, $0 < r < 1$, we have $n > nr > nr^2 > \cdots$.

Example 5. We can use J_2 to represent $\frac{1}{2}$: $\rho_2(J_2) = \frac{1}{2}$. However, J_2^2, which is expected to correspond to $\frac{1}{4}$ does *not* fit this expectation: $\rho_2(J_2^2) = \rho_2(\mathbf{0}_2) = 0$.

Theorem 2. *If $A_1, \cdots, A_N$ are n-square matrices such that, for all i, $1 \leq i \leq N$, either*

1. *A_i is scalar or constant, or*
2. *$A_i = J_n^{p_i}$ for some $p_i \in \mathbb{N}$ and then both A_{i-1} (if $i > 1$) and A_{i+1} (if $i < N$) are constant matrices,*

then $\rho_n(\prod_{i=1}^N A_i) = \prod_{i=1}^N \rho_n(A_i)$.

As remarked above, it is *not* possible to use matrices of natural numbers of a fixed size n to represent *all* fractions $\frac{1}{q}$ for $q \in \mathbb{N}_{>0}$. Instead, we will consider the problem of representing *finite subsets* $\mathcal{Q} \subseteq \{\frac{1}{q} \mid q \in \mathbb{N}_{>0}\}$ by using *n-square* (nilpotent) matrices in such a way that the following property is fulfilled by the representation (μ, ρ): for all $x, y \in \mathcal{Q}$ such that $xy \in \mathcal{Q}$, $\rho(\mu(x)\mu(y)) = xy = \rho(\mu(x))\rho(\mu(y))$, i.e., the number $\rho(\mu(x)\mu(y))$ which corresponds to the matrix product $\mu(x)\mu(y)$ of matrices $\mu(x)$ and $\mu(y)$ representing $x \in \mathcal{Q}$ and $y \in \mathcal{Q}$ is exactly $xy \in \mathcal{Q}$. In Section 6 we discuss how to take benefit from this.

The dimension n of the matrices involved in the representation of $\mathcal{Q}$ is determined by the *least* element in $\mathcal{Q}$. For instance, we can fix n such that $\frac{1}{n}$ is the least number in $\mathcal{Q}$. Then, an obvious representative for $\frac{1}{n}$ is J_n^{n-1} because $\rho_n(J_n^{n-1}) = \frac{1}{n}$ (see Example 5). However, the feasibility of this simple approach depends on the other values in $\mathcal{Q}$.

Example 6. Let $\mathcal{Q} = \{\frac{1}{2}, \frac{1}{4}\}$. If we fix J_4^3 to be the representation of $\frac{1}{4}$, then the representation of $\frac{1}{2}$ should be J_4^2 (because $\rho_4(J_4^2) = \frac{1}{2}$). However, $\rho_4((J_4^2)^2)$ is *not* $\frac{1}{4}$ as one could expect; instead, $\rho_4((J_4^2)^2) = 0$. The following *block* matrices of size 4 whose blocks are combinations of (transposed) Jordan blocks can be used to represent $\mathcal{Q}$ as required:

$$Q_{\frac{1}{2}} = \begin{pmatrix} J_2 & J_2^T \\ \mathbf{0}_{2\times 2} & \mathbf{0}_{2\times 2} \end{pmatrix} \qquad Q_{\frac{1}{4}} = \begin{pmatrix} \mathbf{0}_{2\times 2} & J_2 J_2^T \\ \mathbf{0}_{2\times 2} & \mathbf{0}_{2\times 2} \end{pmatrix}$$

Note that $\rho_4(Q_{\frac{1}{2}}) = \frac{1+1}{4} = \frac{1}{2}$, $\rho_4(Q_{\frac{1}{4}}) = \frac{0+1}{4} = \frac{1}{4}$, and $Q_{\frac{1}{2}} Q_{\frac{1}{2}} = Q_{\frac{1}{4}}$.

Example 7. Let $\mathcal{Q} = \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}\}$. The following matrices of size 8:

$$Q_{\frac{1}{2}} = \begin{pmatrix} J_4 & J_4^3 \\ \mathbf{0}_{4\times 4} & \mathbf{0}_{4\times 4} \end{pmatrix} \quad Q_{\frac{1}{4}} = \begin{pmatrix} J_4^2 & \mathbf{0}_{4\times 4} \\ \mathbf{0}_{4\times 4} & \mathbf{0}_{4\times 4} \end{pmatrix} \quad Q_{\frac{1}{8}} = \begin{pmatrix} J_4^3 & \mathbf{0}_{4\times 4} \\ \mathbf{0}_{4\times 4} & \mathbf{0}_{4\times 4} \end{pmatrix}$$

can be used to represent $\mathcal{Q}$. Note that $\rho_8(Q_{\frac{1}{2}}) = \frac{3+1}{8} = \frac{1}{2}$, $\rho_8(Q_{\frac{1}{4}}) = \frac{2+0}{8} = \frac{1}{4}$, $\rho_8(Q_{\frac{1}{8}}) = \frac{1+0}{8} = \frac{1}{8}$, $Q_{\frac{1}{2}} Q_{\frac{1}{2}} = Q_{\frac{1}{4}}$ and $Q_{\frac{1}{2}} Q_{\frac{1}{4}} = Q_{\frac{1}{4}} Q_{\frac{1}{2}} = Q_{\frac{1}{8}}$, as required.

Example 8. Let $\mathcal{Q} = \{\frac{1}{2}, \frac{1}{3}, \frac{1}{6}\}$. The following (block) matrices of size 6 can be used to represent $\mathcal{Q}$:

$$Q_{\frac{1}{2}} = \begin{pmatrix} J_3 & (J_3^2)^T \\ \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} \end{pmatrix} \quad Q_{\frac{1}{3}} = \begin{pmatrix} J_3^2 & J_3(J_3^2)^T \\ \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} \end{pmatrix} \quad Q_{\frac{1}{6}} = \begin{pmatrix} \mathbf{0}_{3\times 3} & J_3^2(J_3^2)^T \\ \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} \end{pmatrix}$$

Note that $\rho_6(Q_{\frac{1}{2}}) = \frac{2+1}{6} = \frac{1}{2}$, $\rho_6(Q_{\frac{1}{3}}) = \frac{1+1}{6} = \frac{1}{3}$, and $\rho_6(Q_{\frac{1}{6}}) = \frac{0+1}{6} = \frac{1}{6}$. Again, it is not difficult to see that $Q_{\frac{1}{2}} Q_{\frac{1}{3}} = Q_{\frac{1}{3}} Q_{\frac{1}{2}} = Q_{\frac{1}{6}}$. Note, however, that if we add $\frac{1}{4}$ to $\mathcal{Q}$, then we could *not* use these matrices for representing $\frac{1}{4}$. In particular, $Q_{\frac{1}{2}}^2 = Q_{\frac{1}{3}}$, i.e., $\rho_6(Q_{\frac{1}{2}}^2) \neq \frac{1}{4}$ as should be the case.

5 Matrix Interpretations Revisited

As remarked above, termination problems in term rewriting are usually translated into conjunctions of *weak* or *strict symbolic constraints* like $s \succeq t$ or $s \succ t$ between terms s and t coming from (parts of) the TRS. In order to check the satisfaction of these constraints, we need to use *term (quasi-)orderings*. Such term orderings can be obtained by giving appropriate *interpretations* to the function symbols of a signature. Given a signature $\mathcal{F}$, an $\mathcal{F}$-algebra is a pair $\mathcal{A} = (\mathsf{A}, \mathcal{F}_\mathsf{A})$, where A is a set and $\mathcal{F}_\mathsf{A}$ is a set of mappings $f_\mathcal{A} : \mathsf{A}^k \to \mathsf{A}$ for each $f \in \mathcal{F}$ where $k = ar(f)$. For a given valuation mapping $\alpha : \mathcal{X} \to \mathsf{A}$, the evaluation mapping $[\alpha] : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathsf{A}$ is inductively defined by $[\alpha](x) = \alpha(x)$ if $x \in \mathcal{X}$ and $[\alpha](f(t_1, \ldots, t_k)) = f_\mathcal{A}([\alpha](t_1), \ldots, [\alpha](t_k))$ for $x \in \mathcal{X}$, $f \in \mathcal{F}$,

$t_1, \ldots, t_k \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Given a term t with $\mathcal{V}ar(t) = \{x_1, \ldots, x_n\}$, we write $[t]$ to denote the *function* $F_t : \mathsf{A}^n \to \mathsf{A}$ given by $F_t(a_1, \ldots, a_n) = [\alpha_{(a_1,\ldots,a_n)}](t)$ for each tuple $(a_1, \ldots, a_n) \in \mathsf{A}^n$, where $\alpha_{(a_1,\ldots,a_n)}(x_i) = a_i$ for $1 \le i \le n$. We can define a stable quasi-ordering $\succeq$ on terms given by $t \succeq s$ if and only if $[\alpha](t) \succeq_{\mathsf{A}} [\alpha](s)$, for all $\alpha : \mathcal{X} \to \mathsf{A}$, where $\succeq_{\mathsf{A}}$ is a quasi-ordering on A. We can define a stable strict ordering $\sqsupset$ on terms by $t \sqsupset s$ if $[\alpha](t) \succ_{\mathsf{A}} [\alpha](s)$, for all $\alpha : \mathcal{X} \to \mathsf{A}$, where $\succ_{\mathsf{A}}$ is a strict ordering on A.

A matrix interpretation for a k-ary symbol f is a linear expression $F_1x_1 + \cdots + F_kx_k + F_0$ where the $F_1, \ldots, F_k$ are (square) matrices of $n \times n$ natural numbers and the variables $x_1, \ldots, x_k$ (and also the constant term F_0) are n-tuples of natural numbers [10]. An expression like $F\boldsymbol{x}$, where F is an n-square matrix and $\boldsymbol{x}$ is an n-tuple of numbers, is interpreted as the usual matrix-vector product, i.e., the i-th component y_i of $\boldsymbol{y} = F\boldsymbol{x}$ is $y_i = \sum_{j=1}^{n} F_{ij}x_j$. Matrices and vectors are compared by using an *entrywise* ordering: for $A, B \in \mathbb{R}^{p\times q}$, we write $A \ge B$ if $A_{ij} \ge B_{ij}$ for all $1 \le i \le p$, $1 \le j \le q$ [12, Chapter 15]. We also write $A > B$ if $A \ge B$ and $A_{11} > B_{11}$ [10]. Note that this also describes how to compare tuples of numbers. In [1,2], Endrullis et al.'s approach was extended to matrix interpretations with real numbers in matrix entries.

Here, we generalize the notion of matrix interpretation in the following ways:

1. We consider a domain $T_{n,b}(N)$ of *block-based* tuples *over* N, i.e., n-tuples consisting of $\beta = \frac{n}{b}$ tuples of size $b \in \mathbb{N}_{>0}$ which are *constant* tuples $c\mathbf{1}_b$ for some number $c \in N$. If we take $b = 1$ and $N = \mathbb{N}$ or $N = \mathbb{R}_0$, then we are in the original approaches [10] and [1,2], respectively. Note, however, that given $n \in \mathbb{N}_{>0}$, only divisors b of n can be used to establish blocks within n-tuples of numbers.
2. Matrices F in matrix expressions interpreting symbols $f \in \mathcal{F}$ will be *block matrices* $\begin{pmatrix} F_{11} & \cdots & F_{1\beta} \\ \vdots & \ddots & \vdots \\ F_{\beta 1} & \cdots & F_{\beta\beta} \end{pmatrix}$ such that $F_{ij} = C_{ij} + S_{ij}$ is a sum of a b-square *constant* matrix $C_{ij} = c_{ij}\mathbf{1}_{b\times b}$, where $c_{ij} \in N$, and a b-square *scalar* matrix $S_{ij} = s_{ij}I_b$, where $s_{ij} \in N$, for all $1 \le i,j \le \beta$. This is necessary for soundness of the obtained algebraic interpretation: the product of one of such matrices by a block-based tuple as above produces a block-based tuple as above, i.e., $F\boldsymbol{v} \in T_{n,b}(N)$ for all $\boldsymbol{v} \in T_{n,b}(N)$. Furthermore, such matrices are *closed* under addition and matrix product. This is essential to obtain matrices of this kind during the interpretation of the terms, where nested symbols yield products and sums of matrices after interpreting them. Again, if $b = 1$, we are in the usual case for matrix interpretations.
3. Given a matrix valuation ρ, and matrices A, B, we let $A \ge_\rho B$ if $\rho(A) \ge_N \rho(B)$, where $\ge_N$ is an ordering over N. Given $\delta > 0$, the following ordering over (real) numbers is used [15]: $x >_\delta y$ if $x - y \ge \delta$. If $\delta = 1$ and x, y are natural numbers, we obtain the usual well-founded ordering among natural numbers $>_{\mathbb{N}}$. Now, the following (strict and well-founded) ordering $>_{\rho,\delta}$ (or just $>_\rho$ or even $>$ if it is clear from the context) on n-tuples of *nonnegative*

numbers is considered: $\boldsymbol{x} >_{\rho,\delta} \boldsymbol{y}$ if $\rho(x) >_\delta \rho(y)$. Clearly, $(T_{n,b}(\mathbb{N}), >_{\rho,1})$ and $(T_{n,b}(\mathbb{R}_0), >_{\rho,\delta})$ are well-founded orderings.

The previous orderings do *not* take into account the block structure of matrices or tuples. The following one does: for matrices A, B with blocks A_{ij} and B_{ij} for $1 \leq i, j \leq \beta$, we write $A \geq_\rho^b B$ if $A_{ij} \geq_\rho B_{ij}$ for all $1 \leq i, j \leq \beta$. Similarly, $A >_{\rho,\delta}^b B$ if $A_{11} >_{\rho,\delta} B_{11}$ and $A_{ij} \geq_\rho B_{ij}$ for all $1 \leq i, j \leq \beta$. These definitions are adapted to tuples in the obvious way.

Definition 4 (Block-based matrix interpretation). *Let $\mathcal{F}$ be a signature, $n, b \in \mathbb{N}_{>0}$ be such that b divides n, and $\beta = \frac{n}{b}$. An* (n,b)-block-based matrix interpretation *is an $\mathcal{F}$-algebra $\mathcal{A} = (A, \mathcal{F}_\mathcal{A})$ such that*

1. *$A = T_{n,b}(N)$ for $N = \mathbb{N}$ or $N = \mathbb{R}_0$, and*
2. *$\mathcal{F}_\mathcal{A}$ consists of matrix functions $[f](x_1, \ldots, x_k) = F_1x_1 + \cdots + F_kx_k + F_0$ which are closed in A and where, for all $f \in \mathcal{F}$, $1 \leq i \leq k$,*
   - *(a) F_i is an n-square matrix of $\beta \times \beta$ blocks of b-square matrices $C_{j\ell} + S_{j\ell}$ such that $C_{j\ell}$ is a constant matrix, and $S_{j\ell}$ is a scalar matrix for all $1 \leq j \leq \beta$ and $1 \leq \ell \leq \beta$.*
   - *(b) $F_0 = (\boldsymbol{c}_1^T \cdots \boldsymbol{c}_\beta^T)^T$ consists of β b-tuples $\boldsymbol{c}_j = c_j\mathbf{1}_b$ for some $c_j \in N$.*

We make the orderings which are going to be used explicit by adding them to the pair $(A, \mathcal{F}_\mathcal{A})$, thus specifying an ordered *algebra: $(A, \mathcal{F}_\mathcal{A}, \geq_{\rho_n})$, $(A, \mathcal{F}_\mathcal{A}, \geq_{\rho_b}^b)$, etc.*

6 Solving and Transforming Matrix Constraints

In the following, we consider two kinds of constraint solving problems which are relevant in proofs of termination.

6.1 Testing Universally Quantified Symbolic Constraints

Proofs of termination in term rewriting involve solving *weak* or *strict* symbolic constraints $s \succeq t$ or $s \sqsupset t$ between terms s and t coming from (parts of) the rules of the TRS where the variables in s and t are universally quantified in the corresponding constraint. Here, $\succeq$ and $\sqsupset$ are (quasi-)orderings on terms satisfying appropriate conditions [3,6,8].

Example 9. The following symbolic constraints must be checked in order to guarantee termination of the TRS $\mathcal{R}$ in Example 1:

$$\forall X(f(f(X)) \succeq f(g(f(X)))) \tag{5}$$

$$\forall X(f(g(f(X))) \succeq X) \tag{6}$$

$$\forall X(F(f(X)) \sqsupset F(g(f(X)))) \tag{7}$$

$$\forall X(F(f(X)) \sqsupset F(X)) \tag{8}$$

Here, variables X range on terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, $\succeq$ and $\sqsupset$ are intended to be interpreted as a monotonic and stable quasiordering on terms, and a well-founded and stable ordering on terms, respectively.

In the so-called *interpretation method* for checking symbolic constraints as the ones in Example 9, we use appropriate *ordered $\mathcal{F}$-algebras* to *generate* the necessary orderings (see Section 5). In our setting, we are interested in investigating the use of matrix algebras. The following result shows how to *decrease* the value of some of the entries in a block-based matrix algebra over the naturals to obtain an *equivalent* block-based matrix algebra which uses bigger matrices.

Theorem 3. *Let $\mathcal{F}$ be a signature and c be a symbolic constraint $\forall x(s \bowtie t)$ for terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Let $\mathcal{A} = (T_{n,b}(\mathbb{N}), \mathcal{F}_\mathcal{A}, \geq^b_{\rho_b})$ be an (n, b)-block-based matrix interpretation over the naturals and N be the maximum of all entries occurring in any matrix or vector in $\mathcal{A}$. Let $\mathcal{B} = (B, \mathcal{F}_\mathcal{B}, \geq^{bN}_{\rho_{bN}})$ be an (Nn, Nb)-block-based matrix interpretation where $B = T_{Nn,Nb}(\mathbb{N})$ and for all $f \in \mathcal{F}$, $[f]_\mathcal{B}(x_1, \ldots, x_k) = \mu_N(F_1)x_1 + \cdots + \mu_N(F_k)x_k + \nu_N(F_0)$ iff $[f]_\mathcal{A}(x_1, \ldots, x_k) = F_1x_1 + \cdots + F_kx_k + F_0$. Then, $\mathcal{A}$ satisfies c if and only if $\mathcal{B}$ satisfies c.*

Example 10. Consider the following TRSs [10, Example 3]:

$$\mathcal{R} : f(a, g(y), z) \to f(a, y, g(y)) \tag{9}$$
$$f(b, g(y), z) \to f(a, y, z) \tag{10}$$
$$a \to b \tag{11}$$
$$\mathcal{S} : f(x, y, z) \to f(x, y, g(z)) \tag{12}$$

In order to prove termination of $\mathcal{R}$ *relative* to $\mathcal{S}$ (written $SN(R/S)$), we have to check that $\forall x(l \succeq r)$ holds for all rules $l \to r \in \mathcal{R} \cup \mathcal{S}$. Endrullis et al. use the following matrix interpretation for that:

$$[a] = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad [f](x, y, z) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} x + \begin{pmatrix} 1 & 2 \\ 0 & 0 \end{pmatrix} y + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} z + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$
$$[b] = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad [g](x) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} x + \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

By using Theorem 3, we conclude that the following block-based matrix interpretation with bit matrices of dimension 4 can also do the work.

$$[a] = \begin{pmatrix} \mathbf{1}_2 \\ \mathbf{0}_2 \end{pmatrix} \qquad [f](x, y, z) = \begin{pmatrix} I_2 & \mathbf{0}_{2\times 2} \\ \mathbf{0}_{2\times 2} & \mathbf{0}_{2\times 2} \end{pmatrix} x + \begin{pmatrix} I_2 & \mathbf{1}_{2\times 2} \\ \mathbf{0}_{2\times 2} & \mathbf{0}_{2\times 2} \end{pmatrix} y + \begin{pmatrix} I_2 & \mathbf{0}_{2\times 2} \\ \mathbf{0}_{2\times 2} & \mathbf{0}_{2\times 2} \end{pmatrix} z + \begin{pmatrix} \mathbf{0}_2 \\ \mathbf{0}_2 \end{pmatrix}$$
$$[b] = \begin{pmatrix} \mathbf{0}_2 \\ \mathbf{0}_2 \end{pmatrix} \qquad [g](x) = \begin{pmatrix} I_2 & \mathbf{0}_{2\times 2} \\ I_2 & I_2 \end{pmatrix} x + \begin{pmatrix} \mathbf{0}_2 \\ \mathbf{1}_2 \end{pmatrix}$$

6.2 Solving Existentially Quantified Arithmetic Constraints

In many applications, when algebraic interpretations have to be *generated* rather than given by the user, it is usual to work with *parametric* interpretations.

Example 11. For the symbols occurring in $\mathcal{R}$ in Example 1, we can consider the following linear *parametric* interpretation:

$$[f](x) = f_1x + f_0 \qquad [g](x) = g_1x + g_0 \qquad [F](x) = F_1x + F_0$$

where f_1, g_1, and F_1 are expected to be n-square matrices for some $n \in \mathbb{N}_{>0}$ (the case $n = 1$ corresponds to a linear polynomial intepretation) and f_0, g_0, F_0 are n-tuples. The satisfaction of the following *arithmetic* constraints[3]

$$f_1 f_1 \geq f_1 g_1 f_1 \tag{13}$$
$$f_1 f_0 + f_0 \geq f_1 g_1 f_0 + f_1 g_0 + f_0 \tag{14}$$
$$f_1 g_1 f_1 \geq 1 \tag{15}$$
$$f_1 g_1 f_0 + f_1 g_0 + f_0 \geq 0 \tag{16}$$
$$F_1 f_1 \geq F_1 g_1 f_1 \tag{17}$$
$$F_1 f_0 + F_0 > F_1 g_1 f_0 + F_1 g_0 + F_0 \tag{18}$$
$$F_1 f_1 \geq F_1 \tag{19}$$
$$F_1 f_0 + F_0 > 0 \tag{20}$$

is necessary to ensure that $\mathcal{R}$ is terminating.

By a parametric matrix interpretation we mean a matrix intepretation where the entries in matrices are not numbers but rather *parametric coefficients*, i.e., variables for which we have to provide a numeric value, depending on some (existential) constraints.

In general, given a set of variables $\mathcal{X}$, we consider here symbolic arithmetic constraints of the form $s \bowtie t$ for $\bowtie \in \{\geq, >\}$, where s is of the form $\sum_{i=1}^{m_s} s_i$ for $m_s > 0$ and $s_i = s_{i1} \cdots s_{im_{s,i}}$ with $m_{s,i} > 0$ for all $1 \leq i \leq \sigma_s$ and $s_{ij} \in \mathcal{X}$ (t would have an analogous structure). Furthermore, we assume existential quantification over all variables occurring in s and t. Note in Example 11 that we use *constants* like 0 and 1. They are handled as 'special' variables which will receive the intended interpretation.

Consider a *valuation* $\eta : \mathcal{X} \to N$ for the variables in $\mathcal{X}$ as numbers in N. Here, we consider $N = \mathbb{N} \cup \mathcal{Q}$ for some finite subset of rational numbers $\mathcal{Q} \subseteq \mathbb{Q} - \mathbb{N}$ satisfying some conditions. We are interested in representing numbers $\eta(x)$ as matrices $\mu(\eta(x))$ as discussed above. Note that we *cannot* represent arbitrary rational numbers by using matrices over the naturals (Remark 2). Still, when dealing with finite sets $\mathcal{C}$ of arithmetic restrictions, we can restrict the attention to those rational numbers which are required to check its satisfaction for a given valuation η. This includes not only rational numbers $\eta(x)$ which are assigned to $x \in \mathcal{X}$, but also those which could occur during the evaluation of an arithmetic expression due to products $\eta(x)\eta(y)$ of rational numbers $\eta(x)$ and $\eta(y)$ which have been associated to variables x and y. The idea is that $\mathcal{Q}$ should contain such rational numbers.

Definition 5 (Compatible domain of rational numbers). *Let $\eta : \mathcal{X} \to \mathbb{N} \cup \mathcal{Q}$ be a valuation for some $\mathcal{Q} \subseteq \mathbb{Q} - \mathbb{N}$ and $\mathcal{C}$ be a set of arithmetic constraints. Given a multiplicative component $s = s_1 \cdots s_m$ of an arithmetic expression in a constraint in $\mathcal{C}$, let $I_s = \{i_1, \ldots, i_k\}$ be the set of indices of variables in s whose*

[3] By lack of space, we cannot explain how these constraints are obtained. Full details about this standard procedures can be found elsewhere, see, e.g., [7,10,15,16].

valuation is a rational (and noninteger) number, i.e., for all $i \in I_s$, $\eta(s_i) \in \mathbb{Q}-\mathbb{N}$. We say that $\mathcal{Q}$ is compatible *with $\mathcal{C}$ and η if $\prod_{j\in J} \eta(s_i) \in \mathcal{Q}$ for all multiplicative components s in $\mathcal{C}$ and $J \subseteq I_s$ such that $J \neq \varnothing$.*

Example 12. The numeric valuation which corresponds to the poynomial interpretation in Example 1 is:

$$\begin{array}{cccc} \eta(1) = 1 & \eta(0) = 0 & \eta(f_1) = 2 & \eta(f_0) = 2 \\ \eta(g_1) = \frac{1}{2} & \eta(g_0) = \frac{1}{2} & \eta(F_1) = 1 & \eta(F_0) = 0 \end{array}$$

The set $\mathcal{Q} = \{\frac{1}{2}\}$ is compatible with this valuation and with the constraints $\mathcal{C}$ in Example 11. Consider $\mathcal{C}' = \mathcal{C} \cup \{f_1 g_1 f_1 \geq f_1 g_1 f_1 g_1\}$. Now, $\mathcal{Q}$ is *not* compatible with $\mathcal{C}'$ and η because we have that $\eta(g_1)\eta(g_1) = \frac{1}{4} \notin \mathcal{Q}$. If we add $\frac{1}{4}$ to $\mathcal{Q}$, then we get compatibility with $\mathcal{C}'$.

A valuation η is extended to terms and constraints by $\eta(s \bowtie t) = \eta(s) \bowtie_{\mathbb{Q}} \eta(s)$, $\eta(s_1 + \cdots + s_m) = \eta(s_1) + \cdots + \eta(s_m)$, and $\eta(x_1 \cdots x_m) = \eta(x_1) \cdots \eta(x_m)$.

Remark 4. In general, we assume that $+$ is commutative, but (as happens with the matrix product) we do *not* assume commutativity of the product in arithmetic expressions or their valuations.

Now we have to extend μ_n in Definition 3 to deal with rational numbers in $\mathcal{Q}$.

Remark 5. In contrast to natural numbers, we have no systematic way to associate matrices to rational numbers yet. In Section 4 we have investigated some partial solutions to this problem. In particular, Examples 5, 6, 7, and 8, show that the *dimension* n of the considered matrices and tuples heavily depend on the numbers in $\mathcal{Q}$. On the other hand, these examples also provide useful encodings for rational numbers which are frequently used in automatic proofs of termination with polynomial or matrix interpretations [5,11,16].

Assume that μ_n has been extended to each $x \in \mathcal{Q}$ in such a way that: for all $x, y \in \mathcal{Q}$ such that $xy \in \mathcal{Q}$, $\rho_n(\mu_n(x)\mu_n(y)) = xy = \rho_n(\mu_n(x))\rho_n(\mu_n(y))$.

In our setting, variables in $\mathcal{X}$ are interpreted not only as matrices but some of them as *vectors*. Assume that $\mathcal{X}_0 \subseteq \mathcal{X}$ must be interpreted in this way and that the constraints in $\mathcal{C}$ are consistent with this, i.e., whenever a variable $x_0 \in \mathcal{X}_0$ occurs in a constraint $\mathsf{c} \in \mathcal{C}$ of the form $s \bowtie t$, each multiplicative term in s and t must contain a single variable $y_0 \in \mathcal{X}_0$ which must be at the end of the term.

Example 13. For the constraints in Example 11, we have $\mathcal{X}_0 = \{f_0, F_0, g_0, 0\}$.

The vectorial (or n-tuple) representation of $x \in \mathbb{N} \cup \mathcal{Q}$ by μ_n is $\mu_n(x)\mathbf{1}_n$.

Example 14. The matrices over the naturals which correspond to η and $\mathcal{Q}$ in Example 12 is (with the encoding of $\frac{1}{2}$ in Example 5) are:

$$\mu(\eta(f_1)) = \mathbf{1}_{2\times 2} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad \mu(\eta(f_0)) = 2\,\mathbf{1}_2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \quad \mu(\eta(g_1)) = J_2 = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

$$\mu(\eta(g_0)) = J_2\mathbf{1}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \mu(\eta(F_1)) = I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \mu(\eta(F_0)) = 0\,\mathbf{1}_2 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\mu(\eta(1)) = I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \mu(\eta(0)) = 0\,\mathbf{1}_2 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Accordingly, the matrix interpretation over the naturals which corresponds to the polynomial interpretation over the rationals in Example 1 is:

$$[f](x) = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} x + \begin{pmatrix} 2 \\ 2 \end{pmatrix} \quad [g](x) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} x + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad [F](x) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} x$$

This matrix interpretation induced from the polynomial interpretation over the rationals in Example 1 can also be used to solve the constraints in the example.

Our technique could be used to translate matrix interpretations *over the rationals* into matrix interpretations *over the naturals* by just applying the previous translation to the *entries* of the matrix interpretation over the rationals instead to the coefficients of the polynomial interpretation.

7 Conclusions

We have investigated matrix representations of natural and rational numbers which can be used to simulate the arithmetics of natural and rational numbers, respectively. We have introduced the notion of *numeric matrix representation* (Definition 1) which associates a number to a matrix and viceversa and proved that, by using some specific representations (Definitions 2 and 3), the arithmetic of natural numbers is preserved. Furthermore, we have proved that every matrix interpretation over the naturals has an associated bit matrix interpretation of (usually) bigger size of the same associated value (Corollary 4). We have investigated the representation of rational numbers by using matrices of natural numbers. We have proved that this problem has no general solution but we have found some suitable trade-offs for finite subsets of rational numbers by using nilpotent matrices consisting of Jordan blocks. Then we have introduced the notion of block-based matrix interpretation (Definition 4) which generalizes existing approaches to matrix interpretations. We use it to transform matrix interpretations over the naturals into matrix interpretations over $\{0, 1\}$, and also to transform matrix interpretations over the rationals into matrix interpretations over the naturals.

The question posed in the introduction: *are rational numbers somehow* unnecessary *when dealing with matrix interpretations?* could not be answered in full generality due to the lack of a general procedure for building matrix representations for arbitrary finite sets of rational numbers. Of course, this is a main topic of further research and we think that we have settled a good starting point in considering the use of combinations of Jordan blocks as in Examples 5, 6, 7, and 8. Nevertheless, our results suggest that the use of matrices over the naturals of *big size* can somehow play the role of rational numbers in interpretations of smaller size, and in particular in linear polynomial intepretations over the rationals. This does *not* mean that implementing polynomial or matrix intepretations over the rationals is not useful anymore and that natural numbers should be used everywhere. In fact, working with matrix interpretations of big size is computationally expensive. Another interesting consequence of our analysis is the connection between dimension of matrices over the naturals and value of their entries via Proposition 4 and Corollary 4. Roughly speaking, these results can be interpreted by saying that bigger dimensions of matrices permit the use of smaller entries. In practice, most tools put strong numeric bounds to the coefficients or entries of the interpretations. Our results suggest that increasing such bounds could have a similar effect to increasing the size of the matrices. A more precise analysis about the trade-offs in design and efficiency which these considerations could lead to is also subject for future work.

References

1. Alarcón, B., Lucas, S., Navarro-Marset, R.: Proving Termination with Matrix Interpretations over the Reals. In: Proc. of WST 2009, pp. 12–15 (2009)
2. Alarcón, B., Lucas, S., Navarro-Marset, R.: Using Matrix Interpretations over the Reals in Proofs of Termination. In: Proc. of PROLE 2009, pp. 255–264 (2009)
3. Arts, T., Giesl, J.: Termination of Term Rewriting Using Dependency Pairs. Theoretical Computer Science 236, 133–178 (2000)
4. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
5. Borralleras, C., Lucas, S., Navarro-Marset, R., Rodríguez-Carbonell, E., Rubio, A.: Solving Non-linear Polynomial Arithmetic via SAT Modulo Linear Arithmetic. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 294–305. Springer, Heidelberg (2009)
6. Borralleras, C., Rubio, A.: Orderings and Constraints: Theory and Practice of Proving Termination. In: Comon-Lundh, H., Kirchner, C., Kirchner, H. (eds.) Jouannaud Festschrift. LNCS, vol. 4600, pp. 28–43. Springer, Heidelberg (2007)
7. Contejean, E., Marché, C., Tomás, A.-P., Urbain, X.: Mechanically proving termination using polynomial interpretations. Journal of Automated Reasoning 34(4), 325–363 (2006)
8. Dershowitz, N.: Termination of rewriting. Journal of Symbolic Computation 3, 69–115 (1987)
9. Endrullis, J.: Jambox, Automated Termination Proofs For String and Term Rewriting, `http://joerg.endrullis.de/jambox.html`
10. Endrullis, J., Waldmann, J., Zantema, H.: Matrix Interpretations for Proving Termination of Term Rewriting. Journal of Automated Reasoning 40(2-3), 195–220 (2008)
11. Fuhs, C., Navarro-Marset, R., Otto, C., Giesl, J., Lucas, S., Schneider-Kamp, P.: Search Techniques for Rational Polynomial Orders. In: Autexier, S., Campbell, J., Rubio, J., Sorge, V., Suzuki, M., Wiedijk, F. (eds.) AISC 2008, Calculemus 2008, and MKM 2008. LNCS (LNAI), vol. 5144, pp. 109–124. Springer, Heidelberg (2008)
12. Lancaster, P., Tismenetsky, M.: The Theory of Matrices, 2nd edn. With Applications. Academic Press, London (1985)
13. Lucas, S.: MU-TERM: A Tool for Proving Termination of Context-Sensitive Rewriting. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 200–209. Springer, Heidelberg (2004), `http://zenon.dsic.upv.es/muterm`

14. Lucas, S.: On the relative power of polynomials with real, rational, and integer coefficients in proofs of termination of rewriting. Applicable Algebra in Engineering, Communication and Computing 17(1), 49–73 (2006)
15. Lucas, S.: Polynomials over the reals in proofs of termination: from theory to practice. RAIRO Theoretical Informatics and Applications 39(3), 547–586 (2005)
16. Lucas, S.: Practical use of polynomials over the reals in proofs of termination. In: Proc. of PPDP 2007, pp. 39–50. ACM Press, New York (2007)
17. Meyer, C.D.: Matrix Analysis and Applied Linear Algebra. In: Society for Industrial and Applied Mathematics. SIAM, Philadelphia (2000)
18. Zhang, F.: Matrix Theory. Springer, Berlin (1999)

Automated Reasoning and Presentation Support for Formalizing Mathematics in Mizar

Josef Urban[1,⋆] and Geoff Sutcliffe[2]

[1] Radboud University, Nijmegen
[2] University of Miami

Abstract. This paper presents a combination of several automated reasoning and proof presentation tools with the Mizar system for formalization of mathematics. The combination forms an online service called MizAR, similar to the SystemOnTPTP service for first-order automated reasoning. The main differences to SystemOnTPTP are the use of the Mizar language that is oriented towards human mathematicians (rather than the pure first-order logic used in SystemOnTPTP), and setting the service in the context of the large Mizar Mathematical Library of previous theorems, definitions, and proofs (rather than the isolated problems that are solved in SystemOnTPTP). These differences poses new challenges and new opportunities for automated reasoning and for proof presentation tools. This paper describes the overall structure of MizAR, and presents the automated reasoning systems and proof presentation tools that are combined to make MizAR a useful mathematical service.

1 Introduction and Motivation

Formal mathematics, in its interactive and verification aspects, and in the automated reasoning aspect, is becoming increasingly well-known, used, and experimented with [10]. Projects like FlySpeck [9], formal proof of the Four Color Theorem [8], verification of tiny (but real) operating systems [12], and the increased use of verification for software and hardware [7], are stimulating the development of interactive verification tools and interactive theorem provers (ITPs). Linked to this is the development of strong automated theorem proving (ATP) systems, used either independently to solve hard problems in suitable domains [14,17,3], or integrated with interactive tools [15,11,4]. ATP development has also stimulated interesting research in the context of automated reasoning in large theories [16,34,21].

The goal of the work presented here is to make formal mathematics and automated reasoning easily accessible to practitioners in these areas, by putting most of the work into their browsers, and providing a very fast (real-time) server-based experience with a number of ATP, ITP, presentation, and AI tools that work well together. This is important for supporting existing users and attracting new

⋆ Supported by the NWO project "MathWiki a Web-based Collaborative Authoring Environment for Formal Proofs".

users of Mizar, by providing them with an attractive environment for exploring the world of formal reasoning in mathematics. Fast server-based solutions make systems easy to use, to the extent of just "pushing a button" (clicking on a HTML link), rather than having to go through the pains of building an adequate local hardware and software installation, for benefits that might initially not be clear. Server-based solutions are becoming an important part of general computer use, and formal mathematics is no exception. It is not possible to name all the server-based services that already exist for informal mathematics, starting e.g., from the arXiv, Wikipedia, MathOverflow, PolyMath, Wolfram MathWorld, PlanetMath, ProofWiki, the SAGE system for working with CASes, etc.

This paper describes the *Automated Reasoning for Mizar* (MizAR) web service, which combines several automated reasoning and proof presentation tools with the Mizar system for formalization of mathematics, to form a useful mathematical service. MizAR runs in the context of the Mizar Mathematical Library (MML), and uses the Mizar language that is oriented towards human mathematicians. The main inspiration for MizAR is the SystemOnTPTP ATP service [22]. SystemOnTPTP allows users to easily experiment with many first-order ATP systems in a common framework, and provides additional services such as proof presentation with the IDV system [28], discovery of interesting lemmas with the AGInT system [18], and independent proof verification with the GDV verifier [23]. Pieces of the SystemOnTPTP infrastructure also served in the initial implementation of the MizAR web service. SystemOnTPTP's infrastructure is briefly described in Section 2. Section 3 describes the implemented MizAR service, and demonstrates its use. Section 4 considers a number of possible future extensions, and concludes.

2 SystemOnTPTP

The core of SystemOnTPTP is a utility that allows an ATP problem or solution to be easily and quickly submitted in various ways to a range of ATP systems and tools. SystemOnTPTP uses a suite of currently available systems and tools, whose properties (input format, reporting of result status, etc) are stored in a simple text database. The input can be selected from the TPTP (Thousands of Problems for Theorem Provers) problem library or the TSTP (Thousands of Solutions from Theorem Provers) solution library [24], or provided in TPTP format [25] by the user. The implementation relies on several subsidiary tools to preprocess the input, control the execution of the chosen ATP system(s), and postprocess the output. On the input side TPTP2X or TPTP4X is used to prepare the input for processing. A strict resource limiting program called `TreeLimitedRun` is used to limit the CPU time and memory used by an ATP system or tool. `TreeLimitedRun` monitors processes' resource usage more tightly than is possible with standard operating system calls. Finally a program called `X2tptp` converts an ATP system's output to TPTP format, if requested by the user.

The web interfaces SystemB4TPTP, SystemOnTPTP, and SystemOnTSTP provide interactive online access to the SystemOnTPTP utility.[1] The online service

[1] Available starting at `http://www.tptp.org/cgi-bin/SystemOnTPTP`

can also be accessed directly with `http POST` requests. The SystemB4TPTP interface provides access to tools for preparing problems for submission to an ATP system, including conversion from other (non-TPTP) formats to TPTP format, parsing and syntax checking, type checking, and pretty printing. In addition to providing access to ATP systems, the SystemOnTPTP interface additionally provides system reports, recommendations for systems to use on a given problem, and direct access to the SSCPA system [26] that runs multiple systems in competition parallel. The SystemOnTSTP interface provides access to solution processing tools, including parsing and syntax checking, pretty printing, derivation verification using GDV [23], interactive graphical proof presentation using IDV [28], answer extraction [27], and proof conversion and summarization tools. The three interfaces have options to pass the output from a system/tool on to the next interface – from problem preparation, to problem solving, to solution processing. The output is returned to browsers in appropriate HTML wrapping, and can also be obtained in its raw form for processing on the client side (typically when the interfaces are called programmatically using `http POST` requests). The online service is hosted at the University of Miami on a server with four 2.33GHz CPUs, 4GB RAM, and running the Linux 2.6 operating system.

3 MizAR

MizAR is running experimentally on our server[2], where it can be best learned by exploration. A good way to explore is to start with an existing simple Mizar article, e.g., the `card_1` article[3] about cardinal numbers [1][4], from the MML.[5] Within MizAR, select the "URL to fetch article from" field, insert the article's URL into the text box, and press the "Send" button. For experienced Mizar users, there is also a simple way to send the current Mizar buffer to the remote service, by running the `mizar-post-to-ar4mizar` interactive function in the Mizar mode for Emacs [30]. Both actions call the main `MizAR cgi-bin` script with appropriate arguments, which launches the functions described below.

MizAR links together a number of Mizar and ATP-related components, which are useful for general work with formal mathematics in Mizar. The main components are as follows (details are provided in the rest of Section 3):

– Web access to the whole cross-linked HTMLized MML.
– Fast server-based verification of a Mizar article.
– Disambiguation of the article by HTMLization, producing an HTML presentation of the verified article with links to an HTMLized version of the MML. Additional useful information is also extracted during HTMLization, and included in the HTML presentation of the article.
– Fast translation of the article to MPTP (Mizar Problems for Theorem Provers) format.

[2] `http://mws.cs.ru.nl/~mptp/MizAR.html`
[3] `http://mws.cs.ru.nl/~mptp/mml/mml/card_1.miz`
[4] Available at `http://mizar.uwb.edu.pl/JFM/Vol1/ordinal1.html`
[5] All articles are available from `http://mws.cs.ru.nl/~mptp/mml/mml`

– Fast generation of ATP problems in TPTP format, for all the theorems in the article, and for all the atomic inferences done by Mizar.
– Easy access to default ATP systems for solving the ATP problems, and access to SystemOnTPTP for solving more difficult problems.
– Easy access to IDV for visualization and postprocessing of proofs found by the ATP systems.
– Suggesting useful hints for proving (either by ATP or interactively in Mizar) particular Mizar lemmas and theorems.

Figure 1 shows the overall structure of the MizAR system. The leftmost column shows the various forms of the article that are produced, and the two bold boxes in the next column are the HTML presentations for user interaction. The third column shows the software tools that generate the various dataforms, using the article and the background information shown in the rightmost column. A Mizar article is submitted through the web interface or from Emacs. The article is then verified and converted to XML format, which is subsequently rendered in HTML format with links to the MML. The HTML presentation includes links that allow the user to proceed with further processing, and is the main interface for user interaction with MizAR. While the HTML is presented to the user, the article is asynchronously converted to the MPTP format, which is used for generating TPTP format ATP problems. The ATP problems can then be submitted to ATP systems, either locally via SystemOnTPTP. The ATP systems' solutions are used to enrich the HTML presentation, and can be passed on to various post-processing tools. The subcomponents that perform these tasks are described in more detail below.

3.1 Server-Based Verification of a Mizar Article

Unlike many other (especially LCF-inspired) proof assistants, Mizar is a compiler-like batch processor, verifying a whole article in one pass. While a lot of work on Mizar goes into balancing the strength, speed, and obviousness of the proof checking, the process of checking a whole article can get quite time-consuming for longer and more complex articles, especially on older hardware.

There are several advantages to remote server-based verification of Mizar articles. The first obvious advantage is that having everything web-based removes the need for a local installation of Mizar. The second advantage is that even if Mizar is installed locally, it is often more convenient to quickly (try to) verify an article in a browser, instead of launching the verification environment on one's local computer. In cases when the article is available online, it is possible to provide that URL as an argument to the MizAR URL, to directly launch MizAR on the article.[6] This makes such verification snippets available in all kinds of online fora (e-mail discussions, blog and twitter posts, wikis, etc.) with direct rendering of the results. In short, the third advantage is that a web service provides mechanism for online communication of verification results.

[6] For example,
`http://mws.cs.ru.nl/ mptp/cgi-bin/MizAR.cgi?ProblemSource=URL&`
`FormulaURL=http://mws.cs.ru.nl/ mptp/mml/mml/card_1.miz&Name=Test1`

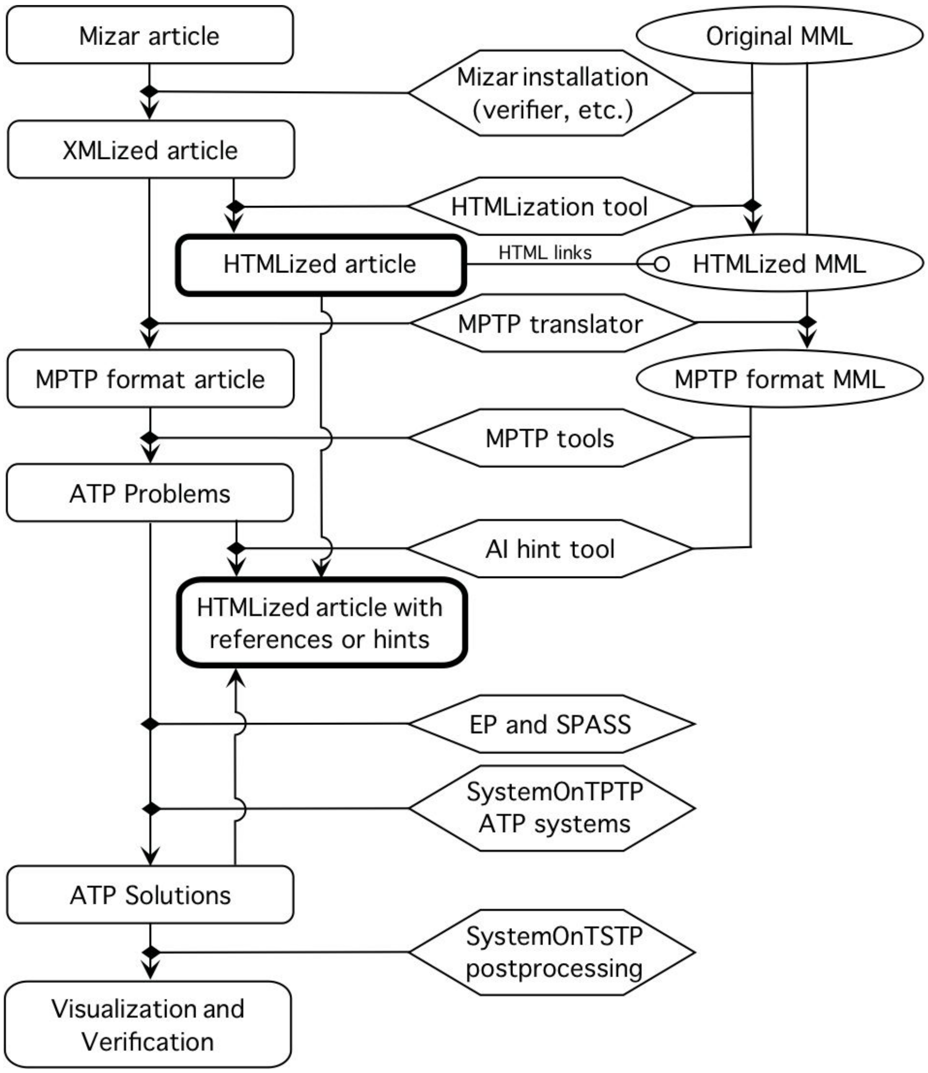


Fig. 1. Structure of the MizAR system

The fourth (and probably greatest) advantage of server-based verification is the raw verification speed. A dedicated server usually runs on reasonably new hardware with enough memory, etc. For example, even for the relatively short `card_1` Mizar article mentioned above, full verification on a recent notebook (1.66GHz Intel Atom) takes 2s, while on a recent lower-range server (2 quad-core hyperthreading 2.27GHz Intel Xeons) the same task takes 0.5s. For a more involved article this difference becomes more visible, and can be the deciding factor for the usability and real-time experience with the system. For example, for the more involved Mizar article `fdiff_1`[7] about real function differentiability [19], the difference is 23s vs. 6s.

The latest advances in CPU power have been achieved mainly by packing multiple CPUs together, instead of raising the speed of individual CPUs. To take advantage of this, Mizar processing has recently been parallelized[8], and the parallel version of the Mizar verifier is running on our server. This (depending on the success of the parallelization) can further significantly improve the real-time verification experience. For example, on the even longer Mizar article `fdiff_2`[9] about real function differentiability [13], the difference between running the parallel version (using eight cores) and the non-parallel version is a factor of four (31s vs. 7.8s). Verification of this article using the above mentioned notebook takes 125s, resulting in a speed-up factor of sixteen (and substantially improving the real-time interaction with the system).

The last important advantage is that a server-based installation supports use of modified, enhanced, and experimental versions of the verifier. This can provide useful additional functionalities. For instance, the Mizar parallelizer requires additional software to run, and a recently modified version of the Mizar verifier that has not yet been distributed to Mizar users. Translation of Mizar articles to ATP formats also requires a version of the verifier that has been compiled with a special flag, again not included in the standard Mizar distribution. An online service can also easily include multiple versions of the Mizar library and binaries, as is done for the MML Query service [2].

3.2 HTMLization of Mizar Articles

There has been quite a lot of recent work on XMLization and HTMLization of Mizar articles [29,35], including the addition of useful additional information into the XML form of the Mizar article and its HTML presentation. There are two major reasons for having a static HTMLized MML available:[10] (i) to provide fast browsing of the theorems and definitions used in a particular formalization, with a number of user-friendly features (like (sub)proof hiding/showing, etc.), and (ii) providing explanations for a number of phenomena in the formalization that are made explicit and clear only during verification, and are hard to decipher from the formalization text alone. The latter includes, for example:

– Explicit HTML presentation of the current goal (thesis), computed by the verifier at each point of the formalization.
– Proper disambiguation of overloaded mathematical symbols. Overloading is necessary in a large body of mathematics including all kinds of subfields, but at the same time makes it difficult for readers of the textual versions of the articles to understand the precise meaning of the overloaded symbols.

[7] `http://mws.cs.ru.nl/~mptp/mml/mml/fdiff_1.miz`

[8] The description of the Mizar parallelization and related experiments is unpublished as of January 2010. The parallelizer is available at `http://github.com/JUrban/MPTP2/raw/master/MizAR/cgi-bin/bin/mizp.pl`

[9] `http://mws.cs.ru.nl/~mptp/mml/mml/fdiff_2.miz`

[10] It is available at `http://mws.cs.ru.nl/~mptp/mml/html/`

– Explicit access to formulae for definition correctness, and formulae expressing properties (projectivity, antisymmetry, etc.) that are computed by the verifier. Making these explicit in the HTML presentation can help users.
– Explicit representation of other features that are implicit in Mizar verification, e.g., definitional expansions, original versions of constructors that have been redefined, etc. Making these explicit in the HTML presentation can also help users.

The static HTMLized MML is an important resource used by MizAR. The articles submitted to MizAR are dynamically linked to the static HTMLized MML. This is a notable difference to SystemOnTPTP, which treats each problem as an independent entity. The first implementation of MizAR has focused on developing the services for a fixed version of the MML. However, management of library versions is a nontrivial and interesting problem. Allowing users to verify new articles and also refactor existing ones will ultimately lead into the area of formal mathematical wikis [6,5].

The main functions of the HTMLization service are to (i) provide quick linking to the static HTMLized MML (thus providing the disambiguation and explanation functions described above), and (ii) allow a number of additional (mainly automated reasoning and AI) services to be launched by suitable CGI and AJAX calls from links in the HTML. These additional services are described in the following subsections. The main features of server-side HTMLization are increased speed, and the availability of additional programs and features. While the Mizar HTML processing was designed to be locally available, using just a browser-based XSL processor (i.e., loading the XML produced by Mizar directly into a browser, which applies the appropriate style sheet), even the basic XSL processing in the browser can take a long time (minutes). Again, having a specialized fast XSL processor installed on the server helps quite a lot, and the HTMLization can be parallelized using techniques similar to the parallelization of the basic verification process. This provides a much better HTMLization response, and also makes additional XSL-based preprocessing possible. This is needed for better HTML quality, and for the translation to ATP formats.

3.3 Generation of ATP Problems in TPTP Format

One of the main objectives of MizAR (as suggested by its name) is to allow easy experimentation with ATP systems over the large body of formal mathematics available in the MML, and to apply ATP functionalities on Mizar articles. The MPTP system [32,31] for translating Mizar articles to the TPTP format has been modified to work in a fast real-time mode, generating ATP problems corresponding to Mizar proof obligations. The MPTP system translates Mizar articles to the MPTP format, which is an extension of the TPTP format with information needed for further processing into ATP problems. Like the static HTMLized MML, a static copy of the MML in MPTP format is available to MizAR. It is used for building translated MML items (theorems, definitions, formulae encoding Mizar type automations, etc.) that are necessary for creating complete ATP problems.

Using MPTP and generating ATP problems requires a quite complex installation and setup (SWI Prolog, Unix, special XSL style sheets, the translated MML in the MPTP format, etc.), so this is a good example of an additional functionality that would be quite hard to provide locally. The MPTP was initially designed for offline production of interesting ATP problems and data, and was not optimized for speed. Several techniques have been used to provide a reasonable real-time experience:

– More advanced (graph-like) data structures have been used to speed up the selection of parts of the MML necessary for generating the ATP problems.
– Larger use has been made of Prolog indexing and the asserted database, for various critical parts of the code.
– The MPTP version of the MML has been factored so that it is possible to work with only the parts of the MML needed for a given article.

These techniques have led to reasonable real-time performance of the MPTP problem generation, comparable to the performance of Mizar verification and HTMLization. For example, the MPTP processing followed by the generation of all 586 ATP problems for the `card_1` article takes 7s on the server.

After the conversion to MPTP format, the ATP problems for an article are generated asynchronously while the user is presented with the HTMLized article. There is a small danger of the user wanting to solve an ATP problem that has not yet been generated. However, it is easy to check if the translation process is finished, and which ATP problems are already available, by examining the system log.

The MPTP processing has not been parallelized (like the Mizar verification and HTMLization). However, there are no serious obstacles to that. Another speed-up option would be to ask MPTP to generate only a subset of the ATP problems (this is actually how the parallelization is going to work), selected in some reasonable way by the formalizer in the user interface. It is also possible to keep the MPTP system loaded and listening once it has generated a subset of problems, and to have it generate new ATP problems from the current article on demand.

The HTMLization of an article and the generation of ATP problems are independent processes that could be separated into two separate services. Users might, for instance, be interested only in HTML-like disambiguation of their articles, or only in getting explanations and advice from ATP systems, without looking at the HTML form of the article. With sufficient CPU-cores in the server, none of these two possible use-cases suffers in terms of the response time.

3.4 Calling ATP Systems

The calling of ATP systems to solve the ATP problems is built into the HTML presentation of the user's article, by linking the available ATP services to keywords in the HTML presentation. This follows the general idea that the HTML serves as the main interface for calling other services. The links to the ATP services in the HTML are the Mizar keywords **by** and **from**, indicating semantic justification in Mizar. For example, the Mizar justification

```
thus ( f is one-to-one & dom f = X & rng f = A )
   by A1, A4, WELLORD2:25, WELLORD2:def 1;
```

in the last line of the proof of theorem `Th4` in the `card_1` article says that the Mizar checker should be able to verify that the formula on the left hand side of the **by** keyword follows from the previously stated local propositions, theorems and definitions `A1, A4, WELLORD2:25, WELLORD2:def 1`, and some knowledge that the Mizar verifier uses implicitly. There are now the following use-cases calling ATP systems:

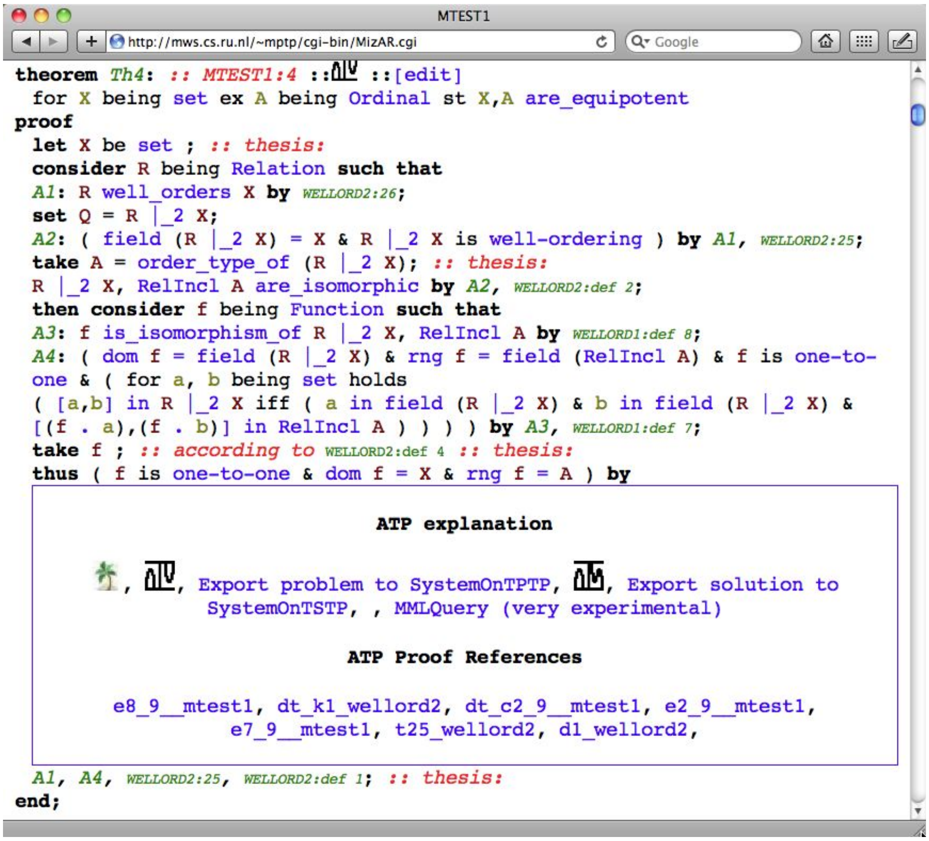
1. Mizar has verified the inference, possibly using some implicit information. The user is interested in knowing exactly what implicit information was used by Mizar, and exactly how the proof was conducted.
2. Mizar has not verified the inference. The user is interested in knowing if the inference is logically valid, if it can be proved by a (stronger) ATP system, and what such an ATP proof looks like.

The first use-case typically happens for one of two reasons. The first reason is that the theory in which the author is working has become very rich, and involves many implicit (typically typing) Mizar mechanisms that make the formal text hard to understand. The TPTP translation has to make all this implicit information explicit in the TPTP problems, and the resulting corresponding ATP proofs show explicitly how this information is used. For the Mizar justification above, clicking on the **by** keyword calls the EP system [20] on the ATP problem, with a several second time limit. If a proof is found, the interface is refreshed with an explanation box that includes (among other things described below) a list of the references used in the proof, as shown in Figure 2. In this case the exact references shown to the user are following:

```
e8_9__mtest1, dt_k1_wellord2, dt_c2_9__mtest1, e2_9__mtest1,
e7_9__mtest1, t25_wellord2, d1_wellord2
```

These references use the MPTP syntax, but are linked (dynamically using AJAX calls) to the corresponding places in the theorem's HTML or the static HTMLized MML), and are given appropriate explanation titles. Note that the EP proof uses seven more references than the four that are in the original Mizar **by** inference. One reference is added because it explicitly denotes the formula being proved (the left-hand side of **by**), and the two remaining references encode implicit type declarations that are used by Mizar (the type of the local constant `R`, and the type of the functor `RelIncl` that is used in proposition `A4` (renamed to `e7_9__mtest1` by MPTP)). The ATP proof can be visualized in the IDV system by clicking on the palm tree icon in the explanation box.

The second reason for the first use-case is cross-verification. In cases when a bug in the Mizar implementation is suspected, or incompleteness in the ATP translation is suspected, the user may be interested in knowing if the Mizar proof can be done by another system (and how). In this sense the environment is used for gathering additional information and debugging. The cross-verification rates for Mizar justifications are reasonably high [33], which makes this usage realistic.



```
theorem Th4: :: MTEST1:4 :: ::[edit]
  for X being set ex A being Ordinal st X,A are_equipotent
proof
  let X be set ; :: thesis:
  consider R being Relation such that
  A1: R well_orders X by WELLORD2:26;
  set Q = R |_2 X;
  A2: ( field (R |_2 X) = X & R |_2 X is well-ordering ) by A1, WELLORD2:25;
  take A = order_type_of (R |_2 X); :: thesis:
  R |_2 X, RelIncl A are_isomorphic by A2, WELLORD2:def 2;
  then consider f being Function such that
  A3: f is_isomorphism_of R |_2 X, RelIncl A by WELLORD1:def 8;
  A4: ( dom f = field (R |_2 X) & rng f = field (RelIncl A) & f is one-to-
  one & ( for a, b being set holds
  ( [a,b] in R |_2 X iff ( a in field (R |_2 X) & b in field (R |_2 X) &
  [(f . a),(f . b)] in RelIncl A ) ) ) ) by A3, WELLORD1:def 7;
  take f ; :: according to WELLORD2:def 4 :: thesis:
  thus ( f is one-to-one & dom f = X & rng f = A ) by

                          ATP explanation

      , , Export problem to SystemOnTPTP, , Export solution to
           SystemOnTSTP, , MMLQuery (very experimental)

                         ATP Proof References

     e8_9__mtest1, dt_k1_wellord2, dt_c2_9__mtest1, e2_9__mtest1,
          e7_9__mtest1, t25_wellord2, d1_wellord2,

  A1, A4, WELLORD2:25, WELLORD2:def 1; :: thesis:
end;
```

Fig. 2. ATP explanation box

The second use-case (finding proofs that are too hard for Mizar) is the real "ATP proof assistance" dream, i.e., using ATP systems to automatically find proofs for ITPs. Users can do this within MizAR by providing a large set of "potentially relevant" Mizar propositions on the right-hand side of the **by** keyword, and letting the EP system try to find a proof. Note that if EP does not find the problem to be countersatisfiable, the user also has the option to try the SPASS ATP system [36] directly from the interface, as shown in Figure 3. This is justified by the general experience that SPASS is reasonably complementary to EP when solving MPTP problems. If SPASS is not successful the user can use the links and icons in the explanation box to inspect the ATP problem, and launch the SystemOnTPTP interface to try the ATP systems available there. The proofs found by the ATP systems can be processed in the SystemOnTSTP interface, including visualization using the IDV system, analysis using the AGInT system for finding the interesting steps in proofs, and ATP-based cross-verification using the GDV verifier.

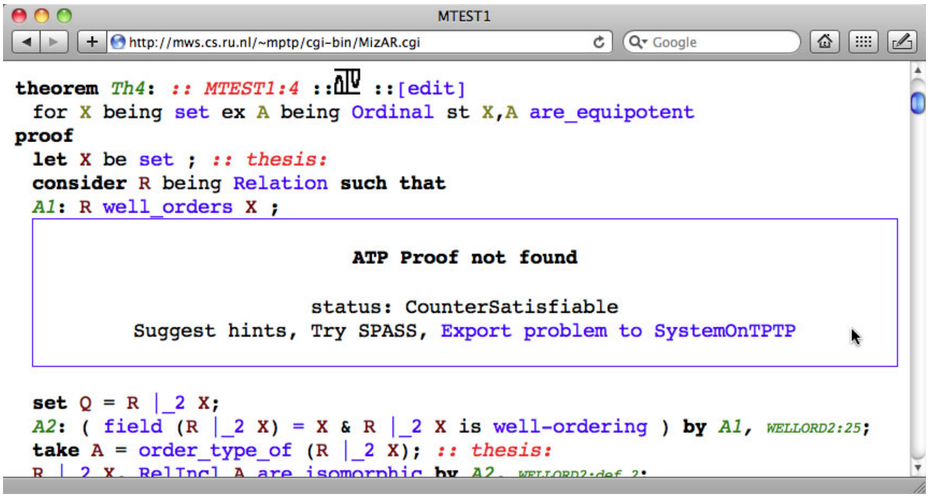

```
theorem Th4: :: MTEST1:4 :: ::[edit]
  for X being set ex A being Ordinal st X,A are_equipotent
proof
  let X be set ; :: thesis:
  consider R being Relation such that
  A1: R well_orders X ;

                         ATP Proof not found

                     status: CounterSatisfiable
          Suggest hints, Try SPASS, Export problem to SystemOnTPTP

  set Q = R |_2 X;
  A2: ( field (R |_2 X) = X & R |_2 X is well-ordering ) by A1, WELLORD2:25;
  take A = order_type_of (R |_2 X); :: thesis:
  R |_2 X, RelIncl A are isomorphic by A2, WELLORD2:def 2;
```

Fig. 3. ATP explanation box for “Proof not found”

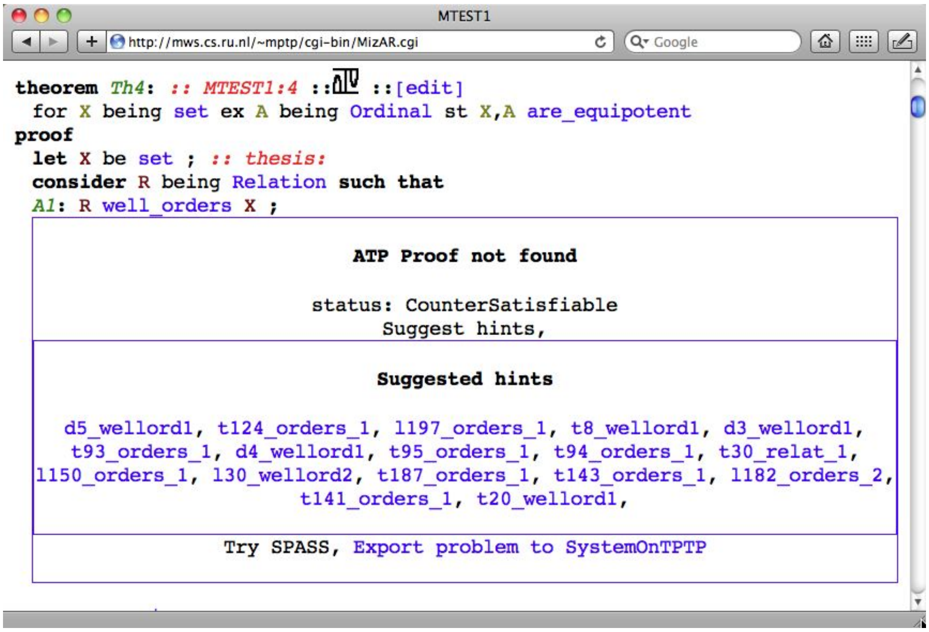

```
theorem Th4: :: MTEST1:4 :: ::[edit]
  for X being set ex A being Ordinal st X,A are_equipotent
proof
  let X be set ; :: thesis:
  consider R being Relation such that
  A1: R well_orders X ;

                         ATP Proof not found

                     status: CounterSatisfiable
                            Suggest hints,

                           Suggested hints

    d5_wellord1, t124_orders_1, l197_orders_1, t8_wellord1, d3_wellord1,
     t93_orders_1, d4_wellord1, t95_orders_1, t94_orders_1, t30_relat_1,
l150_orders_1, l30_wellord2, t187_orders_1, t143_orders_1, l182_orders_2,
                     t141_orders_1, t20_wellord1,

                Try SPASS, Export problem to SystemOnTPTP
```

Fig. 4. ATP explanation box offering hints

3.5 Getting Hints for Necessary Mizar References

If none of the ATP systems can find a proof for an ATP problem (corresponding to a Mizar inference), either because the ATP system timed out or found that the ATP problem is countersatisfiable (as in Figure 3), then typically some more assumptions (Mizar references) have to be added to the TPTP problem. The explanation box can provide hints for proving the Mizar proposition. This is done using the "Suggest hints" link that is put into the box when EP fails to find a proof (see Figure 3). The "Suggest hints" button is linked to a Bayesian advisor that has been trained on the whole MML (i.e., on all of the proofs in it). (See [34] for the details of how the machine learning is organized in the context of a large deductive repository like MML. Other axiom selection systems could be used in a similar way.) The trained advisor runs as a daemon on the web server, and receives queries initiated by clicking on the "Suggest hints" button. This service is very fast, and the hints are usually provided in milliseconds. They are HTMLized and inserted (by AJAX calls) into the explanation box, as shown in Figure 4.

4 Future Work and Conclusions

This paper has introduced the MizAR web service that allows Mizar users to use automated reasoning tools on their Mizar articles. MizAR is to some degree based on and similar to the SystemOnTPTP service for solving first-order ATP problems. The main differences to SystemOnTPTP are the use of the Mizar language that is oriented towards human mathematicians (rather than the pure first-order logic used in SystemOnTPTP), and setting the service in the context of the large Mizar Mathematical Library of previous theorems, definitions, and proofs.

There are obvious differences to those systems, given by the large-theory setting in which Mizar formalization is typically done. There are several use-cases described above, ranging from using HTMLization to disambiguate complicated Mizar syntax, usage of ATP systems to explain Mizar inferences, provide new proofs, and find counterexamples, to using additional AI-based services for proof advice, like the proof advisor trained on the whole MML.

There are many directions for future work in this setting, some of them mentioned above. The service already is available by an interactive call from the Emacs interface. However, Emacs simply generates the request from the current Mizar buffer, and lets the user see the response (and all the associated functionalities) in a browser. Obviously, the ATP and proof advising functionalities could be made completely separate from the HTML presentation, and sent directly to the Emacs session.

As mentioned above, the static MML is now present on the server in both HTML and MPTP format (and obviously in raw text and in the Mizar internal format), but not directly editable by the users. Giving the user the ability to edit the supporting MML forms leads in the direction of formal mathematical wikis, with all the interesting persistence, versioning, linking, user-authentication, and dependency problems to solve. There is an experimental ikiwiki-based prototype available for Mizar and the MML, which solves some of the persistence and user-authentication problems, and that is likely to be merged with the services presented here. Hopefully this will form a rich wiki for formal mathematics, with a large number of services providing interesting additional functionalities to people interested in formal mathematics.

There is also a large amount of work that can be done on making the system nicer and more responsive, e.g., the parallelization of the MPTP processing is yet to be done, better machine learning and hint suggestion methods can be used and linked to the ATP services, and presentations in formats other than HTML (e.g., TeX and PDF are also used for Mizar) would be nice to include.

References

1. Bancerek, G.: The Ordinal Numbers. Journal of Formalized Mathematics 1(1), 91–96 (1990)
2. Bancerek, G., Rudnicki, P.: Information Retrieval in MML. In: Asperti, A., Buchberger, B., Davenport, J.H. (eds.) MKM 2003. LNCS, vol. 2594, pp. 119–132. Springer, Heidelberg (2003)
3. Benzmüller, C., Paulson, L.: Multimodal and Intuitionistic Logics in Simple Type Theory. Logic Journal of the IGPL (2010)
4. Conchon, S., Contejean, E., Kanig, J., Lescuyer, S.: Lightweight Integration of the Ergo Theorem Prover Inside a Proof Assistant. In: Rushby, J., Shankar, N. (eds.) Proceedings of the 2nd Workshop on Automated Formal Methods, pp. 55–59. ACM Press, New York (2007)
5. Corbineau, P., Geuvers, H., Kaliszyk, C., McKinna, J., Wiedijk, F.: A Real Semantic Web for Mathematics Deserves a Real Semantics. In: Lange, C., Schaffert, S., Skaf-Molli, H., Völkel, M. (eds.) Proceedings of the 3rd Semantic Wiki Workshop. CEUR Workshop Proceedings, vol. 360, pp. 62–66 (2008)
6. Corbineau, P., Kaliszyk, C.: Cooperative Repositories for Formal Proofs. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, pp. 221–234. Springer, Heidelberg (2007)
7. D'Silva, V., Kroening, D., Weissenbacher, G.: A Survey of Automated Techniques for Formal Software Verification. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems 27(7), 1165–1178 (2008)
8. Gonthier, G.: Formal Proof - The Four-Color Theorem. Notices of the American Mathematical Society 55(11), 1382–1393 (2008)
9. Hales, T.: A Proof of the Kepler Conjecture. Annals of Mathematics 162(3), 1065–1185 (2005)
10. Hales, T. (ed.): A Special Issue on Formal Proof, vol. 55 (2008)
11. Hurd, J.: First-Order Proof Tactics in Higher-Order Logic Theorem Provers. In: Archer, M., Di Vito, B., Munoz, C. (eds.) Proceedings of the 1st International Workshop on Design and Application of Strategies/Tactics in Higher Order Logics. number NASA/CP-2003-212448 NASA Technical Reports, pp. 56–68 (2003)
12. Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: Formal Verification of an OS Kernel. In: Anderson, T. (ed.) Proceedings of the 22nd ACM Symposium on Operating Systems Principles, pp. 207–220. ACM Press, New York (2009)
13. Kotowicz, J., Raczkowski, K.: Real Function Differentiability - Part II. Formalized Mathematics 2(3), 407–411 (1991)
14. McCune, W.W.: Solution of the Robbins Problem. Journal of Automated Reasoning 19(3), 263–276 (1997)

15. Paulson, L.: A Generic Tableau Prover and its Integration with Isabelle. Artificial Intelligence 5(3), 73–87 (1999)
16. Pease, A., Sutcliffe, G.: First Order Reasoning on a Large Ontology. In: Urban, J., Sutcliffe, G., Schulz, S. (eds.) Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories. CEUR Workshop Proceedings, vol. 257, pp. 59–69 (2007)
17. Phillips, J.D., Stanovsky, D.: Automated Theorem Proving in Loop Theory. In: Sutcliffe, G., Colton, S., Schulz, S. (eds.) Proceedings of the CICM Workshop on Empirically Successful Automated Reasoning in Mathematics. CEUR Workshop Proceedings, vol. 378, pp. 42–53 (2008)
18. Puzis, Y., Gao, Y., Sutcliffe, G.: Automated Generation of Interesting Theorems. In: Sutcliffe, G., Goebel, R. (eds.) Proceedings of the 19th International FLAIRS Conference, pp. 49–54. AAAI Press, Menlo Park (2006)
19. Raczkowski, K., Sadowski, P.: Real Function Differentiability. Formalized Mathematics 1(4), 797–801 (1990)
20. Schulz, S.: E: A Brainiac Theorem Prover. AI Communications 15(2-3), 111–126 (2002)
21. Suda, M., Sutcliffe, G., Wischnewski, P., Lamotte-Schubert, M., de Melo, G.: External Sources of Axioms in Automated Theorem Proving. In: Mertsching, B., Hund, M., Aziz, Z. (eds.) KI 2009. LNCS (LNAI), vol. 5803, pp. 281–288. Springer, Heidelberg (2009)
22. Sutcliffe, G.: SystemOnTPTP. In: McAllester, D. (ed.) CADE 2000. LNCS (LNAI), vol. 1831, pp. 406–410. Springer, Heidelberg (2000)
23. Sutcliffe, G.: Semantic Derivation Verification. International Journal on Artificial Intelligence Tools 15(6), 1053–1070 (2006)
24. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0. Journal of Automated Reasoning 43(4), 337–362 (2009)
25. Sutcliffe, G., Schulz, S., Claessen, K., Van Gelder, A.: Using the TPTP Language for Writing Derivations and Finite Interpretations. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 67–81. Springer, Heidelberg (2006)
26. Sutcliffe, G., Seyfang, D.: Smart Selective Competition Parallelism ATP. In: Kumar, A., Russell, I. (eds.) Proceedings of the 12th International FLAIRS Conference, pp. 341–345. AAAI Press, Menlo Park (1999)
27. Sutcliffe, G., Yerikalapudi, A., Trac, S.: Multiple Answer Extraction for Question Answering with Automated Theorem Proving Systems. In: Guesgen, H., Lane, C. (eds.) Proceedings of the 22nd International FLAIRS Conference, pp. 105–110. AAAI Press, Menlo Park (2009)
28. Trac, S., Puzis, Y., Sutcliffe, G.: An Interactive Derivation Viewer. In: Autexier, S., Benzmüller, C. (eds.) Proceedings of the 7th Workshop on User Interfaces for Theorem Provers, 3rd International Joint Conference on Automated Reasoning. Electronic Notes in Theoretical Computer Science, vol. 174, pp. 109–123 (2006)
29. Urban, J.: XML-izing Mizar: Making Semantic Processing and Presentaion of MML Easy. In: Kohlhase, M. (ed.) MKM 2005. LNCS (LNAI), vol. 3863, pp. 346–360. Springer, Heidelberg (2006)
30. Urban, J.: MizarMode - An Integrated Proof Assistance Tool for the Mizar Way of Formalizing Mathematics. Journal of Applied Logic 4(4), 414–427 (2006)
31. Urban, J.: MPTP 0.2: Design, Implementation, and Initial Experiments. Journal of Automated Reasoning 37(1-2), 21–43 (2006)

32. Urban, J.: Automated Reasoning for Mizar: Artificial Intelligence through Knowledge Exchange. In: Sutcliffe, G., Rudnicki, P., Schmidt, R., Konev, B., Schulz, S. (eds.) Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and The 7th International Workshop on the Implementation of Logics. CEUR Workshop Proceedings, vol. 418, pp. 1–16 (2008)
33. Urban, J., Sutcliffe, G.: ATP-based Cross Verification of Mizar Proofs: Method, Systems, and First Experiments. Journal of Mathematics in Computer Science 2(2), 231–251 (2009)
34. Urban, J., Sutcliffe, G., Pudlak, P., Vyskocil, J.: MaLARea SG1: Machine Learner for Automated Reasoning with Semantic Guidance. In: Baumgartner, P., Armando, A., Gilles, D. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 441–456. Springer, Heidelberg (2008)
35. Urban, J., Sutcliffe, G., Trac, S., Puzis, Y.: Combining Mizar and TPTP Semantic Presentation and Verification Tools. Studies in Logic, Grammar and Rhetoric 18(31), 121–136 (2009)
36. Weidenbach, C., Schmidt, R., Hillenbrand, T., Rusev, R., Topic, D.: SPASS Version 3.0. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 514–520. Springer, Heidelberg (2007)

Some Considerations on the Usability of Interactive Provers

Andrea Asperti and Claudio Sacerdoti Coen

Department of Computer Science
University of Bologna
{asperti,sacerdot}@cs.unibo.it

Abstract. In spite of the remarkable achievements recently obtained in the field of mechanization of formal reasoning, the overall usability of interactive provers does not seem to be sensibly improved since the advent of the "second generation" of systems, in the mid of the eighties. We try to analyze the reasons of such a slow progress, pointing out the main problems and suggesting some possible research directions.

1 Introduction

In [23], Wiedijk presented a modern re-implementation of DeBruijn's Automath checker from the seventies (see [16]). The program was written to restore a damaged version of Jutting's translation of Landau's Grundlagen [20], and the interest of this development is that it is one of the first examples of a large piece of mathematics ever formalized and checked by a machine. In particular, it looks like a good touchstone to reason about the progress made in the field of computer assisted reasoning during the last 30/40 years.

From this respect, the only concrete measure offered by Wiedijk is the compilation time, that passed from 35 minutes of the seventies to the 0.6 seconds of his new system. Of course, this is largely justified by the better performances of microprocessors, and such a small compilation time does only testify, at present, of a substantial underuse of the machine potentialities. As observed by Wiedijk himself, "the user's time is much more valuable than the computer's time", and the interesting question would be to know what a modern system could do for us supposing to grant him 35 minutes, as in the seventies.

A different measure that is sometimes used to compare formalizations is the so called *de Bruijn factor* [21]. This is defined as the quotient between the dimension of the formalization and the dimension of the source mathematical text (sometimes computed on compressed files), and it is supposed to give evidence of the *verbosity*, and hence of the *additional complexity* of the formal encoding. In the case of van Benthem Jutting's work, Wiedijk computed a de Bruijn factor of 3.9 (resp. 3.7 on compressed files). For other formalizations that are investigated in [21], sensibly more recent than the Automath effort, the de Bruijn factor lies around 4. On even more recent works, some authors point out even higher factors (8 and more) [4,2,15].

A more explicit indicator for measuring the progress of the field is the average amount of time required to formalize a given quantity of text (a page, say). The table in Figure 1 reports some of these figures, computed by different people on different mathematical sources and using different systems.

source	**formalization cost** (weeks per page)
Van Benthem [20]	1
Wiedijk [22]	1.5
Hales [12]	1
Asperti [2]	1.5

Fig. 1. Formalization cost

In the case of Van Benthem Jutting's work, the cost factor is easily estimated: the Grundlagen are 161 pages long, and he worked at their formalization for - say - three years during his PhD studies (the PhD program takes four years in Netherlands). Wiedijk [22] computes a formalization cost of 2.5 man-years per megabyte of *target* (formalized) information. Since, according to his own figures, a page in a typical mathematical textbook is about 3 kilobytes of text, and considering a de Bruijn factor of 4, we easily get the value in Figure 1: $3 \cdot 4 \cdot 2.5 \cdot 10^{-3} \cdot 52 \approx 1.5$. In [2], very detailed timesheets were taken during the development, precisely in order to compute the cost factor with some accuracy. Hales [12] just says that his figure is a *standard benchmark*, without offering any source or reference (but it presumably fits with his own personal experience).

Neither the de Bruijn nor the cost factor seem to have progressed over the years; on the contrary, they show a slight worsening. Of course, as it is always the case, we can give opposite interpretations of this fact. The optimistic interpretation is that it is true that the factors are constant, but the mathematics we are currently able to deal with has become much more complex: so, keeping low cost and de Bruijn factors is already a clear sign of progress. It is a matter of fact that the mathematics of the Grundlagen is not very complex, and that remarkable achievements have been recently obtained in the field of interactive theorem proving, permitting the formalization and automatic verification of complex mathematical results such as the asymptotic distribution of prime numbers (both in its elementary [4] and analytic [15] versions), the four color theorem [8,9] or the Jordan curve theorem [13]; similar achievements have been also obtained in the field of automatic verification of software (see e.g. [1] for a discussion). However, it is also true that these accomplishments can be justified in many other different ways, quite independent from the improvements of systems: a) the already mentioned progress of hardware, both in time and memory space; b) the enlarged communities of users; c) the development of good and sufficiently stable libraries of formal mathematics; d) the investigation and understanding of formalization problems and the development of techniques and methodologies for addressing them e) the growing confidence in the potentialities of interactive provers; f) the possibility to get suitable resources and funding.

The general impression is that, in spite of many small undeniable technical improvements, the overall usability of interactive provers has not sensibly improved over the last 25 years, since the advent of the current "second generation" of systems[1]: Coq, Hol, Isabelle, PVS (see [10,14,11,7] for some interesting historical surveys). This is certainly also due, in part, to backward compatibility issues:

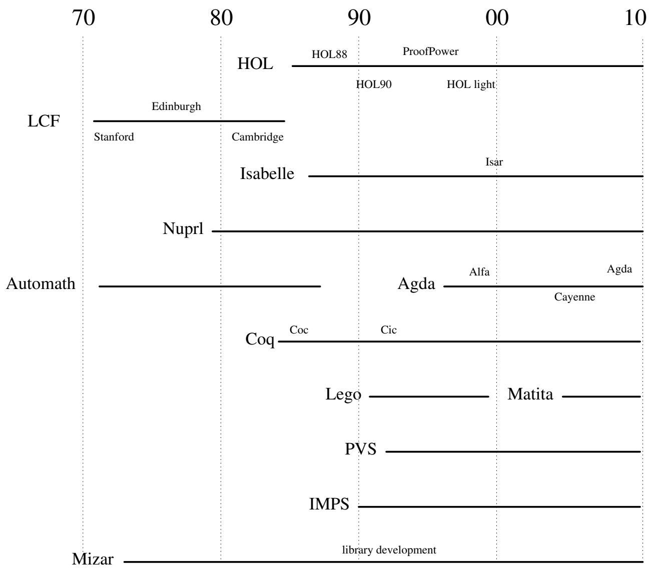


Fig. 2. Rise and fall of Interactive Provers

the existence of a large library of available results and a wide community of users obviously tends to discourage wide modifications. Worse than that, it is usually difficult to get a sensible feedback from users: most of them passively accept the system as they could accept a programming language, simply inventing tricks to overcome its idiosyncrasies and malfunctionings; the few propositive people, often lack a sufficient knowledge of the tool's internals, preventing them from being constructive: either they are not ambitious enough, or altogether suggest completely unrealistic functionalities.

2 The Structure of (Procedural) Formal Developments

In all ITP systems based on a procedural proofstyle, proofs are conducted via a progressive refinement of the goal into simpler subgoals (backward reasoning), by means of a fixed set of commands, called *tactics*. The sequence of tactics (a

[1] The first generation comprised systems like Automath, LCF and Mizar. Only Mizar still survives, to testify some interesting design choices, such as the adoption of a declarative proof style.

tree, actually) is usually called a *script*. In order to gain a deeper understanding about the structure of formal proofs it is instructive to look at the structure of these scripts.

In Figure 3 we summarize the structure of some typical Matita scripts, counting the number of invocations for the different tactics.

Contrib	**Arithmetics**		**Chebyshev**		**Lebesgue**		**POPLmark**		**All**	
lines	2624		19674		2037		2984		27319	
theorems	204		757		102		119		1182	
definitions	11		73		63		16		163	
inductive types	3		4		1		12		20	
records	0		0		7		3		10	
tactic	no.	%	no.	%	no.	%	no.	%	no.	%
apply	629	30.2	6031	34.5	424	28.2	1529	32.7	8613	33.4
rewrite	316	15.2	3231	18.5	73	4.9	505	10.8	4125	16.0
assumption	274	13.2	2536	14.5	117	7.8	493	10.5	3420	13.3
intros	359	17.2	1827	10.4	277	18.4	478	10.2	2941	11.4
cases	105	5.0	1054	6.0	266	17.7	477	10.2	1902	7.4
simplify	135	6.5	761	4.4	78	5.2	335	7.2	1309	5.1
reflexivity	71	3.4	671	3.8	12	0.8	214	4.6	968	3.8
elim	69	3.3	351	2.0	14	0.9	164	3.5	598	2.3
cut	30	1.4	262	1.5	15	1.0	59	1.3	366	1.4
split	6	0.3	249	1.4	50	3.3	53	1.1	358	1.4
change	15	0.7	224	1.3	32	2.1	30	0.6	301	1.2
left/right	18	0.8	72	0.4	76	5.0	72	1.6	238	1.0
destruct	2	0.1	16	0.1	3	0.2	141	3.0	162	0.6
generalize	5	0.2	66	0.4	21	1.4	32	0.7	124	0.5
other	49	2.4	139	0.8	45	3.0	91	1.9	324	1.3
total	2083	100.0	17490	100.0	1503	100.0	4673	100.0	25749	100.0
tac/theo	10.2		23.1		14.7		39.2		21.8	

Fig. 3. Tactics invocations

We compare four developments, of a different nature and written by different people: the first development (Arithmetics) is the basic arithmetical library of Matita up to the operations of quotient and modulo; (Chebyshev) contains relatively advanced results in number theory up to Chebyshev result about the asymptotic distribution of prime numbers (subsuming, as a corollary, Bertrand's postulate) [2]; the third development (Lebesgue) is a formalisation of a constructive proof of Lebesgue's Dominated Convergence Theorem [19]; finally, the last development is a solution to part-1 of the POPLmark challenge in different styles (with names, locally nameless and with de Bruijn indexes).

The interest of these developments is that they have been written at a time when Matita contained almost no support for automation, hence they strictly reflect the structure of the underlying logical proofs.

In spite of a few differences[2], the three developments show a substantial similarity in the employment of tactics.

The first natural observation is the substantial simplicity of the procedural proof style, often blurred by the annoying enumeration of special purpose tactics in many system tutorials. In fact, a dozen tactics are enough to cover 98% of the common situations. Most of this tactics have self-explicatory (and relatively standard) names, so we do not discuss them in detail. Among the useful (but, as we see, relatively rare) tactics missing from our list - and apart, of course, the automation tactics - the most interesting one is probably `inversion`, allowing to derive, for a given instance of an inductive property, all the necessary conditions that should hold assuming it as provable.

Figure 3 gives a clear picture of the typical procedural script: it is a long sequence of applications, rewriting and simplifications (that, comprising `assumption` and `reflexivity`, already count for about 75% of all tactics) sometimes intermixed by case analysis or induction. Considering that almost any proof start with an invocation of `intros` (that counts by itself for another 5% of tactics), the inner applications of this tactic are usually related to the application of higher order elimination principles (also comprising many non recursive cases). This provides evidence that most first order results have a flat, clausal form, that seem to justify the choice of a prolog like automatic proof engine adopted by some interactive prover (like, e.g. Coq).

2.1 Small and Large Scale Automation

In Figure 4 we attempt a repartition of tactics in 5 main categories: **equational reasoning**, **basic logical management** (invertible logical rules and assumptions), exploitation of **background knowledge** (essentially, `apply`), **cases analysis** (covering propositional logic and quantifiers), and finally **creative guessing**, comprising induction and cuts. We agree that not any application of induction or cut really requires a particularly ingenious effort, while some instances of case analysis (or application) may comport intelligent choices, but our main point, here, is to stress two facts: (1) *very few* steps of the proof are really interesting; (2) these *are not* the steps where we would expect to have an automatic support from the machine.

Equational reasoning and basic management of (invertible) logical connectives are a kind of underlying "logical glue": a part of the mathematical reasoning that underlies the true argumentation, and is usually left implicit in the typical mathematical discourse. We refer to techniques addressing these kind of operations as *small scale automation*. The purpose of small scale automation is to reduce the verbosity of the proof script (resolution of trivial steps, verification of side

[2] For instance, rewriting is much less used in (Lebesgue) than in the other developments, since the intuitionistic framework requires to work with setoids (and, at that time, Matita provided no support for setoid-rewriting). Similarly, elimination is more used in (POPLmark) since most properties (type judgements, well formedness conditions and so on) are naturally defined as inductive predicates, and you often reason by induction on such predicates.

functionalities	%
rewriting	16
simplification, convertibility, destructuration	11
equational reasoning	**27**
assumption	13
(invertible) connectives	14
basic logical management	**27**
background knowledge(*apply*)	**33**
case analysis	**7**
induction	4
logical cuts	2
creative guessing	**6**

Fig. 4. Main functionalities

conditions, smart matching of variants of a same notion, automatic inference of missing information, etc.). It must be fast, and leave no additional trace in the proof. From the technical point of view, the most challenging aspect of small scale automation is by far the management of equational reasoning, and many interesting techniques addressing this issue (comprising e.g. congruence [17], narrowing [6] or superposition [18]) have been developed over the years. Although the problem of e-unification is, in general, undecidable, in practice we have at present sufficient knowhow to deal with it reasonably well (but apart from a few experimental exceptions like Matita [3], no major interactive prover provides, at present, a strong native support for narrowing or superposition).

In principle, case analysis and the management of background knowledge is another part of the script where automation should behave reasonably well, essentially requiring that kind of exhaustive exploration that fits so well with the computer capabilities. In fact, the search space grows so rapidly, due to the dimension of the library and the explosion of cases that, even without considering the additional complexity due to dependent types (like, e.g. the existential quantifier) and the integration with equational reasoning, we can effectively explore only a relatively small number of possibilities. We refer to techniques addressing these issues as *large scale automation*. Since the user is surely interested to inspect the solution found by the system, large scale automation must return a proof trace that is both human readable and system executable. To be human readable it should not be too verbose, hence its execution will eventually require small scale automation capabilities (independently of the choice of implementing or not large scale automation *on top* of small scale automation).

2.2 Local and Global Knowledge

An orthogonal way to categorize tactics is according to the amount of knowledge they ask over the content of the library (see Fig. 5).

functionalities	%
rewriting	16
apply	33
library exploitation	**49**
simplification, convertibility, destructuration	11
assumption	13
(invertible) connectives	14
case analysis	**7**
induction	4
logicat cuts	2
local reasoning	**51**

Fig. 5. Operations requiring global or local knowledge

Tactics like `apply` and `rewrite` require the user to explicitly *name* the library result to be employed by the system to perform the requested operation. This obviously presupposes a deep knowledge of the background material, and it is one of the main obstacles to the development of a large, reusable library of formalized mathematics. Most of the other tactics, on the contrary, have a quite local nature, just requiring a confrontation with the current goal and its context. The user is usually intrigued by the latter aspects of the proof, but almost invariably suffer the need to interact with a pre-existent library - written by alien people according to alien principles - and especially the lack of support of most systems in assisting the user in its quest for a useful lemma to exploit. It is a matter of fact that the main branches of the formal repositories of most available interactive provers have been developed by a single user or a by a small team of coordinated people and, especially, that their development stopped when their original contributors, for some reason or another, quitted the job. Reusing a repository of formal knowledge has essentially the same problems and complexity of reusing a piece of software developed by different people. As remarked in the *mathematical components* manifesto[3]

> The situation has a parallel in software engineering, where development based on procedure libraries hit a complexity barrier that was only overcome by switching to a more flexible linkage model, combining dynamic dispatch and reflection, to produce software components that are much easier to combine.

One of the main reasons for the slow progress in the usability of interactive provers is that almost all research on automatic theorem proving has been traditionally focused on *local aspects* of formal reasoning, altogether neglecting the problems arising by the need to exploit a large knowledge base of available results.

[3] `http://www.msr-inria.inria.fr/Projects/math-components/manifesto`

3 Exploiting the Library

One could wonder how far are we from the goal to provide full automatic support for all operations like rewriting and application requiring a stronger interaction with the library.

The chart in Figure 6 compares the structure of the old arithmetical development of Matita with the new version comprising automation.

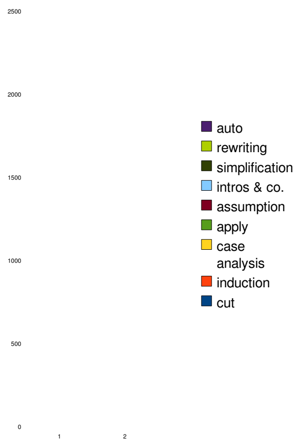


Fig. 6. Arithmetics with (2) and without automation (1)

Applications have been reduced from 629 to 148 and rewriting passed from 316 to 76; they (together with a consistent number of introduction rules) have been replaced by 333 call to automation. It is worth to mention that, in porting the old library to the new system, automation has not been pushed to its very limits, but we constrained it within a temporal bound of *five seconds* per invocation, that looks as a fair bound for an interactive usage of the system. Of course, this is just an upper bound, and automation is usually much faster: the full arithmetical development is compiled in about 3 minutes, that makes an average of less than one second per theorem. Moreover, the automation tactic is able to produce a compact, human readable and executable *trace* for each proof it finds, permitting to recompile the script with the same performance of the original version without automation.

It is not our point to discuss or promote here our particular approach to automation: the above figures must be understood as a purely indicative description of the current state of the art. The interesting point is that the objective to automatize most part of the operations requiring an interaction with the library looks *feasible*, and would give a definitive spin to the usability of interactive provers.

The final point we would like to discuss here is about the possibility of improving automation not acting on the automation algorithm, its architecture or data structures, but merely on our knowledge about the content of library, its internal structure and dependencies. All typical automation algorithms selects new theorems to process according to local information: their size, their "similarity" with the current goal, and so on. Since the library is large and sufficiently stable, it looks worth to investigate different aspects, aimed to estimate the likelihood that applying a given results *in a given situation* will lead us to the expected result. Background knowledge, for humans, is not just a large amount of known results, but also the ability, derived by training and experience, of recognizing specific patterns and to follow different lines of reasoning in different contexts.

This line of research was already traced by Constable et. al [5] more than 20 years ago, but went almost neglected

> *The natural growth path for a system like Nuprl tends toward increased "intelligence". [...] For example, it is helpful if the system is aware of what is in the library and what users are doing with it. It is good if the user knows when to involve certain tactics, but once we see a pattern to this activity, it is easy and natural to inform the system about it. Hence there is an impetus to give the system more knowledge about itself.*

It looks time to invest new energy in this program, paving the way to the third generation of Interactive Provers.

References

1. Asperti, A., Geuvers, H., Natarajan, R.: Social processes, program verification and all that. Mathematical Structures in Computer Science 19(5), 877–896 (2009)
2. Asperti, A., Ricciotti, W.: About the formalization of some results by Chebyshev in number theory. In: Berardi, S., Damiani, F., de'Liguoro, U. (eds.) TYPES 2008. LNCS, vol. 5497, pp. 19–31. Springer, Heidelberg (2009)
3. Asperti, A., Tassi, E.: Smart matching. In: Autexier, S., et al. (eds.) AISC/Calculemus/MKM 2010. LNCS (LNAI), vol. 6167, pp. 263–277. Springer, Heidelberg (2010)
4. Avigad, J., Donnelly, K., Gray, D., Raff, P.: A formally verified proof of the prime number theorem. ACM Trans. Comput. Log. 9(1) (2007)
5. Constable, R.L., Allen, S.F., Bromley, H.M., Cleaveland, W.R., Cremer, J.F., Harper, R.W., Howe, D.J., Knoblock, T.B., Mendler, N.P., Panangaden, P., Sasaki, J.T., Smith, S.F.: Implementing Mathematics with the Nuprl Development System. Prentice-Hall, Englewood Cliffs (1986)
6. Escobar, S., Meseguer, J., Thati, P.: Narrowing and rewriting logic: from foundations to applications. Electr. Notes Theor. Comput. Sci. 177, 5–33 (2007)
7. Geuvers, H.: Proof Assistants: history, ideas and future. Sadhana 34(1), 3–25 (2009)
8. Gonthier, G.: The four colour theorem: Engineering of a formal proof. In: Kapur, D. (ed.) ASCM 2007. LNCS (LNAI), vol. 5081, p. 333. Springer, Heidelberg (2008)
9. Gonthier, G.: Formal proof – the four color theorem. Notices of the American Mathematical Society 55, 1382–1394 (2008)
10. Gordon, M.: From lcf to hol: a short history. In: Proof, Language, and Interaction: Essays in Honour of Robin Milner, pp. 169–186. MIT Press, Cambridge (2000)

11. Gordon, M.: Twenty years of theorem proving for hols past, present and future. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2008. LNCS, vol. 5170, pp. 1–5. Springer, Heidelberg (2008)
12. Hales, T.: Formal proof. Notices of the American Mathematical Society 55, 1370–1381 (2008)
13. Hales, T.C.: The Jordan curve theorem, formally and informally. The American Mathematical Monthly 114, 882–894 (2007)
14. Harrison, J.: A Short Survey of Automated Reasoning. In: Anai, H., Horimoto, K., Kutsia, T. (eds.) Ab 2007. LNCS, vol. 4545, pp. 334–349. Springer, Heidelberg (2007)
15. Harrison, J.: Formalizing an analytic proof of the prime number theorem. J. Autom. Reasoning 43(3), 243–261 (2009)
16. Nederpelt, R.P., Geuvers, J.H., de Vrijer, R.C. (eds.): Selected Papers on Automath. Studies in Logic and the Foundations of Mathematics, vol. 133. Elsevier Science, Amsterdam (1994) ISBN-0444898220
17. Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. J. ACM 27(2), 356–364 (1980)
18. Nieuwenhuis, R., Rubio, A.: Paramodulation-based thorem proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 471–443. Elsevier and MIT Press (2001); ISBN-0-262-18223-8
19. Coen, C.S., Tassi, E.: A constructive and formal proof of Lebesgue's dominated convergence theorem in the interactive theorem prover Matita. Journal of Formalized Reasoning 1, 51–89 (2008)
20. van Benthem Jutting, J.: Checking Landau's "Grundlagen" in the Automath system. Mathematical centre tracts n. 83. Mathematisch Centrum, Amsterdam (1979)
21. Wiedijk, F.: The "De Bruijn factor" (2000), http://www.cs.ru.nl/~freek/factor/
22. Wiedijk, F.: Estimating the cost of a standard library for a mathematical proof checker (2001), http://www.cs.ru.nl/~freek/notes/mathstdlib2.pdf
23. Wiedijk, F.: A new implementation of Automath. Journal of Automated Reasoning 29, 365–387 (2002)

Mechanized Mathematics

Jacques Carette

Department of Computing and Software, McMaster University
www.cas.mcmaster.ca/~carette

Abstract. If one were designing an entirely new mathematical assistant, what might it look like? Problems and some favoured solutions are presented.

In the 50 years since McCarthy's " Recursive Functions of Symbolic Expressions and Their Computation by Machine", what have we learned about the realization of Leibniz's dream of just being able to utter "Calculemus!"[1] when faced with a mathematical dilemma?

In this talk, I will first present what I see as the most important lessons from the past which need to be heeded by modern designers. From the present, I will look at the context in which computers are used, and derive further requirements. In particular, now that computers are no longer the exclusive playground for highly educated scientists, usability is now more important than ever, and justifiably so.

I will also examine what I see as some principal failings of current systems, primarily to understand some major mistakes to avoid. These failings will be analyzed to extract what seems to be the root mistake, and I will present my favourite solutions.

Furthermore, various technologies have matured since the creation of many of our systems, and whenever appropriate, these should be used. For example, our understanding of the *structure* of mathematics has significantly increased, yet this is barely reflected in our libraries. The extreme focus on efficiency by the computer algebra community, and correctness by the (interactive) theorem proving community should no longer be considered viable long term strategies. But how does one effectively bridge that gap?

I personally find that a number of (programming) *language-based* solutions are particularly effective, and I will emphasize these. Solutions to some of these problems will be illustrated with code from a prototype of MathScheme 2.0, the system I am developing with Bill Farmer and our research group.

[1] Let us calculate!

Formal Proof of SCHUR Conjugate Function*

Franck Butelle[1], Florent Hivert[2], Micaela Mayero[1,3], and Frédéric Toumazet[4]

[1] LIPN UMR 7030, Université Paris 13, Villetaneuse, F-93430
[2] LITIS EA 4108, Université de Rouen, Saint-Etienne-du-Rouvray, F-76801
[3] LIP, INRIA Grenoble – Rhône-Alpes, UMR 5668, UCBL, ENS Lyon, Lyon, F-69364
[4] LIGM UMR 8049, Université de Marne-la-Vallée, Champs sur Marne, F-77454

Abstract. The main goal of our work is to formally prove the correctness of the key commands of the SCHUR software, an interactive program for calculating with characters of Lie groups and symmetric functions. The core of the computations relies on enumeration and manipulation of combinatorial structures. As a first "proof of concept", we present a formal proof of the conjugate function, written in C. This function computes the conjugate of an integer partition. To formally prove this program, we use the Frama-C software. It allows us to annotate C functions and to generate proof obligations, which are proved using several automated theorem provers. In this paper, we also draw on methodology, discussing on how to formally prove this kind of program.

1 Introduction

SCHUR [1] is an interactive software for calculating properties of Lie groups and symmetric functions [2]. It is used in research by combinatorists, physicists, theoretical chemists [3] as well as for educational purpose as a learning tool for students in algebraic combinatorics. One of its main uses is to state conjectures on combinatorial objects. For such use, it is important to have some confidence in the results produced by SCHUR.

Until now, the method used to get some confidence in the results has mostly been based on just one example for each command.

The computation of other examples is complex due to the well known combinatorial explosion, especially when using algorithms associated to the symmetric group, see Sect. 2.1. Unfortunately, the combinatorial explosion as well as computing time forbid test generation or verification techniques (model checking). Therefore, in this paper, we focus on formal proof of the existing program.

With the aim of verifying the whole software, we start with proving the correctness of its fundamentals bricks. The main combinatorial object used in SCHUR is integer partition. The first non-trivial operation on integer partitions is the conjugate. Moreover, conjugate function is necessary for more than half of the 240 interactive commands of SCHUR. From this point of view, we can say that conjugate is a critical function of SCHUR.

* This research is supported by the PEPS-CNRS project CerBISS.

The very first work consists in isolating (see Sect. 4.3) the code of this function from the program. Next, we chose to use the most popular tool of program proof community's Frama-C [4] successor of Caduceus. Frama-C is a plug-in system. In order to prove programs, we used Jessie [5], the deductive verification plug-in of C programs annotated with ACSL [6]. The generated verification conditions can be submitted to external automatic provers, and for more complex situations, to interactive theorem provers as well (see Sect. 2.2).

After a short presentation of software tools and theoretical concepts, we will present the formal proof of a program. Finally, after discussing difficulties and mistakes encountered along the way, we will propose a methodology to prove such a software, and finally discuss future work.

2 Presentation of the Software Used

2.1 The SCHUR Software

SCHUR is an interactive software for calculating properties of Lie groups and symmetric functions. A Symmetric Function is a function which is symmetric, or invariant, under any permutation of its variables. For example $f(x_1, x_2, x_3) = x_1 + x_2 + x_3$ is a symmetric function.

SCHUR has originally written by Prof. Brian G. Wybourne in Pascal language. Then it was translated into C by an automatic program making it quite difficult to read. There are almost no comments in the code, the code is more than 50,000 lines long with many global variables. Local variables have names such as $W52$ and so on.

After the death of Prof. Wybourne in November 2003, some people felt that his program should be maintained, and if possible enhanced, with a view to making it freely available to the mathematics and physics research community.

Nowadays, it is open source under the GPL license and includes more than 240 commands. The code still includes very few comments. Some mistakes have been corrected but some interactive commands are so intricate that it is difficult to have more than a few examples to check them against and most people do not even know if the result is correct or not.

This is why we started to work on this code. Firstly some of the commands in SCHUR are very well implemented (for example, plethysm is computed faster by SCHUR than by many other combinatorial toolboxes). Formally proving some key functions inside would also be a major advance for its research community.

2.2 The Frama-C Software

Frama-C [4] is an open source extensible platform dedicated to source code analysis of C software. It is co-developed by two French public institutions: CEA–LIST (Software Reliability Laboratory) and INRIA-Saclay (ProVal project).

Frama-C is a plug-in system. In order to prove programs, we use Jessie [5], the deductive verification plug-in of C programs annotated with ACSL [6]. It uses internally the languages and tools of the Why platform [7]. The Jessie plug-in uses Hoare-style [8] weakest precondition computations to formally prove ACSL properties. The generated verification conditions (VC) can be submitted to external automatic provers such as Simplify [9], Alt-Ergo [10], Z3 [11], CVC3 [12].

These automatic provers belong to SMT (Satisfiability Modulo Theories) solvers. The SMT problem is a decision problem for logical formulas with respect to combinations of background theories expressed in classical first-order logic with equality. First-order logic is undecidable. Due to this high computational difficulty, it is not possible to build a procedure that can solve arbitrary SMT problems. Therefore, most procedures focus on the more realistic goal of efficiently solving problems that occur in practice.

For more complex situations, interactive theorem provers can be used to establish the validity of VCs, like Coq [13], PVS [14], Isabelle/HOL [15], etc. For our purpose, we used Coq (see Sect. 4.2) since it is the one best known to the authors.

3 The Conjugate Function

In this section, the basics of algebraic combinatorics are given so that the reader can understand what is actually proved. Interestingly in this field, though the interpretation of what is actually computed can be of a very abstract algebraic level, the computation itself boils down most of the time to possibly intricate but rather elementary manipulations.

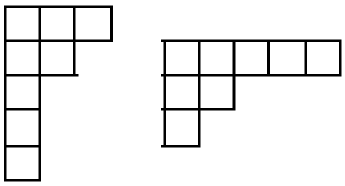
3.1 Combinatorial and Algebraic Background: Integer Partitions

A **partition** of a positive integer n is a way of writing n as a sum of a non-increasing sequence of integers. For example $\lambda = (4, 2, 2, 1)$ and $\mu = (2, 1)$ are partitions of $n = 9$ and $n' = 3$ respectively. We write $|\lambda| = n$ and $|\mu| = n'$ [16].

The **Ferrers diagram** F^λ associated to a partition $\lambda = (\lambda_1, \lambda_2, ..., \lambda_p)$ consists of $|\lambda| = n$ boxes, arranged in $l(\lambda) = p$ left-justified rows of lengths λ_1, λ_2, ..., λ_p. Rows in F^λ are oriented downwards (or upwards for some authors). F^λ is called the shape of λ.

Definition 1. *The conjugate of an integer partition is the partition associated to the diagonal symmetric of its shape.*

For example, for $\lambda = (3, 2, 1, 1, 1)$, here is the Ferrers diagram F^λ and the Ferrers diagram of the conjugate partition:



So the conjugate partition of $(3, 2, 1, 1, 1)$ is $(5, 2, 1)$.

A **semi-standard Young tableau** of shape λ is a numbering of the boxes of F^λ with entries from $\{1, 2, ..., n\}$, weakly increasing across rows and strictly increasing down columns. A tableau is **standard** if and only if each entry appears only once. Here is an example of shape $(4, 2, 2, 1)$ tableau:

$$\begin{array}{|c|c|c|c|}\hline 1&2&2&5\\ \hline 2&4&\multicolumn{2}{c}{}\\ \cline{1-2} 3&6&\multicolumn{2}{c}{}\\ \cline{1-2} 5&\multicolumn{3}{c}{}\\ \cline{1-1}\end{array}$$

A **symmetric function** of a set of variables $\{x_1, x_2, \ldots\}$ is a function $f(x_1, x_2, \ldots)$ of those variables which is invariant under any permutation of those variables (that is for example $f(x_1, x_2, \ldots) = f(x_2, x_1, \ldots)$). This definition is usually restricted to polynomial functions. The most important linear basis of symmetric function's algebra is called the **Schur functions** and they are combinatorially defined as follows: for a given semi-standard Young tableau T of shape λ, write $\mathbf{x}^T$ the product of the x_i for all i appearing in the tableau. Then

$$s_\lambda(\mathbf{x}) = \sum_{T \in \mathrm{Tab}(\lambda)} \mathbf{x}^T. \tag{1}$$

where $\mathrm{Tab}(\lambda)$ is the set of all tableaux of shape λ. We will note $s_\lambda(\mathbf{x})$, s_λ. For example, consider the tableaux of shape $(2, 1)$ using just three variables x_1, x_2, x_3:

$$\begin{array}{|c|c|}\hline 1&1\\ \hline 2&\multicolumn{1}{c}{}\\ \cline{1-1}\end{array}\quad \begin{array}{|c|c|}\hline 1&1\\ \hline 3&\multicolumn{1}{c}{}\\ \cline{1-1}\end{array}\quad \begin{array}{|c|c|}\hline 2&2\\ \hline 3&\multicolumn{1}{c}{}\\ \cline{1-1}\end{array}\quad \begin{array}{|c|c|}\hline 1&2\\ \hline 3&\multicolumn{1}{c}{}\\ \cline{1-1}\end{array}\quad \begin{array}{|c|c|}\hline 1&3\\ \hline 2&\multicolumn{1}{c}{}\\ \cline{1-1}\end{array}\quad \begin{array}{|c|c|}\hline 1&2\\ \hline 2&\multicolumn{1}{c}{}\\ \cline{1-1}\end{array}\quad \begin{array}{|c|c|}\hline 1&3\\ \hline 3&\multicolumn{1}{c}{}\\ \cline{1-1}\end{array}\quad \begin{array}{|c|c|}\hline 2&3\\ \hline 3&\multicolumn{1}{c}{}\\ \cline{1-1}\end{array}$$

The associated Schur function is therefore:

$$s_{(21)}(x_1, x_2, x_3) = x_1^2x_2 + x_1^2x_3 + x_2^2x_3 + 2x_1x_2x_3 + x_1x_2^2 + x_1x_3^2 + x_2x_3^2 \tag{2}$$

thus:

$$s_{(21)} = s_{(21)}(x_1, x_2, x_3) + s_{(21)}(x_1, x_2, x_3, x_4) + \ldots$$

Note that, with this combinatorial definition, the symmetry of $s_{(21)}(x_1, x_2, x_3)$ is not exactly obvious.

We need to recall some well-known results in symmetric function theory: though Schur functions have historically been defined by Jacobi [17], they were named in the honor of Schur who discovered their crucial role in the representation theory of the symmetric group and the general linear group. Namely, after the discovery by Frobenius that the irreducible representation of the symmetric groups are indexed by integer partitions, Schur showed that those functions can be interpreted as characters of those irreducible representation, and by Schur-Weyl duality characters of Lie groups and Lie algebras. Notably we obtain the representation of the general linear groups (GL_n) and unitary groups (U_n) [18] from the symmetric group representations. In this setting, the conjugate of the partition essentially encodes the tensor product of a representation by the sign representation.

Further work by Schur-Littlewood involve infinite sum of Schur functions associated to partitions [19], whose conjugates have a particular form. In particular, these series are used to obtain symplectic (Sp_{2n}) and orthogonal character groups (O_n) (symmetric and orthogonal Schur functions) from standard Schur functions [20].

One particularly important and difficult computational problem here is plethysm (see SCHUR reference manual [1] and [2]). It is the analogue in symmetric functions of the substitution of polynomial inside another polynomial $f(x) \mapsto f(g(x))$. It is called plethysm because by some combinatorial explosion, it involves very quickly a lot (a plethora) of terms, making it something very difficult to compute efficiently. For example, $s_{(21)}(s_{(21)})$, the first example with non trivial partitions in the input is already very hard to compute by hand. First we can regard $s_{(21)}$ as a function in as many monomial as in (2):

$$s_{(21)}(s_{(21)})(x_1, x_2, x_3) = s_{(21)}(x_1^2x_2, x_1^2x_3, x_2^2x_3, x_1x_2x_3, x_1x_2x_3, x_1x_2^2, x_1x_3^2, x_2x_3^2)$$

it can be shown that the following holds:

$$\begin{aligned} s_{(21)}(s_{(21)}) = {} & s_{(22221)} + s_{(321111)} + 2s_{(32211)} + s_{(3222)} + s_{(33111)} + \\ & 3s_{(3321)} + 2s_{(42111)} + 3s_{(4221)} + 3s_{(4311)} + 3s_{(432)} + \\ & s_{(441)} + s_{(51111)} + 2s_{(5211)} + s_{(522)} + 2s_{(531)} + s_{(54)} + s_{(621)} \end{aligned}$$

3.2 Computation in Algebraic Combinatorics

Basically, the architecture of a software for computing in algebraic combinatorics is composed of two parts:

- a computer algebra kernel dealing with the bookkeeping of expressions and linear combinations (parsing, printing, collecting, Gaussian and Groebner elimination algorithm...);
- a very large bunch of small combinatorial functions which enumerate and manipulate the combinatorial data structures.

In algebraic combinatorics software, for each basic combinatorial structure such as permutations or partitions, there are typically 50-200 different functions. Conjugating a partition is a very good example of what those many functions do, that is surgery on lists of integers or lists of lists of integers or more advanced recursive structures like trees... In a basic computation, most of the time is spent mapping or iterating those functions on some sets of objects. But due to combinatorial explosion those sets can be very large so these functions must be very well optimized.

3.3 Properties

The definition of conjugate (diagonal symmetric of its partition shape) is easy to understand but may conduct to naive implementations that may be inefficient.

Let us suppose that we represent an integer partition by an integer array starting from 1. For example $\lambda = (3, 2, 1, 1, 1)$ gives $t[1] = 3$, $t[2] = 2$,... $t[l(\lambda)] = 1$. Recall that $t[i]$ is non-increasing, that is $t[i + 1] \leq t[i]$.

One way to compute the conjugate is to count boxes: in our previous example the first column of λ had 4 boxes, the second had 3 etc. Therefore, to compute the number of boxes in a column j we need to know how many lines are longer than j. As a consequence, if t_c is the array representing the conjugate, the following formula gives the value of the entries of the conjugate:

$$t_c[j] = |\{i \mid 1 \leq i \leq l(\lambda) \land t[i] \geq j\}| \ .$$

Note that $t_c[j] = 0$ if $j > t[1]$, so the previous expression must be computed only from $j = 1$ to $j = t[1]$. This last property will be one of our predicates used to check the correctness of loop invariants.

3.4 SCHUR Implementation

Here follows the code of the conjugate function extracted from the SCHUR software. We expanded type definitions (C "structs" and "typedefs") from the original code just to simplify the work of Frama-C and to make this part of code independent from the rest of the SCHUR software (getting rid of global variables and so on).

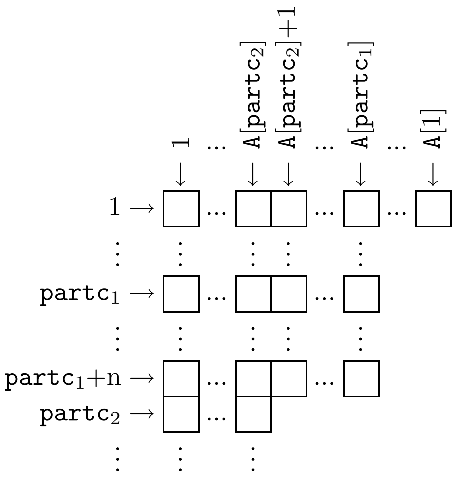
```
#define MAX 100

void conjgte (int A[MAX], int B[MAX]) {
  int i, partc = 1, edge = 0;

  while (A[partc] != 0) {
     edge = A[partc];
     do
          partc = partc + 1;
     while (A[partc] == edge);
     for (i = A[partc] + 1; i <= edge; i++)
          B[i] = partc - 1;
  }
}
```

Note that this implementation is not naive (and not so easy to understand) but its time complexity is optimal (linear in the length of the partition).

The algorithm is based on looking for the set of descents of the partition[1]. The do–while loop follows a "flat" portion of the partition ($t[i] = t[i - 1]$) until a descent is found. Next the for–loop assigns the values of the B array according to the flat portion. The following figure clarifies this: we have denoted $\texttt{partc}_1$ the value of `partc` at the entrance of while loop. $\texttt{partc}_2$ is the value of `partc` after

[1] A descent is such that $t[i] < t[i - 1]$.

the do–while loop. For clarity's sake we supposed A[$\texttt{partc}_2$]+1 to be different from A[$\texttt{partc}_1$]. If we count boxes column by column to construct array B, it is clear that B[i]=$\texttt{partc}_2$-1 for all A[$\texttt{partc}_2$]+1 $\leq i \leq$ A[$\texttt{partc}_1$]=edge.



4 The Formal Proof of the Conjugate Function

4.1 Annotations

In the following paragraphs we present the annotations added to the code. Note that this is the only additions made to it. First we have to specify the model of integers we want to deal with:

```
#pragma JessieIntegerModel(strict)
```

This means that int types are modeled by integers with appropriate bounds, and for each arithmetic operation, it is mandatory to show that no overflow occurs.

Next, we have to express in first-order logic what an integer partition (stored in an array) is:

```
#define MAX 100
/*@ predicate is_partition{L}(int t[]) =
    (\forall integer i; 1 <= i < MAX ==> 0 <= t[i] < (MAX-1)) &&
    (\forall integer i,j; 1 <= i <=j < MAX ==> t[j] <= t[i]) &&
    t[MAX-1]==0;
  */
```

Note that annotations are coded in the C comments, starting with a @. The {L} term is the context (pre, post, etc.), we won't detail it here, see [5,6] for details.

The data structure (array of integers) comes from the way the SCHUR software represents integer partitions. 0 is used as a mark of end of array, just like character strings in C. The MAX value comes from the original source code as well. The first line of the predicate `is_partition` expresses that we are able to compute the conjugate (if at least one element is greater than or equal to MAX-1, the conjugate will no be able to be stored in an array of size MAX-1 with the last element fixed to 0). From the source code it is expressed by an external simple test on `t[1]`, but expressing it like that simplifies automatic provers job. The second line of the predicate defines the non-increasing order.

The following predicate is needed to express how we count blocs to compute the conjugate. It may be read as z equals the number of elements of partition t, whose indexes are included in $\{1, .., j-1\}$ and whose values are greater than or equal to k. It is theoretically possible to express it as an axiomatic theory, a kind of function, but automatic provers we use make a better use of predicates. Note that we need to explicit the $z = 0$ case, in order to be able to prove the global post-condition `is_conjugate(A,B)`.

```
/*@predicate countIfSup{L}(int t[],integer j,integer k,integer z)=
   is_partition{L}(t) &&
   1<= j <= MAX &&
   1<= k < MAX &&
   ((1<=z<j && \forall integer i ;  1<=i<=z ==> t[i]>= k)
    || (z==0 && \forall integer i ; 1<=i<j ==> t[i]<k)) ;
 */
```

Here is what we want to obtain at the end of the computation, t2 is a conjugate of t1 if the following holds:

```
/*@ predicate is_conjugate{L}(int t1[], int t2[]) =
    \forall integer k ; 1<=k<MAX ==> countIfSup(t1,MAX,k,t2[k]);
 */
```

Finally, here is the function. First we have to give precise requirements on the inputs. For example, (`\valid(A+ (1..(MAX-1)))` means that memory has been allocated so array indexes from 1 to MAX-1 are allowed). From the original code, the B array is supposed to be initialized with zeros before calling the function. This is translated into a `requires` directive. Next, we specify which memory elements are modified by the function (`assigns`). This is used for safety proofs. In the end, the output is correct if the post-condition (`ensures`) is met.

```
/*@ requires \valid(A+ (1..(MAX-1)));
    requires \valid(B+ (1..(MAX-1)));
    requires is_partition(A);
    requires \forall integer k; 1<=k<MAX ==> B[k]== 0;
    assigns B[1..A[1]];
    ensures is_conjugate(A,B);
    */
void conjgte (int A[MAX], int B[MAX])
{
  int i, partc=1, edge = 0 ;
```

Now we have to define the loop variant and invariant for each loop (to prove properties). The "loop variant" must decrease, while remaining non negative, to be able to prove termination. We also use a "ghost variable" to store the state of a variable before any modification.

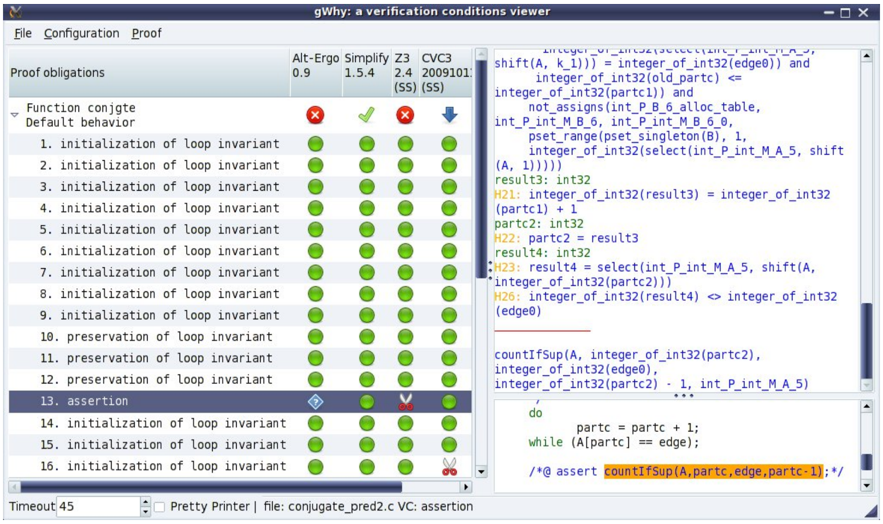
```
  /*@ loop variant MAX-partc;
      loop invariant 1<=partc<MAX;
      loop assigns B[1..A[1]];
      loop invariant \forall integer k;
          A[partc]+1 <=k <= A[1] ==> countIfSup(A,MAX,k,B[k]);
    */
  while (A[partc] != 0) {
     edge = A[partc];

     /*@ ghost int old_partc = partc; */

    /*@ loop variant MAX-partc;
        loop invariant old_partc<=partc ;
        loop invariant \forall integer k;
            old_partc<= k <= partc ==> A[k]==edge;
        loop invariant partc<MAX-1;
     */
     do
         partc = partc + 1;
     while (A[partc] == edge);
```

We also use the assert directive to have a verification point of a property that may help automatic provers for the next properties or global ones.

```
     /*@ assert countIfSup(A,partc,edge,partc-1);*/

     /*@ loop variant edge-i;
         loop invariant i >= A[partc]+1 && edge+1>=i ;
         loop invariant \forall integer k;
             A[partc]+1 <=k <i ==> countIfSup(A,MAX,k,B[k]);
         loop assigns B[ (A[partc]+1)..edge];
       */
     for (i = A[partc] + 1; i <= edge; i++)
         B[i] = partc - 1;
  }
}
```

4.2 Proofs

Fig. 1 to Fig. 3 are snapshots of gWhy (Frama-c graphical interface when using plugin Jessie). We applied this tool on the previous annotated code.



Proof obligations	Alt-Ergo 0.9	Simplify 1.5.4	Z3 2.4 (SS)	CVC3 2009101 (SS)
Function conjgte Default behavior	✗	✓	✗	↓
1. initialization of loop invariant	●	●	●	●
2. initialization of loop invariant	●	●	●	●
3. initialization of loop invariant	●	●	●	●
4. initialization of loop invariant	●	●	●	●
5. initialization of loop invariant	●	●	●	●
6. initialization of loop invariant	●	●	●	●
7. initialization of loop invariant	●	●	●	●
8. initialization of loop invariant	●	●	●	●
9. initialization of loop invariant	●	●	●	●
10. preservation of loop invariant	●	●	●	●
11. preservation of loop invariant	●	●	●	●
12. preservation of loop invariant	●	●	●	●
13. assertion	◇	●	✂	●
14. initialization of loop invariant	●	●	●	●
15. initialization of loop invariant	●	●	●	●
16. initialization of loop invariant	●	●	●	✂

```
shift(A, k_1))) = integer_of_int32(edge0)) and
      integer_of_int32(old_partc) <=
integer_of_int32(partc1)) and
      not_assigns(int_P_B_6_alloc_table,
int_P_int_M_B_6, int_P_int_M_B_6_0,
      pset_range(pset_singleton(B), 1,
      integer_of_int32(select(int_P_int_M_A_5, shift
(A, 1)))))
result3: int32
H21: integer_of_int32(result3) = integer_of_int32
(partc1) + 1
partc2: int32
H22: partc2 = result3
result4: int32
H23: result4 = select(int_P_int_M_A_5, shift(A,
integer_of_int32(partc2)))
H26: integer_of_int32(result4) <> integer_of_int32
(edge0)
_______________
countIfSup(A, integer_of_int32(partc2),
integer_of_int32(edge0),
integer_of_int32(partc2) - 1, int_P_int_M_A_5)
```

```
    do
           partc = partc + 1;
    while (A[partc] == edge);

    /*@ assert countIfSup(A,partc,edge,partc-1);*/
```

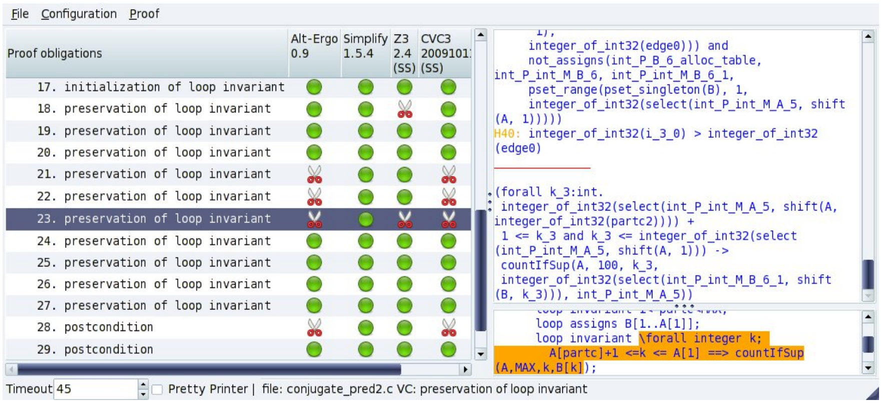
Timeout 45 Pretty Printer | file: conjugate_pred2.c VC: assertion

Fig. 1. Graphical Interface: default behavior

The Verification Conditions (VC, also called proof obligations) that have to be proved one by one (line by line) appear to the left of each of the following snapshots. In the upper right part of the window, we can check at a glance what hypotheses are known and what is to be proved at the bottom of it (under the line). No circularity paradox is possible here, since the proof of a VC can only rely on other VC higher in the control-flow graph of the function.

In the lower right part of the window, the corresponding part of the annotation is highlighted in the source code with some lines before and after it.

We will now focus on the VC part, to the left. We can see (green) dots meaning that this property has been proved by this prover. There is also (blue) rhombus



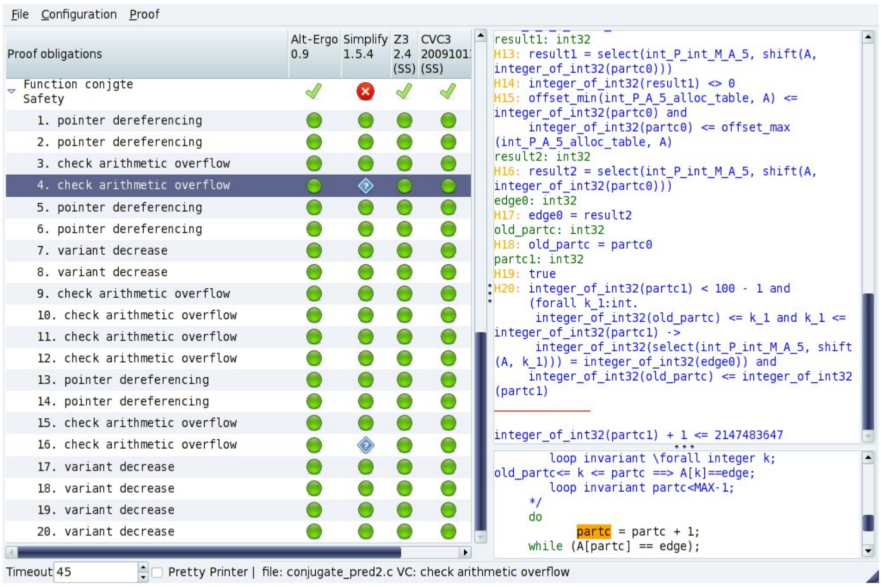
Proof obligations	Alt-Ergo 0.9	Simplify 1.5.4	Z3 2.4 (SS)	CVC3 2009101 (SS)
17. initialization of loop invariant	●	●	●	●
18. preservation of loop invariant	●	●	✂	●
19. preservation of loop invariant	●	●	●	●
20. preservation of loop invariant	●	●	●	●
21. preservation of loop invariant	✂	●	●	✂
22. preservation of loop invariant	✂	●	●	✂
23. preservation of loop invariant	✂	●	✂	✂
24. preservation of loop invariant	●	●	●	●
25. preservation of loop invariant	●	●	●	●
26. preservation of loop invariant	●	●	●	●
27. preservation of loop invariant	●	●	●	●
28. postcondition	✂	●	●	✂
29. postcondition	●	●	●	●

```
      integer_of_int32(edge0))) and
      not_assigns(int_P_B_6_alloc_table,
int_P_int_M_B_6, int_P_int_M_B_6_1,
      pset_range(pset_singleton(B), 1,
      integer_of_int32(select(int_P_int_M_A_5, shift
(A, 1)))))
H40: integer_of_int32(i_3_0) > integer_of_int32
(edge0)
_______________
(forall k_3:int.
 integer_of_int32(select(int_P_int_M_A_5, shift(A,
integer_of_int32(partc2)))) +
 1 <= k_3 and k_3 <= integer_of_int32(select
(int_P_int_M_A_5, shift(A, 1))) ->
 countIfSup(A, 100, k_3,
 integer_of_int32(select(int_P_int_M_B_6_1, shift
(B, k_3)), int_P_int_M_A_5))
```

```
     loop assigns B[1..A[1]];
     loop invariant \forall integer k;
       A[partc]+1 <=k <= A[1] ==> countIfSup
(A,MAX,k,B[k]);
```

Timeout 45 Pretty Printer | file: conjugate_pred2.c VC: preservation of loop invariant

Fig. 2. Graphical Interface continued



Proof obligations	Alt-Ergo 0.9	Simplify 1.5.4	Z3 2.4 (SS)	CVC3 2009101: (SS)
Function conjgte Safety	✓	✗	✓	✓
1. pointer dereferencing	●	●	●	●
2. pointer dereferencing	●	●	●	●
3. check arithmetic overflow	●	●	●	●
4. check arithmetic overflow	●	?	●	●
5. pointer dereferencing	●	●	●	●
6. pointer dereferencing	●	●	●	●
7. variant decrease	●	●	●	●
8. variant decrease	●	●	●	●
9. check arithmetic overflow	●	●	●	●
10. check arithmetic overflow	●	●	●	●
11. check arithmetic overflow	●	●	●	●
12. check arithmetic overflow	●	●	●	●
13. pointer dereferencing	●	●	●	●
14. pointer dereferencing	●	●	●	●
15. check arithmetic overflow	●	●	●	●
16. check arithmetic overflow	●	?	●	●
17. variant decrease	●	●	●	●
18. variant decrease	●	●	●	●
19. variant decrease	●	●	●	●
20. variant decrease	●	●	●	●

```
result1: int32
H13: result1 = select(int_P_int_M_A_5, shift(A,
integer_of_int32(partc0)))
H14: integer_of_int32(result1) <> 0
H15: offset_min(int_P_A_5_alloc_table, A) <=
integer_of_int32(partc0) and
     integer_of_int32(partc0) <= offset_max
(int_P_A_5_alloc_table, A)
result2: int32
H16: result2 = select(int_P_int_M_A_5, shift(A,
integer_of_int32(partc0)))
edge0: int32
H17: edge0 = result2
old_partc: int32
H18: old_partc = partc0
partc1: int32
H19: true
H20: integer_of_int32(partc1) < 100 - 1 and
     (forall k_1:int.
      integer_of_int32(old_partc) <= k_1 and k_1 <=
integer_of_int32(partc1) ->
      integer_of_int32(select(int_P_int_M_A_5, shift
(A, k_1))) = integer_of_int32(edge0)) and
     integer_of_int32(old_partc) <= integer_of_int32
(partc1)
______________

integer_of_int32(partc1) + 1 <= 2147483647
```

```
        loop invariant \forall integer k;
old_partc<= k <= partc ==> A[k]==edge;
        loop invariant partc<MAX-1;
     */
     do
            partc = partc + 1;
     while (A[partc] == edge);
```

Timeout 45 Pretty Printer | file: conjugate_pred2.c VC: check arithmetic overflow

Fig. 3. Graphical Interface: Safety

with a question mark inside (see assertion 13), indicating that this prover will not be able to to prove this property. Actually, this does not mean that this VC is wrong, remember that these provers use heuristics. Sometimes, you may see scissors meaning that the maximum execution time has been reached without proving the VC. Again, this does not mean that the corresponding VC is wrong. Finally, at the top of a column a (green) check or (red, with a white cross inside) point is shown. The first one means that all properties have been proved by that prover. In fig.1, The (blue) arrow at the top of the CVC3 column means that it is still computing some unshown VC (greater than number 16).

The last figure is the final part. The provers have worked on the safety of the code, that is to say, integer bounds (overflow problems), pointer referencing and termination.

As seen in Sect. 4.1, the `B` array has to be initialized with zeros before calling the function. This requirement has been enlightened thanks to the annotations and tools, in particular because without the line `requires \forall integer k;1<=k<MAX ==> B[k]==0`, the postcondition which states that `B` is a conjugate of `A` cannot be proved.

We have also used Coq proof assistant. However, it not being essential to our present point, we chose to live aside the detail of this procedure (see Sect. 4.3).

4.3 Problems, Mistakes

As usual when using formal proof tools, there are several ways to formalize or to annotate programs. Choices made during at this stage are very important for future proofs. For example, declaring a function as an axiomatic theory or as a predicate will suppose corresponding proofs to be different. We can make a similar remark with data-types used in programs.

For these reasons, using "good" annotations which allows automatic provers to prove verification conditions (VC) successfully is a clever way to go about it.

When we deal with 40,000 lines of undocumented code, another critical part of the work consists in "correctly" isolating the piece of code that we want to prove. The code can use global variables, initializations made by other functions, or use intricate data-types and so on.

In the following paragraphs, these problems and associated mistakes are discussed.

Isolating a Part of Program. Generally speaking, the analyzed function must be free of external calls. More precisely if a function is called from it, it has to be incorporated in the code (like macro expansion) or, at least, independently proved.

Next, data types must be simplified. Even if Frama-C can cope with simple structures, it is better to have a first pass on them (unions suppression, typedef expansion and so on).

How to Make Good Annotations? As previously explained, ACLS is a language which is used to annotate C programs. Annotating an existing program consists in choosing properties (comportment, results,...) that the user wants to be "confirmed", such as preconditions, loop invariants, post conditions. In our case, for example, one of the most relevant properties we proved is that *the result B is the conjugate of the partition A*. This property is stated as a postcondition.

As usual, there are several ways to formalize annotations. Particularly when using external provers, a good method is to know how provers work. Here, we have to remember that the automatic provers are SMT solvers (see Sect. 2.2).

As an example, we can give the definition of `countIfSup`. In a first formalization we wrote it as an "axiomatization". But due to another problem that we will describe in the next paragraph, we needed to make some proofs in Coq which used `countIfSup`. Then, to make it easier for Coq, we decided to try to define it inductively. Thanks to this other definition, some conditions were automatically proved by SMT solvers. This example shows how important formalization choices can be.

In the next paragraph, we will explain and illustrate how Coq allowed us to correct some errors in our annotations.

Why Coq? Once annotations are completed, the method consists in using automatic provers (using gWhy for example). As previously explained, if all proof obligations are proved by at least one prover, the work can be considered as finished. But, if one or more proof obligations is/are still unproved, several approaches are possible: the first one consists in verifying that annotations are "sufficient", that is to say a precondition or a loop invariant is not missing. Another approach, when the user suppose that his annotations are correct, is to use an external non automatic prover to try to prove proof obligations that have not been verified previously.

In our case, we used the interactive theorem prover Coq twice. The first time was because a postcondition had not been proved by SMT provers. When we began Coq proof, we discovered that the definition of `countIfSup` was incomplete: the second part of the "`||`" (logical or) was missing.

The second time we used Coq was to prove a loop invariant. Similarly, we detected another incompleteness in `countIfSup` definition ($j < MAX$ instead of $j \leq MAX$). Proof assistants are well adapted to detect this kind of problems. Indeed, building formal proofs manually, a user can easily see which hypotheses are necessary.

After having corrected and replaced the "axiomatization" of `countIfSup` by a predicate, all proof obligations have been proved by at least one automatic prover.

Note that the new definition allowed us to remove from the annotations one additional lemma which, at first, appeared necessary.

Other Vicissitudes. Among the main encountered difficulties, we can mention the confidence in the provers we used. In our case, one of the versions of CVC3 was faulty and proved all VC correct, even when they were false. For this reason we decided to consider that a proof obligation was proved when at least two automatic provers succeed on proving it. It is the case for all our obligations except one (VC # 23 is only proved by Simplify). The proof of VC # 23 is in progress using Coq.

5 Conclusion and Future Work

We have isolated and formally proved one of the key commands of the SCHUR software. This work reinforced us in the idea of formally proving chosen parts of software of the same kind, composed of 40,000 lines of undocumented code.

Thanks to this approach, we have focused on critical points (such as particular initializations of arrays and appropriate bounds) from the original code and by extension, we have understood the progression axis of the methodology. In particular, it is better to know how SMT automatic provers work to try to make a "good" annotation so that obligation proofs will be more easily proved by them. In the methodology, non automatic external provers like Coq may be used to refine annotations, and to prove obligations when no automatic provers succeed.

The conjugate function is a basic brick of combinatorics. This give us perspective to prove other functions. Therefore, as a future work, the second step is to prove algorithms relying on exhaustive enumeration algorithm, such as computation of Littlewood-Richardson coefficients, Koskas numbers, Koskas matrices, representation multiplicity in tensor product decompositions, etc.

The final objective will be to build proved libraries usable for scientific community.

References

1. Butelle, F., King, R., Toumazet, F.: SCHUR, an interactive program for calculating properties of Lie groups and symmetric functions, http://schur.sourceforge.net (Release 6.06)
2. MacDonald, I.G.: Symmetric Functions and Hall Polynomials. Clarendon Press, Oxford University Press (1979) (2nd edn. in 1998)
3. King, R.C., Bylicki, M., Karwowski, J. (eds.): Symmetry, Spectroscopy and SCHUR. Nicolaus Copernicus University Press (2006)
4. Correnson, L., Cuoq, P., Puccetti, A., Signoles, J.: Frama-C User Manual, Beryllium release, http://frama-c.cea.fr
5. Marché, C., Moy, Y.: Jessie Tutorial, Release 2.21 http://frama-c.cea.fr/jessie/jessie-tutorial.pdf
6. Baudin, P., Cuoq, P., Filliâtre, J.C., Marché, C., Monate, B., Moy, Y., Prevosto, V.: ACSL: ANSI/ISO C Specification Language, http://frama-c.cea.fr/download/acsl-implementation-Beryllium-20090902.pdf
7. Filliâtre, J.C., Marché, C., Moy, Y., Hubert, T., Rousset, N.: Why is a software verification platform, Release 2.21, http://why.lri.fr
8. Hoare, C.A.R.: An axiomatic basis for computer programming. Communications of the ACM 12(10), 576–580, 583 (1969)
9. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: a theorem prover for program checking. J. ACM 52(3), 365–473 (2005)
10. Conchon, S., Contejean, E., Bobot, F., Lescuyer, S.: Alt-Ergo is an automatic theorem prover dedicated to program verification, Release 0.9., http://ergo.lri.fr
11. Microsoft Research: Z3 An Efficient SMT Solver, Release 2.4., http://research.microsoft.com/en-us/um/redmond/projects/z3
12. Barrett, C., Tinelli, C.: CVC3. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 298–302. Springer, Heidelberg (2007), http://cs.nyu.edu/acsys/cvc3
13. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions. In: EATCS. Texts in Theoretical Computer Science, Springer, Heidelberg (2004)
14. Shankar, N., Owre, S., Rushby, J.M., Stringer-Calvert, D.W.J.: PVS prover guide, http://pvs.csl.sri.com/doc/pvs-prover-guide.pdf
15. Nipkow, T., Paulson, L.C., Wenzel, M.T.:Isabelle/HOL — A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer, Heidelberg (2002)
16. Andrews, G.E.: The theory of partitions. Cambridge University Press, Cambridge (1984)
17. Jacobi, C.G.J.: De functionibus alternantibus earumque divisione per productum e differentiis elementorum conflatum. Journal für die reine und angewandte Mathematik (Crelles Journal) 22, 360–371 (1841); Reprinted in Gesammelten Werke III, G. Reimer, Berlin (1884)
18. Littlewood, D.E.: The Theory of Group Characters, 2nd edn. Oxford University Press, Oxford (1950)
19. Lascoux, A., Pragacz, P.: S-function series. J. Phys. A: Math. Gen. 21, 4105–4118 (1988)
20. Newell, M.J.: On the representations of the orthogonal and symplectic groups. In: Proc. Roy. Irish Acad., Section A: Mathematical and Physical Sciences, vol. 54, pp. 143–152 (1951)

Symbolic Domain Decomposition

Jacques Carette[1], Alan P. Sexton[2], Volker Sorge[2], and Stephen M. Watt[3]

[1]Department of Computing and Software, McMaster University
www.cas.mcmaster.ca/~carette
[2]School of Computer Science, University of Birmingham
www.cs.bham.ac.uk/~aps|~vxs
[3]Department of Computer Science, University of Western Ontario
www.csd.uwo.ca/~watt

Abstract. Decomposing the domain of a function into parts has many uses in mathematics. A domain may naturally be a union of pieces, a function may be defined by cases, or different boundary conditions may hold on different regions. For any particular problem the domain can be given explicitly, but when dealing with a family of problems given in terms of symbolic parameters, matters become more difficult. This article shows how hybrid sets, that is multisets allowing negative multiplicity, may be used to express symbolic domain decompositions in an efficient, elegant and uniform way, simplifying both computation and reasoning. We apply this theory to the arithmetic of piecewise functions and symbolic matrices and show how certain operations may be reduced from exponential to linear complexity.

1 Introduction

The goal of this paper is to develop general methods to work with domains having symbolically defined parts. The *raison d'être* of symbolic mathematical computation is to compute and reason about general expressions, rather than working with particular values valid only at specific points. Matters are simplest when variables range over a domain of interest and all expressions are valid over the entire domain. Sometimes it is useful to perform simplifications or other operations that are valid over part, but not all, of the domain. In this situation, software systems may or may not record the excluded region. But we are not always so fortunate to have this one-region situation. More generally, the domain of interest may be made up of several pieces with expressions taking different forms on different parts. Moreover, the demarcation of the parts may be defined symbolically. This paper explores how to represent such expressions concisely in a uniform way that simplifies computation and reasoning.

When we do arithmetic with piecewise functions defined on explicit partitions, we can do a mutual refinement of domain partitions to obtain regions that may each be handled uniformly. When the partitions are defined symbolically, however, we obtain a massive explosion of cases — for N binary operations on functions of k pieces there are k^N potential regions. We say "potential" regions because, of this large number, not all of the regions are in fact feasible. Furthermore, it is usually not possible to determine which regions are feasible and which are not. For example, the sum $\sum_{i=1}^{N} f_i(x)$ of the functions

$$f_i(x) = \begin{cases} 0 & \text{for } x < k_i, \\ A_i & \text{for } x \geqslant k_i. \end{cases}$$

has $\sum_{i=0}^{N} N!/i!$ possible orderings of the k_i, with each ordering having between 2 and $N+1$ regions. There is no ordering in which all 2^N regions are realized.

We take the view that it is generally preferable to have a single compact closed-form expression rather than a collection of cases, even if it means introducing some new operations. For example, we are perfectly satisfied using the Heaviside step function and giving the sum of the f_i as $\sum_{i=1}^{N} A_i H(x - k_i)$.

In this paper we show how hybrid sets, a variation on multisets allowing negative multiplicities, enable us to write elegant closed form expressions of the form we desire. This use of hybrid sets also allows us to define a generalised notion of partition, where symbolically defined parts are combined in more useful ways than the usual set operations. Our approach unifies and generalises a number of other techniques, such as the use of oriented regions for domains of integration.

Introducing new operators or generalising existing ones to write single closed form expressions is more than just a cosmetic re-arrangement. It allows one to perform arithmetic and simplifications on whole expressions, and to reason about the expression and about the regions themselves. It is already customary to do this for certain operators. For example, by defining $\int_a^b f\mathrm{d}x = -\int_b^a f\mathrm{d}x$, it becomes possible to write identities that hold universally. Then we have that, independent of the relative order of a, b and c, and subject to f being defined on the requisite domains,

$$\int_a^b f\mathrm{d}x = \int_a^c f\mathrm{d}x + \int_c^b f\mathrm{d}x. \tag{1}$$

With a little work, we can also generalise the integral formula to integrating over oriented subsets of $\mathbb{R}^n$. Some authors similarly adjust the definitions of other operators to obtain universally true statements. Similarly, Karr [8] defines the summation operator $\sum_{m \leqslant i < n}$ so that $\sum_{m \leqslant i < n} = -\sum_{n \leqslant i < m}$ when $n < m$. This allows equations such as the following to hold for any ordering of ℓ, m, n:

$$\sum_{m \leqslant i < n} \big(g(i+1) - g(i)\big) = g(n) - g(m), \qquad \sum_{\ell \leqslant i < n} f(i) = \sum_{\ell \leqslant i < m} f(i) + \sum_{m \leqslant i < n} f(i).$$

This paper formalises and extends these ideas in several ways, giving a generalised framework for domain partitions and piecewise defined functions. We first introduce some preliminaries and hybrid sets (§2) and then their generalisations to our notions of generalised partitions and hybrid functions (§3). We then present how we can decompose domains of hybrid functions to allow for the combination of their symbolically defined pieces (§4), before presenting some applications (§5) and discussing some concrete examples (§6).

2 Preliminaries

2.1 Partitions and Piecewise Functions

The domain of definition of various mathematical objects (functions, vectors, matrices, sequences, etc) may naturally be decomposed into a (disjoint) union of pieces where our object is then defined uniformly. When the pieces are given as an explicit union of provably disjoint sets which form a partition of the domain, the interpretation of a given expression is reasonably straightforward. More formally,

Notation 1. *We use the notation $\mathcal{C}_{i\in I}X_i$, or, more briefly, $\mathcal{C}_I X_i$ to describe a collection of elements $X_1, X_2, \ldots$, indexed by a set I. For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \ldots, n\}$.*

Definition 1. *A* partition *of a set U is a collection $\mathcal{C}_I P_i$ of pairwise disjoint sets such that $\bigcup_I P_i = U$.*

A partition, $\mathcal{C}_I P_i$, induces a total function $\mathcal{X} : U \to I$ which gives the index of P_* where each $u \in U$ sits. Piecewise expressions are then defined on top of a partition.

Definition 2. *A* piecewise expression *over a set U is a collection $\mathcal{C}_I(P_i, e_i)$ where $\mathcal{C}_I P_i$ is a partition of U and each e_i is an expression.*

Typically, each e_i contains a distinguished variable y which is interpreted to range over U.

Definition 3. *We say that $f : U \to S$ is a* piecewise-defined function *if we have a collection $\mathcal{C}_I(P_i, f_i)$ where $\mathcal{C}_I P_i$ is a partition of U, $\forall i \in I.\ f_i : P_i \to S$, and*

$$\forall x : U.\ f(x) = f_{\mathcal{X}(x)}(x).$$

We call $\mathcal{C}_I(P_i, f_i)$ the definition *of f, and each f_i a* piece *of f.*

It is important to note that piecewise-defined functions use their argument in two different ways: once *geometrically* by choosing a set P_i "over" which to work, and once *analytically* to evaluate a function f_i. The definitions above are straightforward generalisations of those found in [5].

Lastly, we have that arithmetic operations on piecewise-defined functions operate component-wise. Note that we will silently use the convention that operations defined on the codomain of a function are lifted pointwise to apply to functions, in other words $(f + g)(x) = f(x) + g(x)$.

Proposition 1. *Let $f, g : U \to S$ be two piecewise-defined functions on the same partition $\mathcal{C}_I P_i$ of U, with $\mathcal{C}_I f_i$ (respectively $\mathcal{C}_I g_i$) the collections of pieces of f (resp. g). Further, let $\star : S \times S \to S$. Then $\mathcal{C}_I(f_i \star g_i)$ is the collection of pieces of $f \star g$ over the partition $\mathcal{C}_I P_i$.*

Note how the partition is entirely untouched. This “separation of concerns” is what enables us to separate the issues of domain decompositions from arithmetic issues of piecewise-defined functions (and expressions).

To simplify our presentation, we introduce a domain restriction operation and a join combinator on (partial) functions, to allow us a more syntactic method of “building up” piecewise functions. These are quite similar to Kahl's table composition combinators [7].

Definition 4. *The restriction f^A of a function f to a domain specified by a set A is*

$$f^A(x) \ ::= \begin{cases} f(x) & \text{if } x \in A \\ \bot & \text{otherwise} \end{cases}$$

Definition 5. *The join, $f \oplus g$, of two (partial) functions f and g, is defined as*

$$(f \oplus g)(x) \ ::= \begin{cases} f(x) & \text{if f(x) is defined and g(x) is undefined} \\ g(x) & \text{if g(x) is defined and f(x) is undefined} \\ \bot & \text{otherwise} \end{cases}$$

This allows us to rewrite a piecewise-defined function f defined by $\mathcal{C}_I(P_i, f_i)$, in terms of its pieces as

$$f = f_1^{P_1} \oplus f_2^{P_2} \oplus \ldots \oplus f_n^{P_n}.$$

But our goal is to work with piecewise-defined functions where we have a *symbolic* partition. We need some new tools for this, which we will develop in the next two sections.

2.2 Hybrid Sets

We consider an extension of multisets, in which elements can occur multiple times, to *hybrid sets*, where the multiplicity of an element in a hybrid set can range over all of $\mathbb{Z}$, instead of just $\mathbb{N}_0$. Thus a hybrid set, over an underlying set U, is a mapping $U \to \mathbb{Z}$, i.e., it is an element of $\mathbb{Z}^U$. We use the following definition, adapted from [9]:

Definition 6. *Given a universe U, any function $H : U \to \mathbb{Z}$ is called a* hybrid set.

We can immediately define some useful vocabulary for working with hybrid sets.

Definition 7. *The value of $H(x)$ is said to be the* multiplicity *of the element x. If $H(x) \neq 0$, we say that x is a member of H and write $x \in H$; otherwise, we write $x \notin H$. The* support *of a hybrid set is the (non-hybrid) subset S of U where $s \in S \iff s \in H$; we will denote the support of H by* $\operatorname{supp} H$. *We (re)use $\emptyset$ to (also) denote the empty hybrid set, i.e. the hybrid set for whom all elements have multiplicity 0.*

Notation 2. *We use the notation $\{\!|x_1^{m_1}, x_2^{m_2}, \ldots|\!\}$ to describe the hybrid set containing elements x_1 with multiplicity m_1, x_2 with multiplicity m_2, etc. While* *our notation allows writing hybrid sets with multiple copies of the same element with different multiplicities, these denote the same hybrid set as that denoted by the normalised form with one copy of each element with a multiplicity which is the sum of the multiplicities of the copies of that element in the non-normalised form. Thus* $\{\!|a^2, b^1, a^{-3}, b^4|\!\} = \{\!|a^{-1}, b^5|\!\}$.

Set unions are usually defined by the boolean algebra structure (and more specifically, via $\vee$) of the membership relation. For hybrid set, this is replaced by arithmetic over $\mathbb{Z}$.

Definition 8. *We define the sum,* $A \oplus B$ *of two hybrid sets* A *and* B *over a universe* U*, to be their pointwise sum. That is* $(A \oplus B)(x) = A(x) + B(x)$ *for all* $x \in U$*. We similarly define their difference,* $A \ominus B$ *to be their pointwise difference, and their product,* $A \otimes B$ *to be their pointwise product. Let* $\ominus B$ *denote* $\emptyset \ominus B$*.*

In other words, we do not use operations $A \cup B$, $A \cap B$ and $A \setminus B$ for hybrid sets, but just $\oplus$, $\ominus$ and $\otimes$. We can easily establish some identities such as $(A \oplus B) \ominus A = B$, $A \ominus A = \emptyset$ and $A \oplus (\ominus B) = A \ominus B$ as these follow directly from $\mathbb{Z}$.

Putting all of this together, we get:

Proposition 2. $\mathbb{Z}^U$ *is a* $\mathbb{Z}$*-module.*

Proof. The abelian group structure is given by $(\mathbb{Z}^U, \oplus, \ominus, \emptyset)$, and $\mathbb{Z}$ acts on hybrid sets by $nH = u \mapsto n \cdot H(u)$.

We need two more technical definitions, which will be useful later.

Definition 9. *We say that two hybrid sets* A *and* B *are* disjoint *if* $A \otimes B = \emptyset$*.*

Definition 10. *We call a hybrid set* reducible *if all its members have multiplicity* 1*. We define a reduction function,* $\mathcal{R}(\cdot)$*, on reducible hybrid sets that returns the (normal) set of members of the hybrid set.*

We should note that these hybrid sets (sometimes also called generalised sets) have been studied before. Hailperin [6] makes the case that Boole [3] actually started from hybrid sets for his algebraization of logic, but restricted himself to nilpotent solutions of the resulting equations, which then correspond closely to our modern notion of Boolean algebra. Whitney wrote two nice papers [13,14] taking up the theme of algebraising logic via characteristic functions. He does allow arbitrary multiplicities, and derives some nice normal forms for certain kinds of partitions, in a way foreshadowing some of our own results (see §3.2 and §4). Blizard [1] focuses on multisets (disallowing negative multiplicities) but has an extensive bibliography of related works, several of which being on (mechanised) theorem proving; he then formalised sets with negative membership in [2]. Blizard concentrates on concepts of union and intersection which closely resemble those of normal set theory, although he does also define the sum union (but not other related concepts). Burgin [4] lists several more works on hybrid sets, some reaching back to the early middle ages. Syropoulos [12] gives a very readable introduction to both multisets and hybrid sets.

3 Generalisations

We now revisit a few basic mathematical constructs and show how they may be modified to work with hybrid sets. This will provide the machinery that we need for symbolic domain decomposition. First, we will examine the notion of a hybrid function on a domain. We then show how sets and hybrid sets may be decomposed using a notion of generalised partitions — an extension of partitions to the hybrid set case. We then address the practical issue of how to make two hybrid partitions compatible by constructing a common refinement. Finally, when working with functions defined over hybrid partitions, we need some way to compute values. Over any given point in the domain, we need to know which functions must be evaluated in computing the final value, which is rather complex for hybrid functions over generalised partitions. For this task, we introduce the notion of pseudo-functions. When expressions on generalised partitions are evaluated, these pseudo-functions avoid evaluating at places where the functions are undefined or where the values are not needed. This allows us to deal with the cases, as in equation (1), where component functions are not defined on some parts of the domain decomposition but any application of the function in those places would anyway have multiplicity zero (*i.e.* not be used).

3.1 Functions of Hybrid Domain

It turns out that a useful definition of a "function" involving hybrid sets is not entirely straightforward. Defining its graph is easiest. The underlying intuition is that we capture the restriction of a function to a domain through the multiplicities of the elements of the function graph in a hybrid set. The hybrid set of a single element of the function graph for element x in U is of the form $\{\!|(x, f(x))^1|\!\}$. Therefore the scalar multiplication of that set by the multiplicity of x in A will impose the appropriate restriction. We use our function restriction notation of Def. (4) only for this hybrid version of function restriction henceforth.

Definition 11. *Let A be a hybrid set over U, $B \subseteq U$, S a set and $f : B \to S$ a (total-on-B) function. A* hybrid function $f^A : U \times S \to \mathbb{Z}$ *is defined by*

$$f^A = \bigoplus_{x \in B} A(x) \{\!|(x, f(x))^1|\!\}$$

Note how the hybrid set $\mathbb{Z}$-module structure automatically takes care of restricting the sum over the support of A. Caution: some hybrid sets do not form a hybrid function (for example $\{\!|(1, 1)^1, (1, 2)^1|\!\}$ is not a hybrid function).

Our definitions work just as well with partial functions as with total functions. But if for a hybrid function f^F, f is undefined at some point in the support of F, f^F will not be defined at that point either. So, without loss of generality, we can always restrict F to where f is defined. For the remainder of this paper, we shall assume this, i.e. whenever we write a hybrid function f^F, f is total over $\operatorname{supp} F$.

Definition 12. *We call a hybrid function* f^H reducible *if the hybrid set* H *is reducible. We extend* $\mathcal{R}(\cdot)$ *in this case by*

$$\mathcal{R}(f^H)(x) = \begin{cases} f(x) & \textit{if } H(x) = 1 \\ \bot & \textit{if } H(x) = 0 \end{cases}$$

We can generalise the join combinator to hybrid sets. This definition is quite central to "making things work".

Definition 13. *The join,* $f^F \varobar g^G$*, of two hybrid functions* f^F *and* g^G *(with codomain* B*), gives a* hybrid relation*, a subset of* $U \times B \times \mathbb{Z}$ *given by*

$$f^F \varobar g^G \ ::= \ f^F \oplus g^G$$

This is a rather "dangerous" definition, as it moves us from the land of functions to that of relations. In other words, it is quite possible that $f^F \varobar g^G$ restricted to $U \times B$ is no longer the graph of a function, but the graph of a relation. But this extra generality will be quite useful for us, although we will have to prove that in the cases which interest us, the resulting hybrid relations are in fact (reducible) hybrid functions.

Theorem 1. *Let* A*,* B *be hybrid sets over* U*,* S *an arbitrary set, and* $f : U \to S$ *a total function. Then*

1. $\mathcal{R}(f^{\emptyset})$ *is the empty function,*
2. $f^A \varobar f^A = f^{2A}$
3. $f^A \varobar f^B = f^{A \oplus B}$*, and thus a hybrid function,*
4. *For* $g : U \to S$ *another total function, then* $f^A \varobar g^B = (f \varobar g)^{A \oplus B}$ *if and only if* $A \oplus B = \emptyset$ *(where* $f \varobar g$ *is the join of regular functions).*
5. *Let* H_1, H_2 *be hybrid sets, with* $\operatorname{supp} H_1$ *and* $\operatorname{supp} H_2$ *disjoint,* $f_1 : \operatorname{supp} H_1 \to S$ *and* $f_2 : \operatorname{supp} H_2 \to S$*, then* $f_1^{H_1} \varobar f_2^{H_2} = (f_1 \varobar f_2)^{H_1 \oplus H_2}$

The proofs are omitted, and follow straightforwardly from the definitions. Note the strong dichotomy between (3) and (4), which comes from the fact that the non-hybrid $\varobar$ is designed to work with functions defined over separate regions.

3.2 Generalised Partitions

Theorem 1 tells us that some collections of hybrid sets are better than others. Being disjoint is much too strong a property. Nicely, for hybrid sets, partitions easily generalise in useful ways.

Definition 14. *We define a generalised partition of a (hybrid) set,* P*, to be a finite collection of hybrid sets,* $\mathcal{C}_{[n]}P_i$*, such that* $P_1 \oplus P_2 \oplus \ldots \oplus P_n = P$

All set partitions of a set are also generalised partitions. Conversely, a generalised partition of a reducible set is a set partition if and only if each generalised partition element is reducible.

Remark 1. We have lifted the *disjointness* condition on partitions. For something to be called a partition of P, it is necessary that the result be equal to P. Still, P belongs to a larger universe U, and a generalised partition's pieces range over U. As long as, in the end, all elements of $U \setminus P$ have multiplicity 0, we get a generalised partition. In this way, we have also lifted the *coverage* condition.

Proposition 3. *For any generalised partition $\mathcal{C}_{[n]}P_i$ of a hybrid set P over U, arbitrary set S, and any function $f : \operatorname{supp} P \to S$,*

$$f^P = f^{P_1} \oplus f^{P_2} \oplus \ldots \oplus f^{P_n} = f^{P_1 \oplus P_2 \oplus \ldots \oplus P_n}$$

is a hybrid function.

For brevity, we sometimes write f^P for either of the right-hand side expressions above. When we want to join different functions over a partition and still get a function, we have to be careful and ensure we are joining "compatible" functions.

Definition 15. *Let P_1, P_2 be a generalised partition of a hybrid set P over U, S an arbitrary set and $f^{P_1} : P_1 \to S$, $g^{P_2} : P_2 \to S$ hybrid functions. We say that P_1, P_2 is a* compatible *partition for f, g if $f(x) = g(x)$ for all $x \in \operatorname{supp} P_1 \cap \operatorname{supp} P_2$.*

Theorem 2. *Using the same notation as above, $f^{P_1} \oplus g^{P_2}$ is a hybrid function if and only if P_1, P_2 is a compatible partition for f, g.*

It is important to note that although $\oplus$ is an associative, commutative operation, the notion of compatibility, while commutative, does not lift to a simple associative condition.

Remark 2. Some of our computations will purposefully use *incompatible* partitions. Note that

$$(f^U \oplus g^U) \oplus g^{\ominus U} = f^U \oplus (g^U \oplus g^{\ominus U}) = f^U \oplus g^{\emptyset} = f^U$$

but that $f^U \oplus g^U$ is in general a hybrid relation, yet the final result is a hybrid function whenever f^U is. We will "design" our hybrid partitions with this feature in mind.

3.3 Refinement

To do arithmetic with hybrid functions, we first need the notion of a refinement and a common refinement. This is similar to the treatment of [5] for piecewise functions.

Definition 16. *A* refinement *of a generalised partition $\mathcal{C}_I P_i$ of P is another generalised partition $\mathcal{C}_J Q_j$ (of another hybrid set Q not necessarily equal to P) such that for every $i \in I$ there exists a sub-collection Q_{j_k} of Q_j such that $P_i = Q_{j_1} \oplus Q_{j_2} \oplus \ldots \oplus Q_{j_m}$. A* common refinement *of a set of generalised partitions is a generalised partition that is simultaneously a refinement to every partition in the set.*

A refinement in this sense may well seem "larger" than the original partition, as in the next example.

Example 1. Let the interval $P = [0,1]$, seen as $\{\!|P^1|\!\}$, and $I_1 = [-1,0)$, $I_2 = (1,2]$, $I_3 = [-1,2]$, then $Q = \{\!|I_1^{-1}, I_2^{-1}, I_3^1|\!\}$ is a refinement of P.

Definition 17. *A refinement is called* strict *if, for each generalised partition being refined, the support of the associated sub-collection is equal to the support of the generalised partition it refines.*

Example 2. $\{\!|[0,1]^1|\!\}, \{\!|(1,2]^1|\!\}, \{\!|(2,3]^1|\!\}$ is a common strict refinement of the two (trivial) hybrid partitions $\{\!|[0,2]^1|\!\}$ and $\{\!|(1,3]^1|\!\}$.

3.4 Pseudo-functions and Pseudo-relations

As seen in the example above, a refinement may "spill over" the original domain, so that if we look at a hybrid function f^P where the underlying f is defined exactly on (the support of) P, f^Q evaluated "pointwise" will not make sense. Nevertheless, we want $f^P = f^Q$. To achieve this, we apply the lambda-lifting trick already used in [5].

Definition 18. *Using the same notation as in Def. 11, we define a* pseudo-function $\tilde{f}^A$ *as*

$$\tilde{f}^A = \bigoplus_{x \in B} A(x) \{\!|(x,f)^1|\!\}$$

i.e. as a member of $U \times (U \to S) \to \mathbb{Z}$*. A* pseudo-relation *is defined similarly. The* evaluation *of a pseudo-function (resp. relation) is defined by mapping each point* $(x,f)^k$ *to* $(x, f(x))^k$*.*

The usefulness of pseudo-functions comes from the following property.

Proposition 4. *For all refinements Q of the generalised partition P, $\tilde{f}^P = \tilde{f}^Q$.*

In other words, even though the pieces of Q might "spill over", if we first "simplify" $\tilde{f}^Q$ by performing $\bigoplus_i Q_i$ to get P, we get a result $\tilde{f}^P$ which can then be safely evaluated. We will elide the ˜ to lighten the notation whenever this would not lead to confusion.

Another useful property of pseudo-functions is that in some cases we can simplify them, regardless of what the underlying function does.

Proposition 5. $\tilde{f}^P \oplus \tilde{g}^Q \oplus \tilde{g}^{\ominus Q} = \tilde{f}^P$

One of the chief advantages of pseudo-functions is that we can do some symbolic manipulations of expressions in terms of these functions as if they were defined on a much larger domain, as long as the eventual term we evaluate does not involve any of these "virtual" terms.

To aid in such computations, when we have pseudo-functions $\tilde{f}^P$ and $\tilde{g}^P$, with $f, g : \operatorname{supp} P \to S$, and some binary operation $\star : S \times S \to S$, we will allow ourselves to write expressions such as $\tilde{f}^P \star \tilde{g}^P$ in the induced term algebra over pseudo-functions. As usual, we lift evaluation pointwise, $(\tilde{f} \star \tilde{g})(x) = f(x) \star g(x)$. Furthermore we say that $\tilde{f} = \tilde{g}$ over a set $\operatorname{supp} P$ whenever $\forall x \in \operatorname{supp} P. f(x) = g(x)$. In other words, we use extensional equality for the intensional terms $\tilde{f}$ and $\tilde{g}$. This means that properties like commutativity, associativity and having inverses lift to the term algebra. As an example, we have that

Proposition 6. *If $\star : S \times S \to S$ is associative and commutative then*

$$\tilde{g}^Q \star \tilde{f}^P \star \tilde{g}^{\ominus Q} = \tilde{f}^P$$

4 Hybrid Domain Decomposition

We now have all the ideas necessary to decompose hybrid domains. An elegant consequence of the formalism is that it allows us to use linear algebra to construct the partitions that we require.

Let $\mathcal{C}_{[n]} A_i$ and $\mathcal{C}_{[m]} B_j$, be generalised partitions of U. We want to find a generalised partition of U that is a common strict refinement of A_i and B_j and has minimal cardinality. The cardinality restriction is to minimise the number of terms required for a symbolic representation of the resulting domain decomposition. Thus we want to choose a minimal generalised partition, $\mathcal{C}_I P_i$ of U such that, in the $\mathbb{Z}$-module of hybrid sets and for some integers $a_{i,j}$, $b_{i,j}$ we have $\bigoplus_i P_i = U$ and $\forall i : 1..n.\ \bigoplus_j a_{i,j} P_j = A_i$ and $\forall j : 1..m.\ \bigoplus_i b_{i,j} P_i = B_j$.

Since this forms a system of $n + m + 1$ simultaneous equations, of which only $n + m - 1$ can be independent, because A and B are, separately, partitions of U, we need that number of independent variables to solve the system. Thus the cardinality of the minimal partition is $n + m - 1$ in the general case, and can only be smaller in specific cases if there are some extra dependencies among $U, A_1, \ldots, A_{n-1}, B_1, \ldots, B_{m-1}$.

This result generalises to a decomposition of U into a minimal generalised partition that is a common strict refinement of r generalised partitions of cardinality $n_1, \ldots n_r$ respectively: The minimal partition required, assuming full independence of the individual domain partitions, has cardinality

$$\left(\sum_{i=1}^{r} n_i\right) + 1 - r$$

If all the individual partitions have the same cardinality, n, this reduces to $r(n - 1) + 1$.

We can automatically compute suitable minimal strict refinement partitions U as follows. If we remove the equations for A_n and B_m from the system of equations in order to get an independent set of simultaneous equations, we get $n + m - 1$ equations that can be written as a linear system in the $\mathbb{Z}$-module:

$$C \cdot \begin{pmatrix} P_1 \\ \vdots \\ P_{n+m-1} \end{pmatrix} = \begin{pmatrix} U \\ A_1 \\ \vdots \\ A_{n-1} \\ B_1 \\ \vdots \\ B_{m-1} \end{pmatrix} \quad \text{where } C = \begin{pmatrix} 1 & 1 & \dots & 1 \\ a_{1,1} & a_{1,2} & \dots & a_{1,n+m-1} \\ \vdots & & & \vdots \\ a_{n-1,1} & a_{n-1,2} & \dots & a_{n-1,n+m-1} \\ b_{1,1} & b_{1,2} & \dots & b_{1,n+m-1} \\ \vdots & & & \vdots \\ b_{m-1,1} & b_{m-1,2} & \dots & b_{m-1,n+m-1} \end{pmatrix}$$

Note that C is an integer matrix. Further, the partition choice matrix, C, must be invertible and C^{-1} must be an integer matrix so that we obtain integral partitions of each domain piece with respect to our partition P of U. An integer matrix is invertible and has an integer matrix inverse if and only if it has a determinant of $+1$ or -1. Hence our problem of constructing an appropriate partition reduces to choosing an integer matrix of the form of C such that its determinant is ± 1. Finally, note that this directly generalises to an arbitrary number of piecewise functions, each of an arbitrary number of pieces.

If we restrict ourselves to triangular matrices, we can choose C to be any integer triangular matrix (upper because the first row of C is all 1s) for which the product of the diagonal elements is 1. Again, a simple way to do this is to choose C to be all 1s along the top row, 1s along the diagonal and 0 everywhere else. Another possibility is all 1s in the whole upper triangle.

For example, two suitable choice matrices with their inverses are

$$\begin{pmatrix} 1 & \dots & \dots & \dots & 1 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & -1 & \dots & \dots & -1 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & \dots & \dots & 1 \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & -1 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -1 \\ 0 & \dots & \dots & 0 & 1 \end{pmatrix}$$

5 Applications

5.1 Arithmetic on Piecewise Functions

We are now ready to generalise the arithmetic properties (see prop. 1) of hybrid functions and pseudo-functions.

Proposition 7. *Let $\mathcal{C}_{[n]}P_i$ be a partition of P, $f^P = f_1^{P_1} \oplus \dots \oplus f_n^{P_n}$ and $g^P = g_1^{P_1} \oplus \dots \oplus g_n^{P_n}$ be two hybrid functions on $P \to S$. Let $\star : (S \times S) \to S$, then for all $x \in \operatorname{supp} P$,*

$$f^P(x) \star g^P(x) = (f_1(x) \star g_1(x))^{P_1} \oplus \dots \oplus (f_n(x) \star g_n(x))^{P_n}.$$

Note how we can apply the above proposition to $f^F \star g^G$, by first restricting F and G to be over a common support (as $x \star \bot = \bot \star y = \bot$), then taking a common *strict* refinement of (the restricted) F and G.

We would like to lift this strictness condition. We can almost do this with pseudo-functions – and with the help of a marked $\oplus$, we can.

Definition 19. *We can* mark *a $\oplus$ with a binary operation $\star : S \times S \to S$, denoted $\oplus^\star$. We define evaluation of $\oplus^\star$ by*

$$(\tilde{f}^F \oplus^\star \tilde{g}^G)(x) = (F(x) + G(x))\left\{\left|(x, (\tilde{f} \star \tilde{g})(x))^1\right|\right\}$$

This operation clearly inherits properties of $\star$ like commutativity, associativity and invertibility. Unlike $\oplus$, $\oplus^\star$ will always result in a (pseudo) function.

Proposition 8. *Let $P_* = \mathcal{C}_{[n]}P_i$ be a partition of P, $Q_* = \mathcal{C}_{[m]}Q_j$ another partition of P, and $R_* = \mathcal{C}_K R_k$ a common refinement of P_* and Q_*. Let $\tilde{f}^P = f_1^{P_1} \oplus \ldots \oplus f_n^{P_n}$ and $\tilde{g}^P = g_1^{Q_1} \oplus \ldots \oplus g_m^{Q_m}$ be two* pseudo *functions with codomain S. Let $\star : (S \times S) \to S$, be associative and commutative, then $\star$ distributes over the partition R_* in terms of $\oplus^\star$. By the results of section 4, we can always choose R_* to be*

$$P_1 \oplus \ldots \oplus P_{n-1} \oplus Q_1 \oplus \ldots \oplus Q_{m-1} \oplus (U \ominus (P_1 \oplus \ldots \oplus P_{n-1} \oplus Q_1 \oplus \ldots \oplus Q_{m-1}))$$

Example 3. Let $A_1 = [0, a), A_2 = [0, 1] \setminus A_1, B_1 = [0, b), B_2 = [0, 1] \setminus B_1$, all seen as hybrid sets. Let

$$f(x) = \begin{cases} 2 & 0 \le x < a \\ 0 & a \le x < 1 \end{cases} \qquad \text{and} \qquad g(x) = \begin{cases} 5 & 0 \le x < b \\ 7 & b \le x < 1 \end{cases}$$

We choose the hybrid (symbolic!) partition $A_1, B_1 \ominus A_1, B_2$, which simultaneously refines both. Then after a few computations we get

$$f * g = \{\!|2 * 5|\!\}^{A_1} \oplus^* \{\!|2 * 5|\!\}^{B_1 \ominus A_1} \oplus^* \{\!|0 * 7|\!\}^{B_2} \tag{2}$$

regardless of whether $a < b$ or $a \ge b$; in fact either (or both) could be outside of $[0, 1)$ and the result, interpreted as a hybrid function, are still correct. Moving from one choice of partition to another is done by undoing the distribution, performing the change of partition, and using commutativity and associativity to regroup like terms. Note that we should have written $2*5$ as $(x \mapsto 2)\tilde{*}(x \mapsto 5)$, but we chose the above for greater clarity.

It should be very clear that expressions such as equation 2 are a formal representation of a piecewise function, and need to be *interpreted* properly in each context.

5.2 Identities for Invertible Operators

When the binary operation we use is invertible, we can perform the operations at any time, as operands may later be removed from a cumulative result by applying their inverses. This may lead to considerable efficiency improvements; even though values and their inverses are cancelled in the calculation (leading to no net effect) this may be more efficient than retaining an "un-evaluable" expression as a pseudo-function.

Proposition 9. *Let f^P be a hybrid function over S where $(S, \star)$ has the structure of an Abelian group (where we will use e for the unit and $-$ for the inverse), then for all $x \in \operatorname{supp} P$,*

$$(f^P \star f^{-P})(x) = (f^P \star (-f)^P)(x) = P(x)\{\!|(x, e)^1|\!\}$$

where $-f$ denotes $x \mapsto -f(x)$.

Both equalities follow readily from the definitions.

5.3 Identities for Linear Operators

Definition 20. *For a linear operator L, and f^P a hybrid function,*

$$L(f^P) \ ::= \ L(x \mapsto P(x) \cdot f(x))$$

Note $L(f^P)$ may not be defined even when $L(f)$ is. If the multiplicity function $P(x)$ is uniformly bounded, then it will exist.

Proposition 10. *Let f^P be a hybrid function, $\mathcal{C}_{[n]} P_i$ a partition of P such that each $P_i(x)$ is uniformly bounded, then*

$$L(f^P) = \sum_{i=1}^{n} L(f^{P_i})$$

The above is the fundamental reason why, under Karr's definition, the summation identities of the introduction hold.

Corollary 1. *For all total functions $f : \mathbb{Z} \to G$ with G an Abelian group, and all $\ell, m, n \in \mathbb{Z}$,*

$$\sum_{\ell \leqslant i < n} f(i) = \sum_{\ell \leqslant i < m} f(i) + \sum_{m \leqslant i < n} f(i).$$

6 Examples

We present two examples of the application of hybrid functions to symbolic computation problems. The first example is concerned with the arithmetic of symbolic matrices, the second presents the idea of merging symbolic spline functions.

6.1 Matrix Addition

Earlier work [10,11] introduced The idea of support functions has been introduced previously to represent symbolic matrices — matrices given in terms of symbolic regions with underspecified elements and symbolic dimensions — and defined arithmetic operations between them. The paper [10] presented a support function based on half-plane constraints that enables full arithmetic, but that suffered from a combinatorial explosion in the number of terms needed to express sum or product matrices. The paper [11] moved to a support function based on interval addition that automatically dealt with cancellation for negative intervals. While this avoided the combinatorial explosion, the approach was restricted to certain types of regions and could not be easily generalised to matrix multiplication. Hybrid functions and generalised partitions solve both of these problems simultaneously. We demonstrate this with the example of matrix addition of two 2×2 symbolic block matrices. Let

$$M_1 = \begin{pmatrix} A_1 & B_1 \\ C_1 & D_1 \end{pmatrix}, \qquad M_2 = \begin{pmatrix} A_2 & B_2 \\ C_2 & D_2 \end{pmatrix}$$

where M_1 and M_2 are $n \times m$ matrices, A_1 and A_2 are of dimensions $h_1 \times k_1$ and $h_2 \times k_2$ respectively, $n, m, h_1, h_2, k_1, k_2 \in \mathbb{N}_0$.

Let $U = \{(i, j) \mid 1 \leqslant i \leqslant n \wedge 1 \leqslant j \leqslant m\}$ be the set of all cell points in the matrices. We define the region occupied by a matrix block similarly, and refer both to a matrix block and to the region it occupies by the same name, relying on context to distinguish them. We can thus write

$$M_1 = A_1^{A_1} \oplus B_1^{B_1} \oplus C_1^{C_1} \oplus D_1^{D_1}, \qquad M_2 = A_2^{A_2} \oplus B_2^{B_2} \oplus C_2^{C_2} \oplus D_2^{D_2} \tag{3}$$

To calculate $M_1 + M_2$ we: (1) Choose a suitable generalised partition P_* of U. (2) Rewrite each block of each matrix into terms restricted to the chosen partition. (3) Substitute into the expressions for the matrices. (4) Add the two matrices region-wise. As established in the section 4, the maximal number of partitions required in our case is $4+4-1 = 7$. We therefore choose 6 independent regions to be $A_1, B_1, C_1, A_2, B_2, C_2$ and obtain the seventh, P_1, by subtracting all other regions from the universe U,

$$P_1 = U \ominus (A_1 \oplus B_1 \oplus C_1 \oplus A_2 \oplus B_2 \oplus C_2). \tag{4}$$

We can then express regions D_1 and D_2 in terms of P_1: $D_1 = U \ominus (A_1 \oplus B_1 \oplus C_1) = P_1 \oplus A_2 \oplus B_2 \oplus C_2$, and $D_2 = P_1 \oplus A_1 \oplus B_1 \oplus C_1$.

Rewriting M_1, M_2 from Eq. (3), we get:

$$\begin{aligned} M_1 &= A_1^{A_1} \oplus B_1^{B_1} \oplus C_1^{C_1} \oplus D_1^{P_1 \oplus A_2 \oplus B_2 \oplus C_2} \\ &= D_1^{P_1} \oplus A_1^{A_1} \oplus B_1^{B_1} \oplus C_1^{C_1} \oplus D_1^{A_2} \oplus D_1^{B_2} \oplus D_1^{C_2} \end{aligned}$$

$$M_2 = D_2^{P_1} \oplus D_2^{A_1} \oplus D_2^{B_1} \oplus D_2^{C_1} \oplus A_2^{A_2} \oplus B_2^{B_2} \oplus C_2^{C_2}$$

This lets us express the sum of the two matrices as the following pseudo-function:

$$\begin{aligned} M_1 + M_2 = {} & (D_1 + D_2)^{P_1} \oplus^+ (A_1 + D_2)^{A_1} \oplus^+ (B_1 + D_2)^{B_1} \oplus^+ (C_1 + D_2)^{C_1} \\ & \oplus^+ (D_1 + A_2)^{A_2} \oplus^+ (D_1 + B_2)^{B_2} \oplus^+ (D_1 + C_2)^{C_2} \end{aligned} \tag{5}$$

Observe that the seven terms of the hybrid function fully capture the result of the matrix addition independent of the order of the symbolic dimensions h_1, h_2, k_1, k_2 of the original blocks. We demonstrate this by evaluating the function for a couple of concrete points in the sum matrix.

First let $h_1 < h_2$ and choose a concrete value of a cell (i, j) where $h_1 < i \leq h_2$ and $1 \leq j < k_1, k_2$. The point should therefore be in a region composed of elements from B_1 and A_2. Instantiating the multiplicities in equation (5) verifies this. Observe that indeed the only regions with multiplicity 1 are $B_1 = \{(i,j) \mid h_1 \leqslant i \leqslant n \wedge 1 \leqslant j \leqslant k_1\}$ and $A_2 = \{(i,j) \mid 1 \leqslant i \leqslant h_2 \wedge 1 \leqslant j \leqslant k_2\}$ whereas the multiplicities for A_1, B_2, C_1, C_2 are all 0. Furthermore, we can compute the multiplicity for P_1 using equation (4). Since the multiplicity of the universe U is always 1 — every element is in this partition — we get $1 - (0 + 1 + 0 + 1 + 0 + 0) = -1$. This then yields the computation below, which confirms that our element is indeed in the anticipated region (where we write region-wise sets $\{\!|(D_1 + D_2)^{-1}|\!\}$ as $(D_1 + D_2)^{-1}$ to alleviate notation)

$$(D_1 + D_2)^{-1} \oplus^+ (B_1 + D_2)^1 \oplus^+ (D_1 + A_2)^1 = B_1^1 \oplus^+ A_2^1 = B_1 + A_2$$

Now assume that the order of the symbolic dimensions for blocks A_1, A_2 is reversed and we have $h_2 < h_1$. If we now compute the value of a cell with (i, j) with $h_2 < i \leq h_1$ and $1 \leq j < k_1, k_2$, we get 0 multiplicity for regions B_1, A_2, but instead multiplicity 1 for A_1, B_2. Again the multiplicity for P_1 is -1 and equation (5) will yield that our cell is in the $A_1 + B_2$ region.

As a final example we compute cell (i, j) with $h_1, h_2 < i \leq n$ and $k_1, k_2 < j \leq m$, which is a point from D_1 and D_2. This time equation (5) simplifies even more quickly, as the multiplicities for $A_1, A_2, B_1, B_2, C_1, C_2$ are all 0 and we only have to consider the multiplicity for P_1 which is 1 with the same considerations as above, yielding $D_1 + D_2$ as the only term that does not vanish.

6.2 Symbolic Spline Interpolation

Numerical computation and manipulation of splines is well understood. Here we show how hybrid domain decomposition can be used to support the previously unexplored symbolic manipulation of splines.

Let $a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$ be a partition of the interval $[a, b] \subset \mathbb{R}$. We call the x_i knots and assume that for each knot we have a corresponding value y_i. Then we can define a spline function S over $[a, b]$ piecewise as

$$S(x) = \begin{cases} S_0(x) & x \in [x_0, x_1] \\ \vdots & \vdots \\ S_{n-1}(x) & x \in [x_{n-1}, x_n] \end{cases} \tag{6}$$

While spline interpolation is traditionally defined for numerical values of the x_i, y_i pairs, we will now define a symbolic spline function. Let $a = c_0 < c_1 < \cdots < c_{n-1} < c_n = b$ be symbolic or "abstract" knots with associated symbolic values $d_0, d_1, \ldots, d_n$, where a d_i is generally given as a function in c_i. We define spline segments $S_{c_i,c_{i+1}}(x)$ for $i = 0, \ldots, n-1$. That is, the segments are parameterised with respect to two knots and their values. Let $P = P_1 \oplus \ldots \oplus P_n$ with $P_i = [c_{i-1}, c_i]$ be a generalised partition. We then define a symbolic spline function $S(x) = S_{c_0,c_1}^{P_1}(x) \oplus \cdots \oplus S_{c_{n-1},c_n}(x)^{P_n}$. For clarity we often omit the (x) part of the term.

Define the merge of two spline segments $S^P_{a,b} \bowtie S^Q_{a',b'}$ to be $S^{P\otimes Q}_{max(a,a'),min(b,b')}$. Clearly this merge will be empty if the two segments do not overlap, otherwise it will be the smallest possible spline for the overlapping interval of the segments.

Now let $P_1 \oplus \cdots \oplus P_n$ and $Q_1 \oplus \cdots \oplus Q_m$ be two generalised partitions of the interval $[a,b]$ and let $S = S^{P_1}_{c_0,c_1} \varobar \cdots \varobar S^{P_n}_{c_{n-1},c_n}$ and $T = T^{Q_1}_{d_0,d_1} \varobar \cdots \varobar T^{Q_m}_{d_{m-1},d_m}$ be two symbolic splines. Observe that the d_i here are knots and not knot values. We can then define the merge $S \bowtie T$ as a binary operation on two hybrid functions as given above in proposition 8.

We observe the merge operation using a simple example. Let $P = P_1 \oplus P_2$ and $Q = Q_1 \oplus Q_2$ be the generalised partitions $a < c < b$ and $a < d < b$ of our universe $[a,b]$, respectively. Let $S = S^{P_1}_{a,c} \varobar S^{P_2}_{c,b}$ and $T = T^{Q_1}_{a,d} \varobar T^{Q_2}_{d,b}$ be two symbolic splines. We choose a common refinement as $P_1, Q_1, R = U \ominus (P_1 \oplus Q_1)$. We can then write $S = S^{P_1}_{a,c} \varobar S^{R\oplus Q_1}_{c,b} = S^{P_1}_{a,c} \varobar S^{R}_{c,b} \varobar S^{Q_1}_{c,b}$ and similarly $T = T^{Q_1}_{a,d} \varobar T^{P_1}_{d,b} \varobar T^{R}_{d,b}$. When we merge both symbolic splines we get

$$S \bowtie T = (S_{a,c} \bowtie T_{d,b})^{P_1} \varobar^{\bowtie} (S_{c,b} \bowtie T_{a,d})^{Q_1} \varobar^{\bowtie} (S_{c,b} \bowtie T_{d,b})^{R} \tag{7}$$

If we now fix the order of our symbolic knots to be $a < c < d < b$ we can evaluate the three components of our spline. First let $a \le x \le c$, which means $P_1 = Q_1 = 1$ and $R = -1$ and (7) evaluates to $S_{a,c} \bowtie T_{a,d}$ which yields a spline between knots a, c. Similarly the other two segments evaluate $S_{c,b} \bowtie T_{a,d}$ and $S_{c,b} \bowtie T_{d,b}$, which yields splines for the intervals $[c,d]$ and $[d,b]$, respectively.

7 Conclusion

We have presented a framework of generalised partitions and domain decomposition based on hybrid sets. This framework has a number of pleasing properties and unifies a number of *ad hoc* notions in common use. More importantly for our purposes, this representation allows easy manipulation of and reasoning about partitions whose pieces are defined symbolically. These specialise correctly for all choices of parameters by negative and positive multiplicities cancelling as needed. The representation (see for example equation (2)) needs to be carefully interpreted, as out of context simplification can result in meaningless results. But if the rules we lay out are followed, our compact representation effectively allows one to compute with very general piecewise-defined functions.

Although a number of previous authors have studied hybrid sets, our study of functions over hybrid sets, generalised partitions, the superposition $\varobar$ and marked superposition $\varobar^\star$ operators appear to all be new. Our applications to computation and reasoning are certainly new.

There remain several intriguing directions for future work. These include normal forms for piecewise functions defined on hybrid partitions, simplification of intermediate expressions involving linear operators or inverse elements and creating partition schemes from algebraic specialisation properties, *e.g.* in Cylindrical Algebraic Decomposition. We have implemented a prototype Maple package for hybrid sets and hybrid functions, but a more general implementation would be of interest.

References

1. Blizard, W.D.: Multiset theory. Notre Dame Journal of Formal Logic 30(1), 36–66 (Winter 1989)
2. Blizard, W.D.: Negative membership. Notre Dame Journal of Formal Logic 31(3), 346–368 (1990)
3. Boole, G.: An investigation of the laws of thought, on which are founded the mathematical theories of logic and probabilities. Walton and Maberly, London (1854), http://www.gutenberg.org/etext/15114
4. Burgin, M.S.: Concept of multisets in cybernetics. Cybernetics and Systems Analysis 28(3), 371–469 (1992)
5. Carette, J.: A canonical form for piecewise defined functions. In: Proc. of ISSAC 2007, pp. 77–84. ACM Press, New York (2007)
6. Hailperin, T.: Boole's Logic and Probability, 1st/2nd edn. North-Holland Publishing Company, Amsterdam (1976)
7. Kahl, W.: Compositional syntax and semantics of tables. SQRL Report 15, Department of Computing and Software, McMaster University (2003), http://www.cas.mcmaster.ca/sqrl/sqrl_reports.html
8. Karr, M.: Summation in finite terms. J. ACM 28(2), 305–350 (1981)
9. Loeb, D.: Sets with a negative number of elements. Advances in Mathematics 91(1), 64–74 (1992)
10. Sexton, A.P., Sorge, V., Watt, S.M.: Computing with abstract matrix structures. In: Proc. of ISSAC 2009, pp. 325–332. ACM Press, New York (2009)
11. Sexton, A.P., Sorge, V., Watt, S.M.: Reasoning with generic cases in the arithmetic of abstract matrices. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) Calculemus 2009. LNCS (LNAI), vol. 5625, pp. 138–153. Springer, Heidelberg (2009)
12. Syropoulos, A.: Mathematics of multisets. In: Calude, C.S., Pun, G., Rozenberg, G., Salomaa, A. (eds.) Multiset Processing. LNCS, vol. 2235, pp. 347–358. Springer, Heidelberg (2001)
13. Whitney, H.: A logical expansion in mathematics. Bulletin of the American Mathematical Society 34(8), 572–579 (1932)
14. Whitney, H.: Characteristic functions and the algebra of logic. Annals of Mathematics 34(3), 405–414 (1933), http://www.jstor.org/stable/1968168

A Formal Quantifier Elimination for Algebraically Closed Fields

Cyril Cohen and Assia Mahboubi

INRIA Saclay – Île-de-France
LIX École Polytechnique
INRIA Microsoft Research Joint Centre
{Cyril.Cohen,Assia.Mahboubi}@inria.fr

Abstract. We prove formally that the first order theory of algebraically closed fields enjoys quantifier elimination, and hence is decidable. This proof is organized in two modular parts. We first reify the first order theory of rings and prove that quantifier elimination leads to decidability. Then we implement an algorithm which constructs a quantifier free formula from any first order formula in the theory of ring. If the underlying ring is in fact an algebraically closed field, we prove that the two formulas have the same semantic. The algorithm producing the quantifier free formula is programmed in continuation passing style, which leads to both a concise program and an elegant proof of semantic correctness.

1 Introduction

Quantifier elimination is a standard way of proving the decidability of first order theories. In this paper, we investigate the formalization of quantifier elimination, and decidability for the first order theory of *algebraically closed fields*, inside the Coq proof assistant [4]. The work does not address the problem of implementing a fast proof producing a decision procedure. Our motivation is to enrich an existing hierarchy of algebraic structures [8] with a decidability result. Beside automation, decidability validates case analysis on first-order statements, even in the context of a constructive development. In this work, we follow the proof given in standard references [2]. Yet we diverge from the usual expositions of the algorithm using continuation passing style to rephrase and prove quantifier elimination in a more elegant way.

The Coq files for this formalization are available on line at the following URL : `http://perso.crans.org/cohen/closedfields`[1]. Since there are no axioms in the developpment, the fact that this code compiles ensures that the constructive quantifier elimination proof is correct.

The article is composed of three parts. First, we reduce quantifier elimination in discrete structures to the elimination of a single existential quantifier. We also build the boolean decision procedure resulting from quantifier elimination. Then, we establish an algebraic characterization of any existential formula with a single

[1] It can be run using Coq v8.2 and SSReflect v1.2.

quantifier. Finally, we show how to compute a quantifier free formula from this characterization, using a continuation passing style formula transformation.

2 Quantifier Elimination and Decidability

2.1 Preliminaries

In this section we recall some standard material, essentially following [11], and introduce some notations needed in the sequel.

Syntax: Signature, Terms, Formulas. In all what follows, we consider *signatures* of the form: $\Sigma = \mathcal{C} \cup \mathcal{F} \cup \mathcal{R}$, formed of a finite set $\mathcal{C}$ of constant symbols, a finite set $\mathcal{F}$ of function symbols with arity, and a finite set $\mathcal{R}$ of relation symbols with arity. Given such a signature Σ and a countable set of variables $\mathcal{V}$, *terms* are inductively defined as: variables in $\mathcal{V}$ and constants in $\mathcal{C}$ are terms, other terms being of the form $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$ is a function with arity n and $t_1 \ldots t_n$ are terms. A term is *closed* if no free variable occur in it. We write $\mathcal{T}(\Sigma, \mathcal{V})$ for terms, and $\mathcal{T}(\Sigma)$ for closed terms.

The *atomic formulas* of a signature Σ are of the form $t_1 = t_2$ where t_1, t_2 are any terms, and $R(t_1, \ldots, t_n)$ where $R \in \mathcal{R}$ is a relation with arity n. The *first order language* of Σ is the set of all first order formulas with these atoms. First order formulas of Σ are recursively defined by: atomic formulas are first order formulas, other formulas being of the form $\neg f$, $f_1 \wedge f_2$, $f_1 \vee f_2$, $(\exists x, f)$, $(\forall x, f)$, $f_1 \Rightarrow f_2$, where $f, f_1, f_2 \in F$ A formula is *closed* if no variable occurs in it. We write $\mathcal{F}(\Sigma, \mathcal{V})$ for formulas, and $\mathcal{F}(\Sigma)$ for closed formulas. Any subset of $\mathcal{F}(\Sigma)$ is called a *theory* over Σ. We use the $\vdash$ predicate to denote provability: $T \vdash \psi$ means that ψ is a first order consequence from formulas in T. The notation $\boldsymbol{x}$ will denote a finite list of variables $x_1, \ldots, x_k$ for some $k \in \mathbb{N}$.

A theory T *admits quantifier elimination* if, for every $\phi(\boldsymbol{x}) \in \mathcal{F}(\Sigma, \mathcal{V})$, there exists $\psi(\boldsymbol{x}) \in \mathcal{F}(\Sigma, \mathcal{V})$ such that ψ is quantifier free and

$$T \vdash \forall \boldsymbol{x}, ((\phi(\boldsymbol{x}) \Rightarrow \psi(\boldsymbol{x})) \wedge (\psi(\boldsymbol{x}) \Rightarrow \phi(\boldsymbol{x})))$$

Semantics: $\boldsymbol{\Sigma}$-Structures, Models. For any signature $\Sigma = \mathcal{C} \cup \mathcal{F} \cup \mathcal{R}$, a Σ-*structure* is the pair of a set E called the *domain*, and an *interpretation function* I assigning an element of E to each constant symbol in $\mathcal{C}$, a function $E^n \to E$ to each function symbol in $\mathcal{F}$ with arity n, and an n-ary relation on E (i.e. a subset of E^n) to each relation symbol in $\mathcal{R}$ with arity n.

For any Σ-structure A, any term $t(\boldsymbol{x})$, and any list e of values in the domain of A at least as long as the list of variables $\boldsymbol{x}$, we define inductively $[\boldsymbol{t}(\boldsymbol{x})]_{A,e}$ as

– if $\boldsymbol{t}(\boldsymbol{x})$ is x_i, then $[\boldsymbol{t}(\boldsymbol{x})]_{A,e} = e_i$
– if $\boldsymbol{t}(\boldsymbol{x})$ is c for some $c \in \mathcal{C}$, then $[\boldsymbol{t}(\boldsymbol{x})]_{A,e} = I(c)$
– if $\boldsymbol{t}(\boldsymbol{x})$ is $f(\boldsymbol{s}(\boldsymbol{x}))$ where $f \in \mathcal{R}$, and where $\boldsymbol{s}$ are terms with variables $\boldsymbol{x}$, then $[\boldsymbol{t}(\boldsymbol{x})]_{A,e} = I(f)([\boldsymbol{s}(\boldsymbol{x})]_{A,e})$

For any Σ-structure A, any atomic formula $\phi(\boldsymbol{x}) = R(\boldsymbol{t}(\boldsymbol{x}))$ where $R \in \mathcal{R}$, and where $\boldsymbol{t}$ are terms with variables $\boldsymbol{x}$, and any a list e of values in the domain of A at least as long as $\boldsymbol{x}$, if $[\boldsymbol{t}(\boldsymbol{x})]_{(A,e)}$ is in $I(R)$, we say that A is *a model* of ϕ, denoted by $A, e \models \phi$. This definition is extended to any first order formula ϕ by induction on the structure of ϕ. We say that a Σ-structure A is a *model* of a theory T, denoted $A \models T$, if and only if $\forall \phi \in T, A \models \phi$.

We say that two formulas $\phi, \psi \in \mathcal{F}(\Sigma, \mathcal{V})$ are T-*equisatisfiable* if in any model M of T, and for any context e, ($M, e \models \phi$ if and only if $M, e \models \psi$).

We say that a theory T *admits semantic quantifier elimination*, if for every $\phi \in \mathcal{F}(\Sigma)$, there exists $\psi \in \mathcal{F}(\Sigma)$ such that ψ is quantifier free and for any model M of T, and for any list e of values, $M, e \models \phi$ iff $M, e \models \psi$. In this work, we formalize the property of semantic quantifier elimination for the theory of algebraically closed fields.

The Theory of Algebraically Closed Fields. The signature we use in this paper to define the theory of fields (and algebraically closed fields) is $\Sigma_{\text{Fields}} = \{0, 1\} \cup \{-, .^{-1}, +, *\} \cup \emptyset$ (so the only atoms are equalities). We will also use $\Sigma_{\text{Rings}} = \{0, 1\} \cup \{-, +, *\} \cup \emptyset$. The Coq formalization features an extra unary relation symbol for units, because the field theory is built by extending the one of rings with units. We have omitted here this extra predicate for sake of readability. We call *first order theory of algebraically closed field*, the set $T_{\text{ClosedFields}}$ of axioms of fields plus an axiom scheme $(A_n)_{n \in \mathbb{N}}$ where A_n states the existence of a root for any monic polynomial of degree n:

$$A_n := \forall a_0, \dots a_{n-1}, \exists x, x^n + a_{n-1}x^{n-1} \cdots + a_1 x + a_0 = 0$$

Theorem 1. $T_{\text{ClosedFields}}$ *admits quantifier elimination.*

This result is attributed to Tarski [19]. The corresponding geometrical formulation of this result, stating that projections of constructible sets are constructible sets is known as Chevalley's Constructibility theorem [7].

2.2 Formalization Issues

In a type theoretic proof assistant like the Coq system, it is a common practice to define the interface of an algebraic structure using *record types*. Here is an example of a possible definition for commutative groups:

```
Record zmodule := Zmodule{
  M : Type;
  zero : M;
  opp : M -> M;
  add : M -> M -> M;
  _ : associative add;
  _ : commutative add;
  _ : left_id zero add;
  _ : left_inverse zero opp add}.
```

The definition of an algebraic structure interface type like
zmodule can be seen as the definition of a signature Σ_{Zmodules}, together with some axioms T_{Zmodules}, shallow embedded in Coq logic. In the case of zmodule, the signature is $\{0\} \cup \{-, +\} \cup \emptyset$, and the axioms are the expected ones. Populating such a type, i.e. building an element (Z : zmodule) is providing a carrier and an interpretation function, hence a structure Z, together with a proof that it satisfies the set of axioms T_{Zmodules}, i.e. that $Z \models T_{\text{Zmodules}}$. So the inhabitants of the type zmodule are a shallow embedding of the models of T_{zmodules}. One can define similar specifications for more complex algebraic structures like ring, unitRing, integralDomain and field. Packaging such structures can be delicate, and record nesting should be used to achieve sharing and inheritance between structures [9,8]. In our setting [8], the structure closedField of algebraically closed field is formally defined by packing a structure of field with the following extra axiom schema:

```
Definition ClosedField.axiom (R : ring) := forall n P,
  n > 0 -> exists x : R, x ^+ n = \sum_(i < n)(P i) * (x ^+ i).
```

where the notation (x ^+ n) stands for x^n, and the right hand side of the equation is an iterated sum [5] forming the polynomial expression whose coefficients are given by the (P : nat -> R) function.

The syntactic equality predicate on such algebraic structures need not to be defined, as Coq provides such a default equality on any type. In Coq this equality is not effective in general, and one cannot base a case analysis on two terms being equal or not. However in all what follows, we restrict our study to the case of *discrete* structures, in particular discrete fields. This means that we assume that there is a boolean equality test exactly reflecting Coq equality on the terms. For instance a classical formalization of real numbers could fit this framework through the assumption of a boolean equality test, and so could a constructive formalization of algebraic numbers.

To state a quantifier elimination result, we also need to provide an abstract representation of first order formulas. Terms in $\mathcal{T}(\Sigma_{\text{Fields}}, \mathbb{N})$ are formally described as the inhabitants of the following inductive type:

```
Variable T : Type.
Inductive term : Type :=
| Var of nat (* variables *)
| Const of R (* constants *)
| Add of term & term (* addition *)
| Opp of term (* opposite *)
| Mul of term & term (* product *)
| Inv of term (* inverse *)
```

where we reflect division by the product by an inverse. Similarly, first order formulas in $\mathcal{F}(\Sigma_{\text{Fields}}, \mathbb{N})$ are defined by:

```
Inductive formula : Type :=
| Bool of bool
| Equal of term & term
| And of formula & formula
| Or of formula & formula
| Implies of formula & formula
| Not of formula
| Exists of nat & formula
| Forall of nat & formula.
```

where quantifiers explicitly take as argument the natural number representing the name of the variable they bind. We now define a Coq predicate holds: `forall F : field, seq F -> formula F -> Prop`, such that `(holds F e f)` is $F, e \models f$ (see section 2.1). This requires the definition of the eval: `forall F : field, seq F -> term F -> F` function interpreting terms as elements in the model with respect to a context, such that `(eval F e t)` formalizes $[t]_{F,e}$. For instance, the interpretation of the abstract formula:

```
'forall 'X_0, 'forall 'X_1, 'forall 'X_2, 'exists 'X_3,
  'X_0 * 'X_3 * 'X_3 + 'X_1 * 'X_3 + 'X_2 == 0 : formula F
```

where some notations pretty-print the constructors of the `formula` inductive type, is the Coq proposition:

```
forall a b c, exists x, a * x * x + b * x + c = 0 : Prop
```

For any `T : Type`, it is straightforward to test if a formula `(t : formula T)` is quantifier free: we just recursively test that `t` does not feature any `Exists` or `Forall` constructor. This results in a boolean test:

```
Definition qf_form : forall T :Type, formula T -> bool.
```

Now the Coq theorem we prove is that there exists a transformation:

```
Definition q_elim : forall F : closedField, formula F -> formula F
```

such that:

```
Lemma q_elim_wf : forall (F : closedField) (f : formula F),
  qf_form (q_elim f).
Lemma q_elimP : forall (F : closedField) (f : formula F),
  forall e : seq F, holds e f <-> holds e (q_elim f)
```

This latter theorem is a formalization of the *semantic quantifier elimination*, assuming that the shallow formalization of models encompasses all models of a given structure.

2.3 Quantifier Elimination by Projection

For the discrete structures we are interested in, and more generally for first order theories with decidable atoms, the elimination of a single existential quantifier entails full quantifier elimination. We give here a formal account of this reduction, for the special case of the theory of discrete *rings*.

We first show that this problem is general enough by describing a transformation to_rformula of any formula $f \in \mathcal{F}(\Sigma_{\text{Fields}}, \mathbb{N})$ into a formula $f' \in \mathcal{F}(\Sigma_{\text{Rings}}, \mathbb{N})$ such that f and f' are T_{Fields}-equisatisfiable.

For an atom of $\mathcal{F}(\Sigma_{\text{Fields}}, \mathbb{N})$:

– right-hand sides are set at 0 by transforming $(t_1 = t_2)$ into $(t_1 - t_2 = 0)$;
– divisions in the left-hand sides are recursively removed by introducing extra quantifications over fresh variables: $C[t^{-1}] = 0$ is transformed into: $\forall x, (x * t - 1{=}0 \wedge t * x - 1 = 0) \vee (x{=}t \wedge \neg\, (\exists x, (x * t - 1{=}0 \wedge t * x - 1{=}0)) \implies (C[x] = 0)$.

This transformation is trivially recursively lifted to any non atomic formula.

For sake of convenience, we introduce a special data-structure for normalized quantifier-free formulas. They can be represented in disjunctive normal form as:

$$\bigvee_{l \in L} \left(\bigwedge_{i \in I} t_i = 0 \wedge \bigwedge_{j \in J} \neg (t_j = 0) \right)$$

and hence encoded by a list (of sub-formulas in the disjunction), of pairs (one for positive and one for negated atoms) of lists of terms (the left hand sides) : `(seq ((seq term R)*(seq (term R))`. We consider a field `F`, equipped with an operator:

```
Variable proj : nat -> seq (term F) * seq (term F) -> formula F.
```

whose first integer argument represents the name of a variable, second argument is a quantifier free conjunctive formula, and which computes a new abstract formula. This operator is meant to eliminate a quantifier from any formula of the form:

$$\exists x_n, \bigwedge_{i \in I} t_i = 0 \wedge \bigwedge_{j \in J} \neg (t_j = 0)$$

We hence require that on a formula on the ring signature, this operator always computes a quantifier free formula on the ring signature:

```
Definition wf_proj_axiom := forall i bc
    dnf_rterm bc -> qf_form (proj i bc) && rformula (proj i bc).
```

and that it computes a formula equivalent to its input:

```
Definition holds_proj_axiom :=
  forall n bc e,
  let (ex_n_bc := ('exists 'X_n, dnf_to_form [:: bc])%T in
  (holds e ex_n_bc) <-> (holds e (proj n bc)).
```

where `dnf_to_form` converts back the convenient representation we introduced for disjunctive normal form (DNF) quantifier free formulas to an inhabitant of the type `formula F`.

Under the assumptions that the `proj` operator satisfies the properties `wf_proj_axiom` and `holds_proj_axiom`, we can now prove that the field `F` enjoys full quantifier elimination, meaning that we can implement the `q_elim` function of section 2.3. This quantifier elimination procedure proceeds by recursion on the structure of the formula, eliminating the inner-most quantifier:

- if the formula has the form $\exists x_n F$, where F is quantifier-free, it converts F into DNF. Then since $\exists x, \bigvee(\bigwedge p_i \wedge \bigwedge \neg q_j)$ is equivalent to $\bigvee(\exists x, (\bigwedge p_i \wedge \bigwedge \neg q_j))$, it returns $\bigvee(\text{proj } n\ (p_i)\ (q_j))$.
- if the formula has the form $\forall x_n F$, where F is quantifier-free, it is equivalent to $\neg\exists\neg F$ since the decidability of atoms implies that the full theory is classical. The formula $\neg F$ being quantifier free, the first case method applies to eliminate the quantifier from $\exists\neg F$, and hence from $\neg\exists\neg F$.

The sequential representation of quantifier free formulas eases the DNF conversions, and their combination with negations in the case of universal quantifiers.

Finally, we obtain a full formal proof that if a field is equipped with a `proj` operator, with a proof of the two `wf_proj_axiom` and `holds_proj_axiom` properties, then we can derive a correct quantifier elimination procedure `q_elim : formula F -> formula F`, which transforms any first order formula into a quantifier-free one, and such that the input and the output of the quantifier elimination are equisatisfiable in any model of a ring with units.

2.4 Decidability

The first order theory of a field is *decidable* if one can construct a *boolean* operator: `s : seq R -> formula R -> bool`, which reflects the satisfiability of any formula, i.e. satisfies the following property:

```
Variable F : field.
Definition DecidableField.axiom (s : seq F -> formula F -> bool):=
  forall e f, (holds e f) <-> (s e f = true).
```

This provides a computational characterization of decidability since s can be seen as a decision procedure for the first order theory of F.

Of course not all fields have a decidable first order theory: for instance the field theory of rational numbers is undecidable [17]. However quantifier elimination entails decidability for any first order theory with decidable atoms. It is hence straightforward to construct by structural recursion a boolean test qf_eval which correctly reflects the validity of such a quantifier free abstract formula (and remains unspecified on quantified formulas). The correctness of this boolean test is expressed by the lemma:

```
Lemma qf_evalP : forall (e : seq R)(f : formula R),
  qf_form f -> (holds e f) <-> (qf_eval e f = true).
```

where qf_form tests that an abstract formula does not contain any quantifier. The function

```
Definition proj_sat e f := qf_eval e (q_elim f).
```

takes a formula, eliminates its quantifiers, and applies the boolean satisfiability test `qf_eval` on the result. It is a correct decision procedure as shown by the formal proof that it satisfies the `DecidableField.axiom` specification.

3 Polynomial Arithmetic

In section 2.3, we have shown that quantifier elimination on the first order language of fields boils down to the one over the signature of rings. This way, we only handle polynomials in the atoms, instead of arbitrarily nested fractions, which significantly simplifies the proofs. This reduction is often left implicit in standard presentations [2]. The price to pay for this reduction is that we no more have access to the field theory of the algebraically closed fields, but only to the theory which can be stated and proved without using divisions. And this is the theory of the underlying integral domain to the algebraically closed field.

3.1 Representation

We represent univariate polynomials as lists of coefficients with lowest degree coefficients in head position. We require polynomials to be in normal form, in the sense that the last element of the list is never zero. Hence the type `{poly T}` of polynomials with coefficients in the type `T` is a so-called sigma type, which packages a list, and a proof that it last element is non zero. The zero polynomial is therefore represented by the empty list, with a default zero value for the head constant function.

It is convenient and standard to define the degree of a univariate monomial as its exponent, except for the zero constant, whose degree is set at $-\infty$. Then the degree of a polynomial is the maximum of the degree of its monomial. To avoid introducing option types, we simply work here with the size of a polynomial, which is the size of its list. This lifts the usual codomain of degree from $\{-\infty\} \cup \mathbb{N}$ to $\mathbb{N}$ since in our case:

$$\mathrm{size}(p) = \begin{cases} 0 & \text{, if and only if } p = 0 \\ \deg(p) + 1 & \text{, otherwise} \end{cases}$$

3.2 Pseudo-divisions, Pseudo-gcd

When R is an integral domain, it is no more possible in general to program the Euclidean division algorithm on $R[X]$ as it would be if R was a field. The usual polynomial Euclidean division actually involves exact divisions between coefficients of the arguments, which might not be tractable inside R. However it might still remain doable if the dividend is multiplied by a sufficient power of the leading coefficient of the divisor. For instance one cannot perform Euclidean division of $2X^2 + 3$ by $2X + 1$ in $\mathbb{Z}[X]$, but one can divide $4X^2 + 6$ by $2X + 1$ inside $\mathbb{Z}[X]$. In the context of integral domains, Euclidean division should be replaced by *pseudo-division.*

Definition 1 (Pseudo-division). *Let R be an integral domain. Let p and q be elements of $R[X]$. A* pseudo-division *of p by q is Euclidean division of αp by q, where α is an element of R which allows the Euclidean division to be performed inside $R[X]$.*

Note that α always exists and can be chosen to be a sufficient power of the leading coefficient of q. We implement a pseudo-division algorithm, which computes on two given polynomials `p` and `q`, respectively `(scalp p q)` a sufficient α, `(p %/ q)`, the corresponding pseudo-quotient, and `(p %% q)` and the corresponding pseudo-remainder. They satisfy the following specification:

```
Lemma divp_spec: forall p q, (scalp p q) * p = p %/ q * q + p %% q
```

Each possible value of α leads to different values for the pseudo-quotient (resp. pseudo-remainder) of two polynomials, but they are always associated. We say that `p` *pseudo-divides* `q`, denoted `(p %| q)` if pseudo-remainders of `p` by `q` are zero. We recover some standard lemmas about divisibility like:

```
Lemma dvdp_mul : forall d1 d2 m1 m2 : {poly R},
  d1 %| m1 -> d2 %| m2 -> d1 * d2 %| m1 * m2.
```

The pseudo greatest common divisor `gcdp` is obtained by replacing division by pseudo-division in the Euclidean algorithm. This is not the optimal algorithm to compute such a greatest common divisor, which is a non trivial problem. We choose here a naive implementation, since at this point, we are not concerned with efficiency. However we recover standard properties of the greatest common divisor, like:

```
Lemma root_gcd : forall p q x,
 root (gcdp p q) x = root p x && root q x.
```

where `root p x` means that `p` evaluates to `0` at the value `x`. We denote by $\mathrm{gcdp}_{i=1}^{n} P_i$ the iteration of the gcdp function on the list $(P_i)_{i=1\dots n}$ (with at least two elements). The polynomial $\mathrm{gcdp}_{i=1}^{n} P_i$ has a root at x is and only if x is a common root of the $(P_i)_{i=1\dots n}$.

3.3 Common Roots, Exclusive Roots

Recall from section 2.3 that we aim at eliminating the existential quantifier from a formula of the form:

$$(1) \quad \exists x \in R, \bigwedge_{i=1}^{n} P_i(x) = 0 \wedge \bigwedge_{j=1}^{m} Q_j(x) \neq 0$$

Indeed, after the reduction from the first order theory of fields to the first order theory of rings, and the normalization of atoms, atoms are zero conditions on polynomial expressions. In other words, given two finite families of polynomials (P_i) and (Q_j), we need to decide if there exists a point in the underlying field which is a common root of the (P_i)s but root of no Q_j:

$$(1) \Leftrightarrow \exists x \in R, (\mathrm{gcdp}_{i=1}^{n} P_i)(x) = 0 \wedge (\prod_{i=1}^{m} Q_i)(x) \neq 0$$

Given two polynomials P and Q with coefficients in an integral domain R, we introduce the greatest divisor of P coprime to Q, denoted $\mathrm{gdco}_Q(P)$ and recursively defined as:

$$\mathrm{gdco}_Q(P) = \begin{cases} 1 & \text{, if } P = 0 \wedge Q = 0 \\ 0 & \text{, if } P = 0 \wedge Q \neq 0 \\ P & \text{, if gcdp } P\ Q = 1 \\ \mathrm{gdco}_Q(P \ / \ \mathrm{gcdp}\ P\ Q) & \text{, otherwise} \end{cases}$$

where the quotient in the last case is in fact a pseudo-quotient. In particular, $\mathrm{gdco}_Q(P)$ satisfies the following property:

$$\exists x \in R, P(x) = 0 \wedge Q(x) \neq 0 \Leftrightarrow \exists x \in R, \mathrm{gdco}_Q(P)(x) = 0$$

Introducing this $\mathrm{gdco}_Q(P)$ operator provides a new equivalent to (1):

$$(1) \Leftrightarrow \exists x \in R, \mathrm{gdco}_{\left(\prod_{i=1}^m Q_i\right)} \left(\mathrm{gcdp}_{i=1}^n P_i\right)(x) = 0$$

which in particular simplifies the one used in [2]. If the underlying field is algebraically closed, then any non zero polynomial has a root as soon as it is non constant, hence:

$$(1) \Leftrightarrow \mathrm{size}\left(\mathrm{gdco}_{\left(\prod_{i=1}^m Q_i\right)} \left(\mathrm{gcdp}_{i=1}^n P_i\right)\right) \neq 1 \quad (2)$$

This is however not a first order formula as such: size, gcdp and gdco are not directly expressible as functions from $\mathcal{T}(\Sigma_{\mathrm{Ring}}, \mathbb{N})$ to $\mathcal{T}(\Sigma_{\mathrm{Ring}}, \mathbb{N})$. For example, consider the formula $\phi(y) := \exists x, x = 0 \wedge xy \neq 0$. We know from above that:

$$\phi(y) \Leftrightarrow \mathrm{size}\left(\mathrm{gdco}_{xy}(x)\right) \neq 1 \qquad (E)$$

and $\mathrm{gcdo}_{xy}(x) = \begin{cases} 0 & \text{if } y = 0 \\ 1 & \text{if } y \neq 0 \end{cases}$ is not expressible in $\mathcal{T}(\Sigma_{\mathrm{Ring}}, \mathbb{N})$.

Nevertheless, (E) can be decomposed like this : $\begin{cases} \text{if } y = 0 & \phi(y) \Leftrightarrow \mathrm{size}(0) \neq 1 \\ \text{if } y \neq 0 & \phi(y) \Leftrightarrow \mathrm{size}(1) \neq 1 \end{cases}$
and hence $\phi(y)$ be described as formula of $\mathcal{F}(\Sigma_{\mathrm{Ring}}, \mathbb{N})$:

$$\phi(y) \Leftrightarrow ((\mathrm{size}(0) \neq 1) \wedge (y = 0)) \vee ((\mathrm{size}(1) \neq 1) \wedge (y \neq 0))$$

The same kind of transformations applies to size so that in the end:

$$\phi(y) \Leftrightarrow (0 \neq 1) \wedge (y = 0)) \vee ((1 \neq 1) \wedge (y \neq 0))$$

In the general case, the translation of (2) into a first order formula uses case analysis, in fact zero tests, on the coefficients of the polynomials (P_i) and (Q_j). The final first order formula is a disjunction compounding the first order characterizations obtained for each case. In the next part we present the algorithm that systematizes this case analysis and the reconstruction of formulas.

4 Quantifier Elimination for Algebraically Closed Fields

Let $\mathcal{P}, \mathcal{Q} \subset \mathcal{T}(\Sigma_{\text{Rings}}, \{x_1, \dots x_n\})$ two finite sets of terms. In this section, we describe how to *effectively* transform a formula :

$$\phi := \exists x_k, \bigwedge_{p \in \mathcal{P}} p = 0 \wedge \bigwedge_{q \in \mathcal{Q}} \neg(q = 0)$$

over Σ_{Rings} into a quantifier free formula ψ in $\mathcal{F}(\Sigma_{\text{Rings}}, \{x_1, \dots x_n\})$ such that:

$$\forall M \models T_{\text{ClosedField}}, \forall e \in M^n, \quad (M, e) \models \phi \Leftrightarrow (M, e) \models \psi$$

The example given in section 3.3 describes how different values for the context lead to different candidates for the quantifier free formula ψ. It is still possible to construct such a ψ because we can construct an algebraic, quantifier free, model-independent description of a finite partition of the space of parameters into cells. Each cell corresponds to a uniform characterization ψ in the language of rings. The description of this partition is obtained by analyzing the tree of successive zero tests performed when computing the degrees and the pseudo divisions involved in the expression (2) of section 3.3.

A term $t \in \mathcal{P} \cup \mathcal{Q}$ can be seen as polynomials in $R[x_1, \dots, x_n]$. Up to ring theory, we can even reorder the sub-terms of t, to factorize the powers of the quantified variable x_k. At this syntactic level, we introduce the type of a *formal polynomial*, defined as a list of ring terms. The elements of this list represent the successive coefficients of the powers of x_k:

```
Definition polyF (T : Type) := seq (term T).
```

We also define the function:

```
Definition abstrX (R : Ring) (i : nat) (t : term R) : (polyF R)
```

by induction on `(t : term F)`, which computes the formal polynomial in the variable x_i associated to `t`. A formal polynomial t can be interpreted as a univariate polynomial $[p_t]_e$ given a large enough context:

```
Fixpoint eval_poly (R : Ring)(e : seq R) (pf : polyF R) :=
if pf is c :: qf then (eval_poly e qf)*'X + (eval e c) else 0.
```

We need to define again all the operations we have used in the informal presentation of section 3 like the size and greatest common division, to make them operate on formal polynomials. We moreover expect this transformation to be semantically sound. For instance, for a function (f: `poly R -> A`), with an arbitrary return type `A`, we could ask its formal counterpart (F_f: `term F -> A`) to satisfy:

$$\forall t \in \mathcal{T}(\Sigma, \{x_1, \dots x_n\}), \forall M \models T_{\text{ClosedFields}}, \forall e \in M^n, f([p_t]_e) = [F_f(t)]_e$$

This is unfortunately not possible: consider the size function on polynomials, applied to the polynomial x. According to the value assigned by a given context to x, the size of x will be either 0 or 1. But there is no way to encode a case analysis at the syntactic level of terms handled by a formal counterpart F_{size}. In fact, these formal functions on terms should return *lists of values*, reflecting all the possible values for all the tests performed by the body of the polynomial function. Going back to section 3.3, the output quantifier free formula is a disjunction over all the different values of the degree, specified by the conditions leading to these respective values. This construction requires inspecting the invariants of the code of the functions, to prove the correctness and soundness of the generated conditions. This approach is the one usually described in the literature (see e.g. [2]). To avoid this painful formula reconstruction, we diverge from this standard presentation. First, we transform the polynomial operations described in section 3.3 to return quantifier free *formulas* instead of terms. This allows to encode case analysis, using the simple construction:

```
Definition ifF (then else cond: formula F) : formula F :=
   ((cond /\ then) \/ ((~ cond) /\ else)).
```

Simultaneously, we concisely internalize the administration of conditions in the body of the formal counterpart by programming them in *continuation passing style* (CPS). The CPS version of a function `f : A`$_1$`->...-> A`$_n$`-> B` has the form `f_cps (k : B -> T) (a`$_1$`: A`$_1$`)...(a`$_n$`: A`$_n$`) : T`, where `k` is called the *continuation*. For example, the rewritten `size` function is

```
Fixpoint sizeT (k : nat -> formula F) (p : polyF) :=
  if p is c::q then
    sizeT (fun n => if n is m.+1 then (k m.+2)
                    else ifF (k 0) (k 1) (c == 0)) q
  else k 0.
```

which is a translation of a CPS version of the size function over the polynomials of section 3.1. Note that working with continuations means that the code handles the formulas to be output, instead of the arguments, and can hence feature the `ifF` construction to directly build the case disjunction in the language of formulas. The zero test on formal polynomial is then:

```
Definition isnull (k : bool -> formula F) (p: polyF) :=
  sizeT (fun n => k (n == 0)) p.
```

The semantic specification that a CPS function
`fT : (B -> formula F)-> (A`$_1$`->...-> A`$_n$`-> formula F)` which builds the formal counterpart of a function `f : A`$_1$`->...-> A`$_n$`-> B` is formally expressed by a lemma of the form:

```
Lemma fTP : forall (k : B -> formula F) (a₁ : A₁) ... (aₙ : Aₙ),
forall (e : seq (term F)),
 qf_eval e (fT k a₁ ... aₙ) = qf_eval e (k (f a₁ ... aₙ)).
```

We use here the `qf_eval` function since the output formulas should always be quantifier free. For example, `sizeT` is correct w.r.t. `size` since we prove that:

```
Lemma sizeTP : forall k p e,
  qf_eval e (sizeT k p) = qf_eval e (k (size (eval_poly e p))).
```

We transform and specify the gdcp operation of 3.3 in the same CPS way to obtain a gcdpT function, naturally extended to lists of formal polynomials by gdcpTs. And we can express a first quantifier elimination lemma, that takes a list of formal polynomials ps and a formal polynomial q:

```
Definition ex_elim_seq (ps : seq polyF) (q : polyF) : formula F :=
  gcdpTs (gdcopT q (sizeT (fun n => Bool (n != 1)))) ps.
```

where Bool is a constructor of formula F, see section 2.2.

The projection function required at section 2.3 is finally implemented as:

```
Definition proj (x : nat) (pqs : seq (term F) * seq (term F)) :=
  ex_elim_seq (map (abstrX x) (fst pqs))
    (abstrX x (\big[Mul/1%T]_(q <- snd pqs) q)).
```

where \big is a notation for iterated operators [5], which constructs the formal term representing the product of the polynomials coming from the list (snd pqs). Proving that the proj operator outputs quantifier free formulas is straightforward: we have to check that each continuation can only output quantifier free formula. This is done by a trivial case analysis on each of the continuations. Proving its semantic correctness, i.e. the holds_proj_axiom axiom of section 2.3, is a combination of the semantic correctness of the CPS functions with the results formally proved in section 3.3.

5 Conclusion

This paper describes the implementation in Coq of a quantifier elimination algorithm for algebraically closed fields, together with a complete proof of correctness. To the best of our knowledge, the present work is the first to address formally quantifier elimination for a *generic* structure of algebraically closed fields. The algorithm operates on an abstract representation of first order formulas, equipped with an evaluation operator interpreting them as Coq statements. It transforms a formula in the language of fields into a new formula in the language of rings, and finally obtains a quantifier free formula in the language of fields. Each step is semantically correct: it preserves the provability of the Coq interpretations in any algebraically closed field structure. This semantic approach hence results in a Coq reflexive decision procedure for algebraically closed fields, except that we do not provide the reification mechanism.

The present procedure is not *optimized*. Yet this should come as a natural follow-up to this work. Efficient decision procedures for the theory of real closed fields for instance often deal with the universal fragment of the theory using Gröbner bases computations. Harrison [10] has formalized both quantifier elimination for the theory of complex numbers, and a proof producing a version of Buchberger algorithm [6]. An efficient Gröbner based tactic is also available in Coq [15] for complex numbers. It would be interesting to investigate how to merge these Coq developments.

The reduction of quantifier elimination to the elimination of a single existential is a standard result. Nipkow proposes in [13] a modular framework to build decision procedures along this motive. However, the continuation passing style we use to feed the prerequisite existential elimination seems an original idea. This approach makes the programing and the specification elegant and concise. Moreover, the produced formulas are in general more compact than the one produced by an invariant-based approach. We plan to investigate how the method we have presented scales to the theory of real closed fields [2].

From a model-theoretic point of view, our approach deserves further discussion in two ways. First, we weaken the characterization of quantifier elimination given in section 2.1. Indeed, we use Coq native records to shallow embed models, which may not be equivalent to a deep formalization of the $\models$ relation. However, this is enough to establish decidability results. A generalization would be to consider all the models definable in set theory, deeply embedding the formalization of structure, satisfiability and models. Such a work could rely on previous works about the formalization of set theory in Coq [1,18]. A second axis would be to investigate a constructive link between provability and satisfiability, i.e. proving Gödel's completeness theorem in Coq [12]. This theorem has already been formalized within the Isabelle system [16], and proving it in Coq could for instance rely on the structures already introduced for a Coq formal proof of Gödel's incompleteness theorem [14], and the constructive approaches to the Gödel's completeness result [3]. Proving this theorem would result in the equivalence between syntactic quantifier elimination and semantic (deep version of) quantifier elimination.

Acknowledgments. The authors wish to thank Georges Gonthier for numerous comments and improvements. The proofs relating quantifier elimination and decidability were done in close collaboration with him, and he suggested the successful idea of using CPS-style to transform algorithms on polynomials into formula producing ones. The authors are also grateful to Thierry Coquand for his comments and suggestions on preliminary versions of this work.

References

1. Barras, B.: Sets in coq, coq in sets. In: Proceedings of the 1st Coq Workshop. Technical University of Munich Research Report (2009)
2. Basu, S., Pollack, R., Roy, M.-F.: Algorithms in Real Algebraic Geometry, Algorithms and Computation in Mathematics. Springer, New York (2006)
3. Berardi, S., Valentini, S.: Krivine's intuitionistic proof of classical completeness for countable languages. Ann. Pure Appl. Logic 129(1-3), 93–106 (2004)
4. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development, Coq'Art: the Calculus of Inductive Constructions. Springer, Heidelberg (2004)
5. Bertot, Y., Gonthier, G., Biha, S.O., Pasca, I.: Canonical big operators. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2008. LNCS, vol. 5170, pp. 86–101. Springer, Heidelberg (2008)

6. Buchberger, B.: Groebner Bases: Applications. In: Mikhalev, A.V., Pilz, G.F. (eds.) The Concise Handbook of Algebra, pp. 265–268. Kluwer Academic Publishers, Dordrecht (2002)
7. Chevalley, C., Cartan, H.: Schémas normaux; morphismes; ensembles constructibles. In: Séminaire Henri Cartan, Numdam, vol. 8, pp. 1–10 (1955-1956), http://www.numdam.org/item?id=SHC_1955-1956__8__A7_0
8. Garillot, F., Gonthier, G., Mahboubi, A., Rideau, L.: Packaging mathematical structures. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 327–342. Springer, Heidelberg (2009)
9. Geuvers, H., Pollack, R., Wiedijk, F., Zwanenburg, J.: A constructive algebraic hierarchy in Coq. Journal of Symbolic Computation 34(4), 271–286 (2002)
10. Harrison, J.: Complex quantifier elimination in HOL. In: Boulton, R.J., Jackson, P.B. (eds.) TPHOLs 2001: Supplemental Proceedings, Division of Informatics, University of Edinburgh, pp. 159–174 (2001); Published as Informatics Report Series EDI-INF-RR-0046, http://www.informatics.ed.ac.uk/publications/report/0046.html
11. Hodges, W.: A shorter model theory. Cambridge University Press, Cambridge (1997)
12. Gödel, K.: Über die Vollständigkeit des Logikkalküls. PhD thesis, University of Vienna, Austria (1929)
13. Nipkow, T.: Linear quantifier elimination. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 18–33. Springer, Heidelberg (2008)
14. O'Connor, R.: Incompleteness & Completeness, Formalizing Logic and Analysis in Type Theory. PhD thesis, Radboud University Nijmegen, Netherlands (2009)
15. Pottier, L.: Connecting gröbner bases programs with coq to do proofs in algebra, geometry and arithmetics. In: LPAR Workshops. CEUR Workshop Proceedings, vol. 418. CEUR-WS.org (2008)
16. Ridge, T., Margetson, J.: A mechanically verified, sound and complete theorem prover for first order logic. In: Hurd, J., Melham, T. (eds.) TPHOLs 2005. LNCS, vol. 3603, pp. 294–309. Springer, Heidelberg (2005)
17. Robinson, J.: Definability and decision problems in arithmetic. Journal of Symbolic Logic 14, 98–114 (1949)
18. Simpson, C.T.: Formalized proof, computation, and the construction problem in algebraic geometry (2004)
19. Tarski, A.: A decision method for elementary algebra and geometry. In: RAND Corp., Santa Monica, CA (1948) (manuscript); Republished as A Decision Method for Elementary Algebra and Geometry, 2nd edn. University of California Press, Berkeley (1951)

Computing in Coq with Infinite Algebraic Data Structures[⋆]

César Domínguez and Julio Rubio

Departamento de Matemáticas y Computación, Universidad de La Rioja,
Edificio Vives, Luis de Ulloa s/n, E-26004 Logroño (La Rioja, Spain)
cesar.dominguez, julio.rubio@unirioja.es

Abstract. Computational content encoded into constructive type theory proofs can be used to make computing experiments over concrete data structures. In this paper, we explore this possibility when working in Coq with chain complexes of infinite type (that is to say, generated by infinite sets) as a part of the formalization of a hierarchy of homological algebra structures.

Keywords: Theorem proving, formal methods, computer algebra, program verification.

1 Introduction

One main feature of constructive type theory, through the well-known Curry-Howard isomorphism, is the equivalence between proving and programming. This is clearly one of the advantages of Coq [5] with respect to other proof assistants, like Isabelle/HOL [22]. This characteristic is the base of reflective tactics, pioneered by S. Boutin [6], and successfully used, for instance, in [14,20].

Computing can play another role when formalizing a proof. It can be useful, for example, to check some conjecture over concrete cases. When dealing with standard data structures (as lists, trees, and the like), these experiments can be done in a parallel line by programming the tests in Java, C, or any other programming language. If infinite data structures occur, programming them is a more delicate task, and it can be rewarding to keep a tighter link among programs and specifications.

Infinite data structures, presented as coinductive objets as streams, have been dealt with in the theorem proving literature (see [5] for instance). In this work, we undertake another via to manage the infinity, working with algebraic structures of infinite type (that is to say, generated by infinite sets) [24]. We report in this paper on an experiment of this nature, in the area of homological algebra. It is well-known that homological information is not computable over general (infinite type) chain complexes (see [23]). For instance, if (C, d) is an *acyclic* chain complex, and $x \in C_n$ is a *cycle* (this means $d_n(x) = 0$), then *there exists*

⋆ Partially supported by Ministerio de Ciencia e Innovación, project MTM2009-13842-C02-01, and by European Commission FP7, STREP project ForMath.

$z \in C_{n+1}$ such that $d_{n+1}(z) = x$ (that is, x is a *boundary*). But if C_{n+1} is a free module of infinite type, and no other information is available, there is no general algorithm computing a pre-image z of x.

Sergeraert's effective homology [25] is a theory allowing solving large classes of problems of this sort, even in the infinite dimensional case. This paper continues our previous work in translating Sergeraert's ideas to theorem provers [2,3,4,1], with the aim of formalizing this part of algorithmic mathematics and, more importantly, of applying formal methods to the study of the Kenzo system [12] (a Common Lisp program developed by Sergeraert to implement effective homology algorithms). The first important milestone in this area was the mechanized proof in the Isabelle/HOL proof assistant of the Basic Perturbation Lemma (BPL), published in [2]. This formal proof was carried out in the Higher Order Logic (HOL) built on top of Isabelle, and therefore extracting programs from it was not a simple task. The findings on this topic were reported in [3]. A different approach is being carried out by T. Coquand and A. Spiwack [9] who are using Coq to model a part of Category Theory, and then trying to obtain a BPL proof in this larger context.

The data structures of effective homology are organized in two layers (as algebraically modeled in [17,10]): the first layer is composed of algebraic data structures (chain complexes, simplicial sets, . . .) and the second one of standard data structures (lists, trees, . . .) which are representing *elements* of data from the first layer. Infinite type data structures appear only in the first layer. Computing in this first layer can be done in an abstract way, and it is equivalent in Coq to proving theorems. For example, a theorem stating "the direct sum of two chain complexes is a chain complex" contains an algorithm constructing the mentioned direct sum. Coq can deal with this structure, no matter whether it is of finite or infinite type. But actual computations really take place *within* algebraic structures of the first layer. To compute with Coq in this sense has no advantage of being any more direct. It is needed to construct concrete instances of chain complexes and other possibly infinite algebraic data structures. Then we must build concrete elements (second layer) of these particular structures, and finally put to work the algorithms abstractly described in the first layer.

In this paper we discuss this procedure in a case related to the effective homology of the cone of a chain complex morphism. This formalization was part of the implementation in Coq of the algorithm computing the effective homology of a bicomplex (see [11]). Now, we use the computing capabilities of Coq to explore whether some concrete cones are acyclic or not, as a previous step to proving a general property.

The paper is organized as follows. Section 2 contains some preliminaries on algebraic structures, both in Mathematics and in Coq. Section 3 describes the formalization in Coq of the algorithm computing the effective homology of a cone, in a way that slightly generalizes our previous work in [11]. Concrete Coq instances of chain complexes of infinite type are introduced in Section 4. Then explicit calculations with elements are presented in Section 5, using Coq as a computing tool to check some conjectures. The paper ends with conclusions, future work, and the bibliography. The Coq source files are available at `https://esus.unirioja.es/psycotrip/archivos_documentos/CCIADS.zip`

2 Algebraic Data Structures in Coq

In this section we introduce the algebraic structures which support our constructions. They include chain complexes, chain complex morphisms, and reductions and effective homologies of chain complexes. The formalization in Coq of these structures are also described.

We assume as known the notions of *ring*, *module* over a ring and *module morphism* (see [16] for instance). A ring R commutative and with unity is fixed all through the paper, and modules are supposed to be *left* R-modules.

We have built these basic structures in Coq using records called `Ring`, `Module` and `ModHom`, respectively. They are based on the ones included in CoRN [13] (but simplifying them: basically eliminating the apartness relation included in setoids which is not used by us, since we are working in a *discrete* mathematics setting). Besides, further constructions as for instance the addition or the composition of module morphisms are defined, and are represented using the infix notation `[+h]` or `[oh]`, respectively.

A *free R-module generated over a set B* is the module $R[B]$ whose elements are linear combinations with elements of B as generators. The addition and the external product by elements of R are defined in the natural way. Since we are planning to work in a constructive logic setting, it is convenient to define a *free* module as one module M where an explicit isomorphism is known between M and $R[B]$ (the set of generators B must also be explicitly given). If B is finite, the free module is said *of finite type.*

The formalization of free modules in Coq follows the ideas given by L. Pottier in the Coq contributions web page [19]. There, a definition can be found of a module built by freely generation from a basis, which is given by a *setoid* (*i.e.* a set with an equality, usually denoted by `[=]`), using the module operations. If we call B the basis setoid, this is representing the mathematical structure $R[B]$ introduced above. Then, our formalization of free modules consists of a record with a module and an explicit isomorphism to such a freely generated module. In order to deal with finite sets in a constructive type theory, more care is needed. For instance, several alternatives for defining finite sets in a constructive logic are included in [8]. Finite algebraic structures have also been implemented in Coq in [15] as a first milestone of a long-term effort to formalize the Feit-Thompson theorem. Our formalization is the following. Given a natural number $k \in \mathbb{N}$, let us denote $FS(k)$ the (finite) setoid $\{0, 1, \ldots, k-1\}$ (with the Leibniz equality). We consider a setoid B as *finite* if it is endowed with a natural number $k \in \mathbb{N}$ and an explicit bijection to $FS(k)$. Then, a free module of finite type is a free module, but we impose that the generator set is *equal* (in the Coq internal sense) to $FS(k)$.

We concentrate ourselves in the sequel on *free* modules, since it is the unique kind of modules dealt with in the Kenzo system [12].

We are ready to introduce the first *graded* concept, needed in Homological Algebra and Algebraic Topology.

Definition 1. *A* graded module *M is a family of R-modules indexed by the integer numbers $(M_i)_{i\in\mathbb{Z}}$. A graded module is* free *(or* free of finite type*) if M_i is free (free of finite type, respectively) for all $i \in \mathbb{Z}$. If $x \in M_i$, the index i is called* degree *of the element x.*

Definition 2. *Given a graded module M a* differential operator *d on M is a family of module morphisms $(d_i\colon M_{i+1} \to M_i)_{i\in\mathbb{Z}}$ such that $d_i \circ d_{i+1} = 0$ for all $i \in \mathbb{Z}$.*

Definition 3. *A* chain complex *is a pair $CC = (M, d)$ where M is a graded module and d a differential operator on M. A chain complex is called* free *(or* free of finite type*) when its underlying graded module is free (free of finite type, respectively).*

Chain complexes have a corresponding notion of morphism.

Definition 4. *A* chain complex morphism *(or, simply, a* chain morphism*) $f\colon CC \to CC'$ between two chain complexes $CC = (M, d)$ and $CC' = (M', d')$ is a family of module morphisms $(f_i\colon M_i \to M'_i)_{i\in\mathbb{Z}}$ such that $f_i \circ d_i = d'_i \circ f_{i+1}$ for all $i \in \mathbb{Z}$.*

Given a ring `R: Ring`, a graded module can be formalized in Coq with the following *dependent* type: `Z -> Module R`, which accurately represents a family of modules indexed by the integer numbers. Then, a (free) chain complex can be formalized in Coq using the following record structure:

```
Record ChainComplex: Type:=
 {GrdMod:> Z -> FreeModule R;
  Diff: forall i:Z, ModHom (R:=R) (GrdMod (i + 1)) (GrdMod i);
  NilpotencyDiff: forall i:Z, (Nilpotency (Diff i)(Diff (i + 1))}.
```

where the nilpotency property is defined by `Nilpotency(g:ModHom B C) (f:ModHom A B):= forall a: A, ((g[oh]f)a)[=]Zero`.

In a similar way, given two chain complexes `CC1 CC2: ChainComplex R`, a chain complex morphism `ChainComplexHom` is represented as a record with a family of module morphisms `GrdModHom:>forall i:Z,ModHom(CC1 i)(CC2 i)` which commutes with the chain complex differentials.

Now, the central definition in effective homology theory: *reduction*. A reduction establishes a link between a "big" chain complex, called *top complex*, and a smaller one, called *bottom complex*, in such a way that if all the homological problems are solved in the bottom complex, then it is the same in the top one.

Definition 5. *A* reduction *is a 5-tuple (TCC, BCC, f, g, h)*

$$\begin{tikzcd} TCC \arrow[loop left, "h"] \arrow[r, shift left, "f"] & BCC \arrow[l, shift left, "g"] \end{tikzcd}$$

where $TCC = (M, d)$ and $BCC = (M', d')$ are chain complexes (named top *and* bottom *chain complex), $f\colon TCC \to BCC$ and $g\colon BCC \to TCC$ are chain morphisms, $h = (h_i\colon M_i \to M_{i+1})_{i\in\mathbb{Z}}$ is a family of module morphisms (called* homotopy operator*), which satisfy the following properties for all $i \in \mathbb{Z}$:*

1. $f_i \circ g_i = id_{M'_i}$
2. $d_{i+1} \circ h_{i+1} + h_i \circ d_i + g_{i+1} \circ f_{i+1} = id_{M_{i+1}}$
3. $f_{i+1} \circ h_i = 0$
4. $h_i \circ g_i = 0$
5. $h_{i+1} \circ h_i = 0$

And now, the relevant case. In a free chain complex *of finite type* the homological problems can be solved algorithmically in a simple way (at least in cases where the ring R allows one to diagonalize matrices over R; this includes the case $R = \mathbb{Z}$, the most important one in Algebraic Topology; see [24]). Thus, if from a chain complex (possibly of infinite type) we can get a reduction to a chain complex of finite type, the homological problem is solved for the initial complex. This is the strategy followed in the Kenzo system. And it is the very notion of chain complex with *effective homology*.

Definition 6. *A chain complex CC is with* effective homology *if it is free and it is endowed with a reduction where CC itself is the top chain complex and the bottom chain complex is free of finite type.*

Given a chain complex `CC1: ChainComplex R`, a homotopy operator is represented in Coq as a family of module morphisms `HomotopyOperator:= forall i: Z, ModHom(C1 i)(C1(i + 1))`. The reduction notion is then formalized as a record `Reduction` with two chain complexes `topCC:ChainComplex R`, `bottomCC:ChainComplex R` and three morphisms `f_t_b:ChainComplexHom topCC bottomCC`, `g_b_t:ChainComplexHom bottomCC topCC`, `h_t_t:HomotopyOperator topCC`. Besides, five fields representing the five reduction properties are included. For instance, the field which corresponds to the second property is: `rp2: homotopy_operator_property f_t_b g_b_t h_t_t` with:

```
Definition homotopy_operator_property:= forall(i: Z)(a: C1(i+1)),
  (((Diff C1(i+1))[oh]h(i+1))[+h](h i[oh](Diff C1 i))[+h]
       (g(i+1)[oh]f(i+1))) a [=] a.
```

Some comments on these Coq definitions are needed. Why are the elements in this definition considered to be on the `i+1`-th degree and not on the `i`-th degree, as it is the usual definition of reduction? The same decision was previously taken when the definition of differential was introduced. It is clear that as we are considering the definition for all the integers, both definitions are equivalent. But, a Coq technical problem is easily avoided thanks to our definition. We are going to focus our attention on the `(h i[oh](Diff C1 i))` component of the definition. The differential takes an element in degree `i+1` and obtains an element in degree `i` which is translated to a component in degree `i+1` by the homotopy operator. If we consider the *mathematically equivalent* definition, considering the differential defined from degree `i` to `i-1`, then the corresponding component would be `(h(i-1)[oh](Diff C1 i))`. In this composition, the differential takes an element in degree `i` and returns an element in degree `i-1`, which is now translated to a component in degree `i-1+1`. In Coq this element is *equal* but is not *convertible* to `i`. So, we will obtain a Coq type error from this sum of morphisms. A *transition* function between equal but not directly convertible types (which it is essentially an identity between types) can be introduced allowing us to overcome this drawback[1].

The concept of *free of finite type chain complex* is then obtained in Coq as a specialization of the *chain complex* structure: simply adding that the family of modules are free modules of finite type. In a similar way it is formalized the concept of *effective homology* as a specialization of the *reduction* structure by declaring the *bottomCC* is of finite type.

3 Effective Homology of the Cone in Coq

In this section we first define the notion of the cone of a chain complex morphism. Then, the main result that we are going to deal with is stated: the effective homology of a cone. We also show how this theorem can be proved in Coq.

Definition 7. *Given a pair of chain complexes* $CC = ((M_i)_{i\in\mathbb{Z}}, (d_i)_{i\in\mathbb{Z}})$ *and* $CC' = ((M'_i)_{i\in\mathbb{Z}}, (d'_i)_{i\in\mathbb{Z}})$ *and a chain complex morphism* $\alpha\colon CC \to CC'$*, the* cone *of* α*, denoted by* $Cone(\alpha)$*, is a chain complex* $((M''_i)_{i\in\mathbb{Z}}, (d''_i)_{i\in\mathbb{Z}})$ *such that, for each* $i \in \mathbb{Z}$*,* $M''_i = M_i \oplus M'_{i+1}$ *and* $d''_i(x, x') = (-d_i(x), d'_{i+1}(x') + \alpha_{i+1}(x))$ *for any* $x \in M_{i+1}$ *and* $x' \in M'_{i+2}$.

Now, the theorem which determines the effective homology of a cone can be stated.

Theorem 1. *Given two reductions* $r = (TCC, BCC, f, g, h)$ *and* $r' = (TCC', BCC', f', g', h')$ *and a chain morphism* $\alpha\colon TCC \to TCC'$ *between their top chain complexes, it is possible to define a reduction* $r'' = (Cone(\alpha), BCC'', f'', g'', h'')$ *with* $Cone(\alpha)$ *as top chain complex and:*

- $BCC'' = Cone(\alpha')$ *with* $\alpha'\colon BCC \to BCC'$ *defined by* $\alpha' = f' \circ \alpha \circ g$
- $f'' = (f, f' \circ \alpha \circ h + f')$, $g'' = (g, -h' \circ \alpha \circ g + g')$, $h'' = (-h, h' \circ \alpha \circ h + h')$

$$\begin{tikzcd}
TCC \arrow[loop above, "h"] \arrow[r, shift left, "f"] \arrow[d, "\alpha"'] & BCC \arrow[l, shift left, "g"] \arrow[d, dashed, "\alpha'"'] \\
TCC' \arrow[loop below, "h'"] \arrow[r, shift left, "f'"] & BCC' \arrow[l, shift left, "g'"]
\end{tikzcd}$$

Besides, if TCC *and* TCC' *are objects with effective homology through the reductions* r *and* r'*, then* $Cone(\alpha)$ *is an object with effective homology through* r''.

In [11] we formalized in Coq the effective homology of a bicomplex. That result can be considered as a generalization of the previous theorem to an infinite

[1] We acknowledge T. Coquand for the suggestion of this idea.

(indexed by the natural numbers) family of reductions. Nevertheless, in order to obtain it, the chain complexes must be positive, *i.e.*, with null components in the negative indexes (or, in other equivalent presentation, indexed by the natural numbers). In this paper, we have not this constraint since we work with a general definition of chain complex, with modules indexed by integer numbers.

The formalization of Theorem 1 in Coq is obtained as follows. Given two chain complexes `CC0 CC1: ChainComplex R` and a chain complex morphism `f: ChainComplexHom CC1 CC0`, the cone of this morphism is a chain complex with family of modules `ConeGrdMod:= fun i: Z => Sum_FreeModule (CC1 i) (CC0(i+1))` (with the direct sum of free modules `Sum_FreeModule` defined in a natural way) and with differential operator defined as follows:

```
Definition ConeDiffGround:= fun (i: Z)(ab:(ConeGround (i+1))) =>
 ([--](Diff CC1 i(fst ab)), ((Diff CC0(i+1))(snd ab)[+]f(i+1)(fst ab))).
```

It is not difficult to prove that these functions define a module morphism which satisfies the differential condition. This last property allows one to build the cone chain complex associated to a chain complex morphism: `Cone(f)`.

Given now two reductions `r1 r2: Reduction R` and a chain complex morphism between their top chain complexes `alpha: ChainComplexHom(topCC r1) (topCC r2)`, it is possible to define a chain complex morphism `alpha'` between the bottom chain complexes through the function `alpha'':= fun n: Z => (f_t_b r2 i)[oh](alpha i)[oh](g_b_t r1 i)`.

The first part of Theorem 1 is proved if we build a reduction between `Cone(alpha)` and `Cone(alpha')`. The first chain complex morphism of the reduction is defined in the following way:

```
Definition f_cone_reductionGround:
 forall i: Z, (Cone alpha) i -> (Cone alpha') i:=
  fun (i: Z)(ab: (Cone alpha) i) => ((f_t_b r1 i) (fst ab),
   (((f_t_b r2 (i+1)) [oh] (alpha (i+1)) [oh] (h_t_t r1 i)) (fst ab)) [+]
     (f_t_b r2 (i+1)) (snd ab)).
```

Analogous definitions are provided for the two other morphisms of the reduction. Then we state Coq lemmas for the reduction properties on these morphisms. The proof of these lemmas consists in applying mainly equational reasoning over setoid equalities, following closely the *paper and pencil* proof. It allows building the reduction of a cone: `ConeReduction(alpha)`.

Finally, given two effective homologies `r1 r2: EffectiveHomology R` and a chain complex morphism `alpha` between their top chain complexes, `ConeReduction(alpha)` is directly a reduction of the cone. Then, in order to define an effective homology for the cone it remains to prove that the bottom free chain complex of this reduction is free of finite type. It is easily obtained in Coq since the direct sum of free chain complex of finite type is free of finite type.

4 Instances of Chain Complexes of Infinite Type

A working representation in a proof assistant of the concepts included in previous sections has to be sound, but also needs to be useful. The second feature can be shown by formally proving some results. This was the purpose of the previous section. The first feature can be illustrated by providing instances of the representations, that accurately reflect usual mathematical entities. This is the aim of this section which includes different instances of all the previous structures.

First, we define some elementary instances which will act as building blocks for more elaborated constructions. The first example is the null free module $M^{(0)}$ (*i.e.*, a module with the unit as unique element). This is indeed a free module of finite type, generated by the setoid with zero elements. Then, a null free chain complex can be defined $CC^{(0)} = ((M^{(0)})_{i\in\mathbb{Z}}, (d^{(0)})_{i\in\mathbb{Z}})$ (*i.e.*, with the previous module in each degree and the null differential). This chain complex can be also built as a free chain complex of finite type $FCC^{(0)}$, defined from the corresponding free module of finite type. Obviously, a trivial effective homology for this chain complex can be defined.

Another basic example is the free module of the integers $\mathbb{Z}$ (over the ring of integers) which we denote in Coq by `ZFreeModule`. This module can be also implemented as a module of finite type, `ZFinFreeModule`, generated by the setoid with only one element. Then, an example of free chain complex is $CC^{(1)} = ((M^{(1)})_{i\in\mathbb{Z}}, (d^{(1)})_{i\in\mathbb{Z}})$ with $(M^{(1)})_i = \mathbb{Z}$, $\forall i \in \mathbb{Z}$, and $(d^{(1)})_i \colon \mathbb{Z} \to \mathbb{Z}$ such as $(d^{(1)})_i(x) = 2 * x$ if i is even and $(d^{(1)})_i(x) = 0$ otherwise:

$$\begin{tikzcd}
\cdots & \mathbb{Z} \arrow[l, "0"'] & \mathbb{Z} \arrow[l, "\times 2"'] & \mathbb{Z} \arrow[l, "0"'] & \mathbb{Z} \arrow[l, "\times 2"'] & \mathbb{Z} \arrow[l, "0"'] & \cdots \arrow[l, "\times 2"'] \\
\text{degree} & \text{-2} & \text{-1} & 0 & 1 & 2 &
\end{tikzcd}$$

The Coq formalization of the required differential is obtained through the functional type `fun i: Z => if (Zeven_bool i) then x2_ModHom else (ModHom_zero ZFreeModule ZFreeModule)`. It is easy to prove that this morphism satisfies the nilpotency condition. A similar free chain complex of finite type $FCC^{(1)}$ can be defined using the corresponding family of free modules of finite type. Besides, we can define a trivial effective homology between both complexes that we name `Id_Z_2x_0_EffectiveHomology`:

$$\begin{tikzcd}
CC^{(1)} \arrow[loop above, "0"] \arrow[r, shift left, "id"] & FCC^{(1)} \arrow[l, shift left, "id"]
\end{tikzcd}$$

The previous examples are chain complexes of *finite type*, since the modules are free of finite type (in that case with zero or one generator). An example of a free module of *infinite type* is $\mathbb{Z}[\mathbb{N}]$, the free module generated by the natural numbers (over the ring of integer numbers) which we denote in Coq by `Z_nat_FreeModule`. It is defined by taking as free module the one freely generated from the setoid denoted in Coq by `nat_as_Setoid` (that is to say, the setoid of natural numbers with the Leibniz equality). The definition is then completed with the same module as module representation and the identity as isomorphism between them. To keep notations clear, the generator i of $\mathbb{Z}[\mathbb{N}]$ will be denoted by x_i, $\forall i \in \mathbb{N}$.

Now, a chain complex of infinite type $CC^{(2)} = ((M^{(2)})_{i\in\mathbb{Z}}, (d^{(2)})_{i\in\mathbb{Z}})$ is built where $(M^{(2)})_i = \mathbb{Z}[\mathbb{N}]$, $\forall i \in \mathbb{Z}$, and $(d^{(2)})_i \colon \mathbb{Z}[\mathbb{N}] \to \mathbb{Z}[\mathbb{N}]$ defined on generators (and then extended to all elements by freely generation) in the following way: if i is even, $(d^{(2)})_i(x_j) = x_j$ if j is even and $(d^{(2)})_i(x_j) = 0$ otherwise; and if i is odd, $(d^{(2)})_i(x_j) = 0$ if j is even and $(d^{(2)})_i(x_j) = x_j$ otherwise. This differential on generators can be illustrated with the following diagram:

$$\begin{tikzcd}
\mathbb{Z}[\mathbb{N}] & \mathbb{Z}[\mathbb{N}] \arrow[l, "\text{i even}"', "(d^{(2)})_i"] & \mathbb{Z}[\mathbb{N}] & \mathbb{Z}[\mathbb{N}] \arrow[l, "\text{i odd}"', "(d^{(2)})_i"] \\
x_0 & x_0 \arrow[l, mapsto] & 0 & x_0 \arrow[l, mapsto] \\
0 & x_1 \arrow[l, mapsto] & x_1 & x_1 \arrow[l, mapsto] \\
x_2 & x_2 \arrow[l, mapsto] & 0 & x_2 \arrow[l, mapsto] \\
0 & x_3 \arrow[l, mapsto] & x_3 & x_3 \arrow[l, mapsto] \\
\dots & \dots & \dots & \dots
\end{tikzcd}$$

This chain complex is named in our representation `Z_nat_ChainComplex`. Its differential can be easily defined using auxiliary functions as `fun n: nat_as_Setoid => if even_bool n then Var _ n else Unit _ _`. Here, we are using the `Unit` notation for the null element as in L. Pottier' s development. It is not difficult to prove that this morphism satisfies the nilpotency condition (in other words, it is really a differential).

Now, it is possible to define a homotopy operator $h^{(2)}$ on $CC^{(2)}$ built on generators in the same way as the previous differential (but, defined from an element in the module at degree i to an element in the module at degree $i+1$). Obvious morphisms allow us to complete an effective homology from this last free chain complex to the null free chain complex of finite type $FCC^{(0)}$. This last effective homology proves that $CC^{(2)}$ is acyclic.

In order to define a more interesting effective homology we define the free chain complex $CC^{(1)} \oplus CC^{(2)}$ obtained from the direct sum of the two previous chain complexes. Then, it is easy to define an effective homology `Z_x_Z_nat_EffectiveHomology`:

$$\begin{tikzcd}
CC^{(1)} \oplus CC^{(2)} \arrow[loop above, "{(0,h^{(2)})}"] \arrow[r, shift left, "\pi_1"] & FCC^{(1)} \arrow[l, shift left, "{(id,0)}"]
\end{tikzcd}$$

where π_1 is the canonical projection in the first component.

Finally, we consider a free chain morphism between the top chain complexes of `Z_x_Z_nat_EffectiveHomology` and `Id_Z_2x_0_EffectiveHomology` again through the canonical projection in the first component:

$$\begin{tikzcd}
CC^{(1)} \oplus CC^{(2)} \arrow[loop above, "{(0,h^{(2)})}"] \arrow[r, shift left, "\pi_1"] \arrow[d, "\pi_1"'] & FCC^{(1)} \arrow[l, shift left, "{(id,0)}"] \arrow[d, "\alpha'"] \\
CC^{(1)} \arrow[loop below, "0"] \arrow[r, shift left, "id"] & FCC^{(1)} \arrow[l, shift left, "id"]
\end{tikzcd}$$

Then, we can obtain in Coq the cone of this morphism and the effective homology associated to it, named `Example_Cone_EffectiveHomology`, as a particular instance of our general result developed in the previous section:

$$\begin{tikzcd}
Cone(\pi_1) \arrow[loop above, "h^{Ex}"] \arrow[r, shift left, "f^{Ex}"] & Cone(\alpha') \arrow[l, shift left, "g^{Ex}"]
\end{tikzcd}$$

We will use this effective homology instance to make concrete computations in Coq in the following section.

5 Computing with Infinite Data Structures in Coq

Working in the Coq constructive type theoretic setting allows us to obtain from proofs directly computable terms. In the previous section we obtained instances of meaningful examples of all our data structures, so we can now make calculations with them through the associated algorithms (which have been proved correct in Coq). In particular we can make computations within instances of chain complexes of infinite type.

We will use the `vm_compute` Coq tactic for evaluating terms. It computes the goal using the optimized call-by-value evaluation bytecode-based virtual machine [19]. Another option consists in using the Coq extracting code mechanism. Nowadays, the functional languages available as output in Coq are OCaml, Haskell and Scheme [18]. This extracted code should be, in principle, efficient but the presence of dependent types makes it complicated, at least in the Haskell case. Being Scheme a kind of Lisp, its dynamical typing style should be more convenient from this point of view in order to be our target language in which extracts our code. Nevertheless it seems to be the least developed frame (see [19] again). Since Kenzo is implemented in Common Lisp it is clear that the problems encountered with Scheme are important for us if we want to extract code which was directly comparable with the Kenzo code. We do not follow this line in this paper. We explore rather the possibilities of the *internal* execution of Coq terms.

We are going to choose as an example the top chain complex of `Example_Cone_EffectiveHomology`, *i.e.* $Cone(\pi_1)$. This is an example of chain complex of *infinite type*. For instance, we want to compute its differential applied to the element $(5, 7{*}x_4{+}8{*}x_0, 3)$ at degree 2. Since the module at degree 2 of the cone (and, in fact, at *any* degree) is $\mathbb{Z}\oplus\mathbb{Z}[\mathbb{N}]\oplus\mathbb{Z}$, the element $(5, 7{*}x_4{+}8{*}x_0, 3)$ has a component in each module. The first and third components appear simply as integers, because $\mathbb{Z}$ is considered a free module over a singleton which is skipped. On the contrary, elements in the second component are true combinations in $\mathbb{Z}[\mathbb{N}]$ with generator x_i (recall our convention of naming x_i the element i of $\mathbb{N}$). Thus the modules of the cone are not presented as *free* modules, but they are isomorphic to modules freely generated, as it is inferred from the results of Section 3.

The second element of the tuple $(5, 7 * x_4 + 8 * x_0, 3)$ is represented in Coq by `e:= Law (Op (R:= Z_as_Ring) 7 (Var _ (4%nat: nat_as_Setoid))) (Op (R:= Z_as_Ring) 8 (Var _ (0%nat: nat_as_Setoid)))`.

The required Coq code is then the following:

```
Eval vm_compute in
 ((Diff(topCC Example_Cone_EffectiveHomology) 2) (5, e, 3)).
```

and the result returned by Coq is:

```
= (-10, Inv e, 5): topCC Example_Cone_EffectiveHomology 2
```

i.e., $(-10, -(7 * x_4 + 8 * x_0), 5)$. If we apply now the (degree 1) differential to this element we obtain:

```
= (0, Inv (Inv (Law (Op 7 (Unit Z_as_Ring nat_as_Setoid))
                (Op 8 (Unit Z_as_Ring nat_as_Setoid)))), 0)
    : topCC Example_Cone_EffectiveHomology 1
```

or, in plain notation, $(0, -(-(7*()+8*()), 0)$ which it is equal (in the setoid) to the null element. It should be recalled that our formalization of the free module generated by the natural numbers directly use the L. Pottier definition for free modules, and, as a consequence, we are not working with canonical elements on the free modules or with structures which allow a reduction to them.

Now, we focus our attention on homotopy operators, that is to say on morphisms which increase in one unity the degree into the graded module. We use as ambient structures the chain complexes $Cone(\pi_1)$ and $Cone(\alpha')$ introduced in the previous section.

Some examples of homotopy operators for $Cone(\alpha')$, $h = (h_i\colon Cone(\alpha')_i \rightarrow Cone(\alpha')_{i+1})_{i\in\mathbb{Z}}$, are the following:

– $h1 = (h1_i)_{i\in\mathbb{Z}}$, such that $h1_i(a, b) := (0, a)$, $(a, b) \in Cone(\alpha')_i$ for all $i \in \mathbb{Z}$
– $h2 = (h2_i)_{i\in\mathbb{Z}}$, such that $h2_i(a, b) := (b, 0)$, $(a, b) \in Cone(\alpha')_i$ for all $i \in \mathbb{Z}$

Both can be easily implemented in Coq. For example, the first one is represented through:

```
Definition h1': forall i:Z, bottomCC Example_Cone_EffectiveHomology i ->
 bottomCC Example_Cone_EffectiveHomology(i + 1):=
  fun (i:Z)(c: bottomCC Example_Cone_EffectiveHomology i) => (0, fst c).
```

There exist special homotopy operators called *contracting homotopies* which express algorithmically that the chain complex is *acyclic* [24].

Definition 8. *A chain complex is* acyclic *if it is possible to define an effective homology from it to the null chain complex.*

Corollary 1. *Let $CC = (M, d)$ be a chain complex, CC is acyclic if and only if there exists a homotopy operator h defined on CC such that $d \circ h + h \circ d = id$. Such an operator is called* contracting homotopy*.*

We can test if the previous homotopy operators define a contracting homotopy. For instance, the corresponding tactic at degree `i=1` choosing as element `(5, 7): bottomCone 2` for the first candidate is:

```
Eval vm_compute in
(((Diff (bottomCC Example_Cone_EffectiveHomology) 2)[oh](h1 2))[+h]
 ((h1 1)[oh](Diff(bottomCC Example_Cone_EffectiveHomology) 1)))(5, 7).
```

resulting in: `= (0, 0): bottomCC Example_Cone_EffectiveHomology 2.`

For the second homotopy operator over the same element we obtain:

```
Eval vm_compute in
(((Diff (bottomCC Example_Cone_EffectiveHomology) 2)[oh](h2 2))[+h]
 ((h2 1)[oh](Diff(bottomCC Example_Cone_EffectiveHomology) 1)))(5, 7).
```

resulting in: `= (5, 7): bottomCC Example_Cone_EffectiveHomology 2.`

This means that $h1$ is not a contracting homotopy for $Cone(\alpha')$. It could be, anyway, acyclic. The homotopy operator $h2$ could be a candidate for contracting homotopy and, in fact, if we test other elements in other dimensions we always obtain the identity.

Moreover, using the homotopy operator $h2$ and the one h^{Ex} in the effective homology at the end of the previous section, we can define a new homotopy operator over $Cone(\pi_1)$ with the formula $h = h^{Ex} + g^{Ex} \circ h2 \circ f^{Ex}$. Graphically:

$$\begin{tikzcd}
Cone(\pi_1) \arrow[loop above, "h^{Ex}"] \arrow[loop below, "h=h^{Ex}+ g^{Ex}\circ\ h2\ \circ f^{Ex}"] \arrow[r, shift left, "f^{Ex}"] & Cone(\alpha') \arrow[l, shift left, "g^{Ex}"] \arrow[loop above, "h2"]
\end{tikzcd}$$

This homotopy operator can be easily defined in Coq in the following way:

```
Definition h_topCone:
(HomotopyOperator(topCC Example_Cone_EffectiveHomology)):=
 fun n: Z => (h_t_t Example_Cone_EffectiveHomology) n [+h]
  (((g_b_t Example_Cone_EffectiveHomology) n) [oh] (h2 n) [oh]
  ((f_t_b Example_Cone_EffectiveHomology) n)).
```

We can test if it is a candidate to be a contracting homotopy:

```
Eval vm_compute in
(((Diff(topCC Example_Cone_EffectiveHomology) 2)[oh](h_topCone 2))
  [+h]((h_topCone 1)[oh]
   ((Diff(topCC Example_Cone_EffectiveHomology) 1))))(5, e, 3).
```

whose result is an element equal (in the setoid) to `(5, e, 3)`.

The testing with other elements and at other degrees is always successful and this allows us to conjecture that it is really a contracting homotopy.

If that is the case, it could be used to solve a problem that, in general, is undecidable when working with chain complexes of infinite type. If an element x is a cycle (that is to say, $d_n(x) = 0$) and the chain complex is acyclic, then there exists an element z such that $d_{n+1}(z) = x$. Or, in other words, z is a pre-image of x for the differential. Let us compute such a pre-image in our example. To this aim, we choose again $x = (-10, -(7 * x_4 + 8 * x_0), 5)$ as an element at degree 2. We know already it is a cycle, because it has been previously computed. Then, if our homotopy operator h is actually a contracting homotopy, the image $h(x)$ must be a pre-image of x for d (since $dh(x) + hd(x) = x$, but $hd(x) = 0$). We can test in Coq this fact as follows. First we apply the homotopy operator on the element:

```
Eval vm_compute in (h_topCone 2)(-10, Inv e, 5).
```

obtaining an element equal to `(5, e, 0)`. And due to our previous computations we know that this element is indeed in the right pre-image because

```
Eval vm_compute in
((Diff(topCC Example_Cone_EffectiveHomology) 2))(5, e, 0).
```

gives the required element `(-10, Inv e, 5)`.

This behaviour is not accidental. The testing is reflecting a general result relating cones and reductions. Namely:

Proposition 1. *Let (M, N, f, g, h) be a reduction. Then $Cone(f)$ is an acyclic chain complex.*

The constructive proof of this proposition gives exactly the formula we were testing before. Finally, we could proof in Coq that $h2$ and h are indeed contracting homotopies which is now an easy exercise. Also Corollary 1 and Proposition 1 could be formalized in Coq, although more effort is required. Both tasks are proposed as future work.

6 Conclusions and Further Work

In this paper we have presented some examples relating deduction and computing in the Coq proof assistant. Even if constructive type theory always allows, in principle, the modeler to execute terms (by reducing them) this is rarely used in development (or, at least, it is rarely documented). In our case, testing has been worked out in an infinite dimensional setting. Concretely, we have constructed concrete instances of chain complexes of infinite type, we have computed in Coq with their elements, and we have checked some formula producing a contracting homotopy on one of the chain complexes. This testing corresponds to a general theorem that could be, later on, proved in Coq, too.

The chain complexes of infinite type used as examples in this paper are, in some sense, artificial. It can be considered as a demonstration of feasibility. In a future step, we will undertake the implementation in Coq of more meaningful infinite dimensional spaces. Our first candidates will be *loop spaces*. The chain complex associated to a combinatorial loop space (see Kan's G construction in [21]) is of infinite type. Under good conditions, its homology groups are, however, of finite type. Computing these homology groups was one of the first challenges solved by Kenzo (see [24]), and working with them in Coq would be an interesting issue.

One unpleasant aspect of our work is that we are working in a context where combinations are not in normal form. This implies that, once a function has been applied, some work is needed to prove the result is equal to some assumed test value. Several approaches are known to tackle this reduction to canonical form, and we should systematically explore some of them to propose a more comfortable way of doing testing in Coq. Another via to avoid this difficulty could be to give setoids up and work inside the *ssreflect* framework [14].

Another related line is that of code extraction. We should retake the works on going from Coq to Scheme [18], and adapt them to Common Lisp. Since that we have a model (Kenzo [12]) of the programs we would like to extract, the challenge would be to devise Coq statements and proofs in such a way that the extracted programs would be as close as possible to the selected Kenzo fragment.

Finally we could study the possibilities of tools like QuickCheck [7] in our setting. This system allows to test properties of programs automatically by generating a large number of cases (although, up to our knowledge, there is no direct application to Coq code).

Acknowledgement. The authors wish to thank the anonymous reviewers for their useful comments.

References

1. Andrés, M., Lambán, L., Rubio, J.: Executing in Common Lisp, Proving in ACL2. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, pp. 1–12. Springer, Heidelberg (2007)
2. Aransay, J., Ballarin, C., Rubio, J.: A Mechanized Proof of the Basic Perturbation Lemma. J. Autom. Reason. 40(4), 271–292 (2008)
3. Aransay, J., Ballarin, C., Rubio, J.: Generating certified code from formal proofs: a case study in homological algebra. Form. Asp. Comput. 22, 193–213 (2010)

4. Aransay, J., Domínguez, C.: Modelling Differential Structures in Proof Assistants: The Graded Case. In: Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A. (eds.) EUROCAST 2009. LNCS, vol. 5717, pp. 203–210. Springer, Heidelberg (2009)
5. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. In: Coq'Art: The Calculus of Inductive Constructions. Springer, Heidelberg (2004)
6. Boutin, S.: Using reflection to build efficient and certified decision procedures. In: Ito, T., Abadi, M. (eds.) TACS 1997. LNCS, vol. 1281, pp. 515–529. Springer, Heidelberg (1997)
7. Claessen, K., Hughes, J.: QuickCheck: a lightweight tool for random testing of Haskell programs. In: Proceedings of the fifth ACM SIGPLAN International Conference on Functional Programming, SIGPLAN Notices, vol. 35(9), pp. 268–279 (2000)
8. Coquand, T., Spiwack, A.: Constructively finite. In: Contribuciones científicas en honor de Mirian Andrés Gómez. Servicio de Publicaciones de la Universidad de La Rioja (2010)
9. Coquand, T., Spiwack, A.: Towards Constructive Homological Algebra in Type Theory. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, pp. 40–54. Springer, Heidelberg (2007)
10. Domínguez, C., Lambán, L., Rubio, J.: Object-Oriented Institutions to Specify Symbolic Computation Systems. Rairo. Theor. Inf. Appl. 41, 191–214 (2007)
11. Domínguez, C., Rubio, J.: Effective Homology of Bicomplexes, formalized in Coq, https://esus.unirioja.es/psycotrip/archivos_documentos/EHBFC.pdf
12. Dousson, X., Sergeraert, F., Siret, Y.: The Kenzo Program. Institut Fourier, Grenoble (1999), http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/
13. Geuvers, H., Pollack, R., Wiedijk, F., Zwanenburg, J.: A constructive algebraic hierarchy in Coq. J. Symb. Comput. 34(4), 271–286 (2002)
14. Gonthier, G.: Formal Proof: The Four-Color Theorem. Notices of the AMS 55(11), 1382–1393 (2008)
15. Gonthier, G., Mahboubi, A., Rideau, L., Tassi, E., Théry, L.: A Modular Formalisation of Finite Group Theory. In: Schneider, K., Brandt, J. (eds.) TPHOLs 2007. LNCS, vol. 4732, pp. 86–101. Springer, Heidelberg (2007)
16. Jacobson, N.: Basic Algebra II, 2nd edn. W.H. Freeman and Company, New York (1989)
17. Lambán, L., Pascual, V., Rubio, J.: An Object-Oriented Interpretation of the EAT System. Appl. Algebra Eng. Commun. Comput. 14(3), 187–215 (2003)
18. Letouzey, L.: Extraction in Coq: An Overview. In: Beckmann, A., Dimitracopoulos, C., Löwe, B. (eds.) CiE 2008. LNCS, vol. 5028, pp. 359–369. Springer, Heidelberg (2008)
19. LogiCal project. The Coq Proof Assistant (2010), http://coq.inria.fr/
20. Mahboubi, A.: Implementing the cylindrical algebraic decomposition within the Coq system. Math. Struct. Comput. Sci. 17(1), 99–127 (2007)
21. May, P.: Simplicial Objects in Algebraic Topology. Van Nostrand (1967)
22. Nipkow, T., Paulson, L.C., Wenzel, M.T.: Isabelle/HOL: A proof assistant for higher order logic, LNCS, vol. 2283. Springer, Heidelberg (2002)
23. Rubio, J., Sergeraert, F.: Computing with locally effective matrices. Int. J. Comput. Math. 82(10), 1177–1189 (2005)
24. Rubio, J., Sergeraert, F.: Constructive Algebraic Topology. Bull. Sci. math. 126, 389–412 (2002)
25. Sergeraert, F.: The computability problem in Algebraic Topology. Adv. Math. 104, 1–29 (1994)

Formally Verified Conditions for Regularity of Interval Matrices

Ioana Paşca

INRIA Sophia Antipolis
Ioana.Pasca@sophia.inria.fr

Abstract. We propose a formal study of interval analysis that concentrates on theoretical aspects rather than on computational ones. In particular we are interested in conditions for regularity of interval matrices. An interval matrix is called regular if all scalar matrices included in the interval matrix have non-null determinant and it is called singular otherwise. Regularity plays a central role in solving systems of linear interval equations. Several tests for regularity are available and widely used, but sometimes rely on rather involved results, hence the interest in formally verifying such conditions of regularity. In this paper we set the basis for this work: we define intervals, interval matrices and operations on them in the proof assistant Coq, and verify criteria for regularity and singularity of interval matrices.

Keywords: interval analysis, regularity of interval matrices, formal verification, Coq, SSReflect.

1 Interval Analysis and Validation

Interval analysis is a branch of mathematics motivated by its practical applications. It is of use when dealing with inequalities, approximate numbers or error bounds in computations. We use an interval x as the formalization of the intuitive notion of an unknown number $\tilde{x}$ known to lie in x. In interval analysis we do not say that the value of a variable is a certain number, but we say that a value of a variable is in an interval of possible values. This way we cannot be wrong because of rounding errors or method errors, we can only be imprecise by giving an interval for the value that is very big. Thus, when using interval analysis one often says that thanks to the techniques used the result is guaranteed to be within the obtained bounds. This guarantee is given by the nature of the algorithms.

We argue that it is precisely this nature of the algorithms of interval analysis that justifies their formal study in a proof assistant: since we want results that are guaranteed to be correct, we want to make sure the algorithms that produce them act as they are supposed to. There are not a lot of proofs to be done for basic interval arithmetic: correction of addition, multiplication or standard functions have already been studied in formal systems (see for example [15]).

However, there are other types of algorithms that are of interest for the interval analysis community, in particular algorithms for solving linear systems of interval equations. An important issue in this case is establishing that the interval matrix associated to the system is regular. An interval matrix is said to be regular if all scalar matrices included in the interval matrix have non-null determinant. There are a bunch of criteria for testing regularity of interval matrices. Forty of them are listed in a recent paper of Rohn [21]. Among them there are criteria of theoretical interest only (which we shall alternatively call basic criteria) and efficient criteria used in practice. The basic criteria are usually (but not always) easier to understand and to prove correct. They are often not of practical interest but they are used as a basis to obtain more efficient criteria. It is the case of the sufficient conditions for regularity and singularity of interval matrices discussed by Rex and Rohn in [20].

The above reference has been pointed out to us by researchers interested in robotics [16]. They use the results in [20] to establish that a certain interval matrix is regular and then solve the associated system to get the set of valid coordinates for the next position of the robot. This particular robot is designed for providing help in crises situation by lifting a stretcher with an injured person from an accident scene. This is one example of many an such safety critical applications motivate our formal study of interval analysis.

We implement real intervals and a basic interval arithmetic. We use the proof assistant Coq [1,5] and its standard library as well as the SSReflect extension [9] and the libraries it provides. We discuss the choice of implementation and the interesting issues that occurred in section 2. We then generalize concepts and operations to interval matrices in section 3 and we start talking about systems of linear interval equations in section 4, where we mainly show the criteria we proved for checking regularity of interval matrices. The last section mentions formalizations related to our own on matrices and interval arithmetic. It also presents the possible future directions for this work.

Our formalization concerns intervals with real bounds. In practice we use intervals with bounds in some machine representable subset of real numbers, like floating point numbers. In this case operations on intervals also include a rounding step. Even though we are not dealing with such intervals directly, we will often comment in the paper on how our formalization can be used as a model for floating point intervals.

2 Intervals

In this section we define real intervals and operations on intervals as presented in [17] and we describe the Coq formalization for them.

2.1 Definitions

$\mathbb{R}$ denotes the set of real numbers. A real *interval* is a set of the form

$$x = [\underline{x}, \overline{x}] := \{\tilde{x} \in \mathbb{R} \mid \underline{x} \leq \tilde{x} \leq \overline{x}\}$$

where $\underline{x}, \overline{x}$ are elements of $\mathbb{R}$ with $\underline{x} \leq \overline{x}$. In particular intervals are closed and bounded subsets of $\mathbb{R}$ and we can use the standard set theoretic notation. The set of all real intervals is denoted by $\mathbb{IR}$. We use x as notation for a generic interval, $\underline{x}$ or $\inf(x)$ for the lower bound of x and $\overline{x}$ or $\sup(x)$ as the upper bound of x.

An interval is called thin if $\underline{x} = \overline{x}$ and thick if $\underline{x} < \overline{x}$. Thin intervals contain only one real number and we can identify a thin interval with the unique number contained in it. In particular, real numbers need not be distinguished notationally from intervals.

Now we need to find a good way to formalize real intervals in Coq. We want to capture two aspects:

– we have a dual view of intervals: on one hand an interval can be seen as a pair of real numbers representing its lower and upper bounds and on the other hand an interval is the set of real numbers comprised between the two bounds;
– a real number can be seen as an interval.

To achieve this we define an interval as a structure that contains two real numbers inf and sup representing the lower and upper bounds and a proof that the lower bound is smaller than the upper bound.

```
Structure IR : Type := ClosedInt { inf : R ; sup : R ; leq_proof : inf ≤_b sup }.
```

The type of the field leq_proof is inf $\leq_b$ sup which is a proposition obtained by coercing the boolean inf $\leq_b$ sup to the proposition (inf $\leq_b$ sup) $=$ true which is proof irrelevant [11]. To better understand the information this last sentence hides we created a special section for the interested reader. This next section details Coq' s internal mechanisms that played a role in our formalization. We note that this section is not needed for understanding the rest of the paper so the reader may completely skip it and go directly to the section entitled "Getting the expected behavior".

Technical details
Using the proof assistant Coq means we benefit from some of its features.

The type Prop is the type of logical propositions in Coq. We present its features that influenced our choice of implementation in what follows:

– *The type* Prop *does not benefit of strong elimination.*
To make it more clear, in Coq we have data which are in type Type and logical propositions on these data which are in type Prop. Data and propositions do not live at the same level, more precisely we can use data to build another data or a proposition but we cannot build a piece of data from a proposition, we can only build other propositions. In particular, if we have a disjunction $P \vee Q$ in Prop we cannot build a function that returns a certain piece of data based on whether P or Q is satisfied. This corresponds to a disjunction that is not necessarily decidable.
This is why whenever we want to be able to distinguish two cases we use a similar construction under Type: $\{P\} + \{Q\}$ is a set with one element

such that we can determine if this element is P or Q. This corresponds to a disjunction that is effectively decidable. In particular we can build functions that return a certain data based on whether P or Q is true.

– *Proof irrelevance is not automatic for the type* Prop.
This means that two proofs of the same statement may not be equal. This can produce undesired effects when we have terms that depend on proofs. It is the case of our intervals, as an interval is a triplet (inf, sup, leq_proof), where leq_proof is a proof and thus belongs to type Prop. Now, take the intervals $x = (1, 2, \mathsf{leqx})$ and $z = (1, 2, \mathsf{leqz})$. To show that $x = z$ we not only have to show that $1 = 1$ and $2 = 2$ but also that $\mathsf{leqx} = \mathsf{leqz}$. The latter is generally not provable unless we have at least a weak version of proof irrelevance.
However, there are propositions for which we can show they have a unique proof. It is the case of a proposition expressing equality of two booleans (or, more generally, of two terms of a type equipped with a decidable equality) [11].

The standard library Reals
Coq provides an axiomatic definition of the real numbers. The formalization is based on 17 axioms which introduce the reals as a complete, archimedean, ordered field that satisfies the least upper bound principle. This choice of implementation has as positive effect the fact that we can handle real numbers in a manner similar to that of math books on classical real analysis. In particular, we can reason on cases thanks to the trichotomy axiom: for two real numbers x, y exactly one of the following relations holds: $x < y$ or $x = y$ or $x > y$. These relations have all type Prop but the disjunction is put in Type (we have a term of type $\{x < y\} + \{x = y\} + \{x > y\}$) which means we can define data by distinguishing cases in this disjunction. In particular it means we can define a boolean function $(x, y) \mapsto x \leq_b y$ that is true when $x \leq y$ and false otherwise.

As we saw in the previous paragraph, the proposition $\mathsf{x} \leq_b \mathsf{y} = \mathsf{true}$ has only one proof.

The coercion mechanism implemented in Coq allows us to say a certain type is a subtype of another type. A coercion is a function from the subtype to the type that is automatically inserted by the system. For example, we can use a natural injection from natural numbers to the real numbers as a coercion. Then, everytime the system expects a real but gets a natural instead, it will automatically insert this coercion to get a real. A coercion is not displayed by the pretty-printer, so its use is mostly transparent to the user.

An example that is of interest to us is the coercion from booleans to propositions. We coerce a boolean b to the proposition $\mathsf{b} = \mathsf{true}$ (this is the approach taken in the SSReflect extension of Coq).

To summarize everything, $\mathsf{inf} \leq_b \mathsf{sup}$ (the type of the field leq_proof in our definition of intervals) is a proposition obtained by coercing the boolean $\mathsf{inf} \leq_b \mathsf{sup}$ to the proposition $(\mathsf{inf} \leq_b \mathsf{sup}) = \mathsf{true}$ which is proof irrelevant, therefore any two proofs of this proposition will be equal.

Getting the expected behavior

Thanks to our choice of implementation we can prove that equality of two intervals is equivalent to the equality of the respective bounds. It is independent of the proof that inf $\leq_b$ sup.

```
Lemma eq_intervalP : forall x z : IR, x = z ↔ inf x = inf z ∧ sup x = sup z.
```

So our intervals can be viewed as pairs of real numbers. We can also view them as sets of real numbers by using Coq's coercion mechanism. Sets are defined by predicates and belonging to a set means satisfying the predicate. We coerce an interval to the predicate on real numbers that asserts that a real is between the lower and the upper bounds of the interval. This coercion allows us to transparently use our intervals as sets of real numbers. We also define a coercion from a real number to the corresponding thin interval, so we can directly use real numbers as intervals.

We define the midpoint and the radius of an interval:

$$x_c = \mathrm{mid}(x) := \frac{\overline{x} + \underline{x}}{2} \quad ; \quad \Delta_x = \mathrm{rad}(x) := \frac{\overline{x} - \underline{x}}{2}$$

An interval x is equal to the interval $[x_c - \Delta_x, x_c - \Delta_x]$. The membership relation can also be expressed as

$$\tilde{x} \in x \Leftrightarrow |\tilde{x} - x_c| \leq \Delta_x$$

The corresponding Coq lemma is

```
Lemma in_mid_rad: forall (x: IR) (tx: R), tx \in x ↔ Rabs (tx − mid x) ≤ rad x.
```

In the above statement, \in is an infix notation for satisfying a predicate, or, equivalently, belonging to the set described by that predicate. This lemma illustrates how the coercion mechanism lets us transparently use an interval as a set.

2.2 Operations on Intervals

We define the elementary operations on intervals (addition, opposite, multiplication) by giving the explicit formulas to compute their bounds. To avoid heavy notations we use the same symbols for operations on numbers and on intervals.

$$x + z := [\underline{x} + \underline{z}, \overline{x} + \overline{z}]$$

$$-x := [-\overline{x}, -\underline{x}]$$

$$xz := [\min(\underline{xz}, \underline{x}\overline{z}, \overline{x}\underline{z}, \overline{xz}), \max(\underline{xz}, \underline{x}\overline{z}, \overline{x}\underline{z}, \overline{xz})]$$

We define separately the multiplication of an interval by a scalar, even though this is equivalent to the multiplication by a thin interval :

$$ax := [\min(a\underline{x}, a\overline{x}), \max(a\underline{x}, a\overline{x})]$$

The reason for this choice is that multiplication of an interval by a scalar enjoys more algebraic properties than interval multiplication in general. When using these properties it is more convenient if they are attached to a specific operation than if we have to provide a proof that the interval is thin each time we use them.

In implementing our operations we take into account that our definition of intervals contains a proof that the lower bound is smaller than the upper bound. We need to provide these proofs before actually defining the operations. For the operations we are considering this is not a big effort as they are straightforward. To illustrate our treatment of elementary operations we take the example of *addition*. We give the proof and define addition according to the formula above:

```
Lemma add_i_wd : forall x z, inf x + inf z ≤_b sup x + sup z.
Definition add_i x z :=
  @ClosedInt (inf x + inf z) (sup x + sup z) (add_i_wd x z).
```

The sum of two intervals can also be characterized by:

$$x + z = \{\tilde{x} + \tilde{z} \mid \tilde{x} \in x, \tilde{z} \in z\} \tag{1}$$

We want to show the equivalence of the two characterizations. We proceed by proving double inclusion of the corresponding sets. We have one inclusion that is straightforward:

$$\{\tilde{x} + \tilde{z} \mid \tilde{x} \in x, \tilde{z} \in z\} \subseteq [\underline{x} + \underline{z}, \overline{x} + \overline{z}] \tag{2}$$

as everytime we have two real numbers $\tilde{x}, \tilde{z}$ with $\tilde{x} \in x$ (which means $\underline{x} \leq \tilde{x} \leq \overline{x}$) and $\tilde{z} \in z$ (which means $\underline{z} \leq \tilde{z} \leq \overline{z}$) then their sum $\tilde{x} + \tilde{z} \in [\underline{x} + \underline{z}, \overline{x} + \overline{z}]$ which is by definition $x + z$.

The other inclusion is less straightforward:

$$[\underline{x} + \underline{z}, \overline{x} + \overline{z}] \subseteq \{\tilde{x} + \tilde{z} \mid \tilde{x} \in x, \tilde{z} \in z\}$$

We need to show that each time a number belongs to the sum of two intervals x and z, than there exists $\tilde{x} \in x$ and $\tilde{z} \in z$ such that our number is written as $\tilde{x} + \tilde{z}$. The difficulty comes form the fact that the decomposition of a number in a sum is not unique. To give a decomposition of a real $s \in x + z$ in an appropriate sum we consider the following cases:

$$s = \begin{cases} \underline{x} + (s - \underline{x}) & \text{, if } s \in [\underline{x} + \underline{z}, \underline{x} + \overline{z}] \text{ with } \underline{x} \in x, (s - \underline{x}) \in z \\ (s - \overline{z}) + \overline{z} & \text{, if } s \in [\underline{x} + \overline{z}, \overline{x} + \overline{z}] \text{ with } (s - \overline{z}) \in x, \overline{z} \in z \end{cases}$$

The proof of equality (1) does not appear in standard books of interval analysis, as it is clear for the trained mathematician that the equality is trivially true. However, in a formal system we needed to go into some detail to show this equality. We remark also that equality (1) does not hold for an interval arithmetic that uses outward rounding of the interval bounds, like, for example, floating point interval arithmetic. In this case only the first inclusion holds (relation (2)).

Addition on intervals enjoys nice properties: it is associative, commutative, accepts the thin interval 0 as a neutral element. This means that the set of real intervals with addition has a commutative monoid structure. This will ease our work, as general theorems concerning the commutative monoid structure are directly available from the SSReflect libraries, in particular we will be able to use lemmas concerning indexed operations when defining operations on interval matrices as we shall see in section 3. We remark that other properties are not satisfied. For example, an interval that is not thin does not have an opposite with respect to the neutral element, which means addition on intervals is not a group operation. This also means that intervals with addition and multiplication do not form a ring. This fact will play a role in our manipulation of interval matrices (see section 3).

Properties relating the bounds of an interval, the center, the radius and operations on intervals usually simplify to straightforward properties of real numbers. Such proofs can often be discarded by automatic procedures, like ring or field for dealing with equalities and fourier for dealing with inequalities over the real numbers in Coq.

3 Matrices

To describe interval matrices we use the SSReflect library which contains a formalization of matrices with elements of an arbitrary type T. For operations on rows and columns (e.g deleting a row, swapping two rows etc) no additional properties are required for T. Once one starts talking about operations on matrices like addition or multiplication, the type of elements T has to be a ring. The library provides all the basic operations and their properties, the notions of determinant and inverse. Details on the matrix library can be found in [8,3]. As we saw in the previous section, intervals do not have a ring structure, so we will need to redefine operations for interval matrices (section 3.2). Nevertheless, all results in the generic SSReflect matrix library can directly be used for real matrices, as real numbers have a ring structure. There are still other notions, specific to real matrices that are not part of the library. We detail them in the following section.

3.1 More Results on Real Matrices

We generalize some basic real number concepts to matrices in a componentwise manner. If $A = [A_{ij}]_{m\times n}$ is a real matrix, the absolute value function $|A| = [|A_{ij}|]$. Similarly, a comparison relation $\omega \in \{\leq, <, \geq, >\}$ for two matrices A and B is given by $A\ \omega\ B \Leftrightarrow \forall ij, A_{ij}\ \omega\ B_{ij}$.

We need norms for matrices and we reuse the author's previous developments described in [19], where a generic matrix norm is defined. This paper proves general properties on this norm and gives an instantiation to the maximum row sum norm: $\|A\| = \max_i \sum_j |A_{ij}|$.

We define what it means for a matrix to be:

- symmetric : $\forall ij, A_{ij} = A_{ji}$
- positive definite (for square matrices) : $\forall x \in \mathbb{R}^n, x \neq 0 \Rightarrow x^T A x > 0$

We define the eigenvalues of a square matrix as the roots of its characteristic polynomial. The definition of the latter is available in the SSReflect libraries and described in [3].

```
Definition eigenv (A: 'M[R]_n) := root (char_poly A).
```

Then the spectral radius is defined as the maximum of the absolute values of the eigenvalues. We use standard notations: λ usually denotes an eigenvalue and $\rho(A)$ denotes the spectral radius of A. A collection of basic results are established for eigenvalues:

- for each eigenvalue λ there is an associated eigenvector (a vector $x \neq 0$ such that $Ax = \lambda x$),
- the absolute value of an eigenvalue is smaller than the norm of the matrix,
- the spectral radius is smaller than the norm of the matrix,
- all eigenvalues of a positive definite matrix are positive.

Some other results on eigenvalues that we need but that we have not formalized yet are the Perron-Frobenius theorem and properties of Rayleigh's quotient. The formalization of the Perron-Frobenius theorem is more involved and it is the topic of an independent study. Rayleigh's quotient is the quantity $x^T Ax/x^T x$ for a non-null vector x and the result we need is $\forall x, x \neq 0, \lambda_{\min} \leq x^T Ax/x^T x \leq \lambda_{\max}$. This proof is not complicated. It requires some concepts of multivariate analysis which the author has studied in previous work [19].

3.2 Interval Matrices

An interval $m \times n$ matrix is a $m \times n$ matrix with interval elements

$$A = [A_{ij}]_{m\times n}, \ A_{ij} \in \mathbb{IR}.$$

Interval vectors are not treated separately: a vector is a special kind of matrix. We have column vectors, they are therefore $n \times 1$ matrices.

An interval matrix is interpreted as a set of real matrices by the convention

$$A = \{\tilde{A} \in M(\mathbb{R})_{m\times n} \mid \tilde{A}_{ij} \in A_{ij}, i = 1, \ldots, m, j = 1, \ldots, n\}.$$

```
Definition inSetm (A : 'M[IR]_(m, n)) (tA : 'M[R]_(m, n) :=
  forall i j, tA i j \in A i j.
```

The concepts we described for intervals generalize to interval matrices, usually componentwise. This allows us to relate certain real matrices to each interval matrix.

$$\underline{A} = \inf(A) := [\underline{A}_{ij}] \qquad \overline{A} = \sup(A) := [\overline{A}_{ij}]$$

$$A_c = \text{mid}(A) := [\text{mid}(A_{ij})] \qquad \varDelta_A = \text{rad}(A) := [\text{rad}(A_{ij})]$$

An example of Coq definition:

```
Definition minf (A : 'M[R]_(m, n)) := \matrix_(i, j) inf (A i j).
```

To talk about operations with interval matrices we recall that in order to use the generic operations from the SSReflect library we need to have a ring structure on the underlying type. But we saw in the previous section that real intervals with addition and multiplication do not form a ring. So we need to define specific operations for interval matrices. To illustrate how this is done in Coq we give the example of multiplication of a matrix by a vector:

```
Definition mmul_i (A : 'M[IR]_(m, n)) (x : 'M[IR]_(n, 1)) :=
  \col_i \big[add_i / 0 ]_j mul_i (A i j) (x j).
```

In the above definition \col_ is the column vector, \big[add_i / 0]_ is an indexed sum of intervals: $\sum_j A_{ij}x_j$. Properties are established by work with indexed operations using the dedicated SSReflect library [2]. Here the fact that interval addition has a commutative monoid structure comes in handy, as many theorems on indexed operations apply straightforwardly. Similar to the characterization for addition of two intervals, we show the characterization for the multiplication of an interval matrix by a real vector.

$$A\tilde{x} = \{\tilde{A}\tilde{x} \mid \tilde{A} \in A\} \tag{3}$$

Here the same issues arise as for the proof of relation (1). We note that this result is not true in general, for the multiplication of an interval matrix by an interval vector. Take for example:

$A = \begin{pmatrix} 1\,1 \\ 0\,1 \end{pmatrix}$ and $x = \begin{pmatrix} [-1,0] \\ [1,2] \end{pmatrix}$ then we have $\begin{pmatrix} 0 \\ 2 \end{pmatrix} \in Ax = \begin{pmatrix} [0,2] \\ [1,2] \end{pmatrix}$ but $\begin{pmatrix} 0 \\ 2 \end{pmatrix}$ not of the form $\tilde{A}\tilde{x}$ with $\tilde{A} \in A, \tilde{x} \in x$ because A is a thin matrix, therefore $\forall \tilde{A} \in A, \tilde{A} = A$ and solving $A\tilde{x} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$ gives $\tilde{x} = \begin{pmatrix} 2 \\ -2 \end{pmatrix} \notin x$.

4 Regularity of Interval Matrices

In this section we introduce systems of linear interval equations and consider some of their basic aspects, in particular conditions for regularity of the interval matrices associated to these systems. The proofs are taken from [17,20].

4.1 The Solution Set of a System of Linear Interval Equations

A system of linear interval equations with coefficient matrix $A \in M(\mathbb{IR})_{m\times n}$ and right-hand side $b \in \mathbb{IR}^m$ is defined as the family of linear systems of equations

$$\tilde{A}\tilde{x} = \tilde{b} \text{ with } \tilde{A} \in A, \tilde{b} \in b$$

The *solutions set* of such a system is given by:

$$\Sigma(A, b) := \{\tilde{x} \in \mathbb{R}^n \mid \exists\tilde{A} \in A, \exists\tilde{b} \in b \text{ such that } \tilde{A}\tilde{x} = \tilde{b}\}$$

```
Definition sigma_sol A b := fun x ⇒
   exists tA, inSetm A tA ∧ exists tb, inSetm b tb ∧ tA *m x = tb.
```

We begin by giving some alternative characterizations of the solution set:

$$\Sigma(A,b) = \{\tilde{x} \in \mathbb{R}^n \mid A\tilde{x} \cap b \neq \emptyset\} = \{\tilde{x} \in \mathbb{R}^n \mid 0 \in A\tilde{x} - b\}$$

We do not detail the entire proof, but we give an example of a proof step where equalities like relation (1) or (3) intervene:
" if $A\tilde{x} \cap b \neq \emptyset$ then $A\tilde{x} \cap b$ contains some $\tilde{b} \in \mathbb{R}^m$; clearly $\tilde{b} \in b$ and by relation (3), $\tilde{b} = \tilde{A}\tilde{x}$ for some $\tilde{A} \in A$ ".

As a corollary we have a result by Oettli and Prager for the characterization of the solution set:

$$\tilde{x} \in \Sigma(A,b) \Leftrightarrow |A_c\tilde{x} - b_c| \leq \Delta_A|\tilde{x}| + \Delta_b.$$

In what follows we will only be interested in square matrices $A \in M(\mathbb{IR})_{n\times n}$. In the study of $\Sigma(A,b)$ the regularity of the interval matrix A plays an important role, for example in establishing that $\Sigma(A,b)$ is non-empty and bounded.

4.2 Basic Regularity Criteria

The interval matrix A is called *regular* if each scalar matrix $\tilde{A} \in A$ is nonsingular (which means $\det \tilde{A} \neq 0$), and it is said to be singular otherwise.

```
Definition regular (A : 'M[IR]_n) := forall tA, inSetm A tA → \det tA <> 0.
Definition singular (A : 'M[IR]_n) := exists tA, inSetm A tA ∧ \det tA = 0.
```

We first recall a characterization of regularity for real matrices that is available in the SSReflect matrix library:

$$\forall \tilde{A} \in M(\mathbb{R})_{m\times n}, \det \tilde{A} \neq 0 \Leftrightarrow \forall \tilde{x} \in \mathbb{R}^n, \tilde{A}\tilde{x} = 0 \Rightarrow \tilde{x} = 0. \tag{4}$$

Based on the previous proofs we can give criteria for checking regularity of interval matrices.

Criterion 1. *A is regular if and only if* $\forall \tilde{x} \in \mathbb{R}^n, 0 \in A\tilde{x} \Rightarrow \tilde{x} = 0$.

Criterion 2. *A is regular if and only if* $\forall \tilde{x} \in \mathbb{R}^n, |A_c\tilde{x}| \leq \Delta_A|\tilde{x}| \Rightarrow \tilde{x} = 0$.

In the same terms we can express singularity:

Criterion 3. *A is singular if and only if $\exists \tilde{x} \in \mathbb{R}^n, x \neq 0$ such that*

$$|A_c\tilde{x}| \leq \Delta_A|\tilde{x}|. \tag{5}$$

Moreover, we can build a singular matrix from a solution of the inequation 5. Let $\tilde{x} \neq 0$ be such a solution.

We can consider the vectors $y, z \in \mathbb{R}^n$ defined by

$$y_i = \begin{cases} (A_c\tilde{x})_i/(\Delta_A|\tilde{x}|)_i & \text{, if } (\Delta_A|\tilde{x}|)_i \neq 0, \\ 1 & \text{, if } (\Delta_A|\tilde{x}|)_i = 0 \end{cases} \qquad z_j = \begin{cases} 1 & \text{, if } \tilde{x}_j \geq 0, \\ -1 & \text{, if } \tilde{x}_j < 0 \end{cases}$$

Then for the matrix $\tilde{A}$ given by

$$\tilde{A}_{ij} = (A_c)_{ij} - y_i z_j (\Delta_A)_{ij}$$

we have $\tilde{A} \in A$ and $\tilde{A}\tilde{x} = 0$ for $\tilde{x} \neq 0$, then from 4 we get $\det \tilde{A} = 0$.

The criteria described above are not convenient in practice. They are important from a theoretical point of view as they can serve as basis for deriving verifiable regularity criteria, where by verifiable we mean that there are known algorithms that can perform the verification of our criterion.

4.3 Verifiable Regularity Criteria

We present verifiable criteria that are based on checking positive definiteness of a matrix, computing the midpoint inverse and computing eigenvalues.

Generally the proofs for these criteria follow rather naturally from the proofs presented in the previous sections on real matrices and on basic regularity criteria. To illustrate this we give a criterion that establishes regularity by a positive definiteness check and we detail its proof.

Criterion 4. *If the matrix*

$$A_c^T A_c - \|\Delta_A^T \Delta_A\| I$$

is positive definite for some consistent matrix norm $\|\cdot\|$*, then* A *is regular.*

Proof. We do a proof by contradiction. We suppose that A is singular, so by Criterion 3 we get that there exists an $x \neq 0$ such that $|A_c x| \leq \Delta_A |x|$. We may normalize x to achieve $\|x\|_2 = 1$.

Then we have

$x^T A_c^T A_c x \leq |A_c x|^T |A_c x|$ - by properties of transpose and absolute value

$|A_c x|^T |A_c x| \leq (\Delta_A |x|)^T \Delta_A |x|$ - by hypothesis

$(\Delta_A |x|)^T \Delta_A |x| = |x|^T \Delta_A^T \Delta_A x$ - by properties of the transpose

$|x|^T \Delta_A^T \Delta_A x \leq \lambda_{\max}(\Delta_A^T \Delta_A)$ - by properties of Rayleigh's quotient

$\lambda_{\max}(\Delta_A^T \Delta_A) \leq \rho(\Delta_A^T \Delta_A)$ - by definition of the spectral radius

$\rho(\Delta_A^T \Delta_A) \leq \|\Delta_A^T \Delta_A)\|$ - by properties relating spectral radius to norm

$\|\Delta_A^T \Delta_A)\| = \|\Delta_A^T \Delta_A)\|(x^T x)$ - by hypothesis that $1 = (\|x\|_2)^2 = x^T x$.

Reading the beginning and the end we get

$$x^T A_c^T A_c x \leq \|\Delta_A^T \Delta_A)\|(x^T x)$$

This is equivalent to

$$x^T(A_c^T A_c - \|\Delta_A^T \Delta_A)\|I)x \leq 0$$

which means that the matrix $(A_c^T A_c - \|\Delta_A^T \Delta_A)\|I)$ is not positive definite, a contradiction to the hypothesis. **Qed.**

The above proof gives an idea of how we prove such criteria based on concepts we previously described. We will not detail the rest of the proofs.

We formulate another criterion in terms of the approximate midpoint inverse R. This is very convenient as in practice we generally have the inverse computed in finite precision arithmetic which may affect validity of criteria given in terms of the exact midpoint inverse A_c^{-1}. However, we present such criteria also, as Corollary 1 and Corollary 2.

Criterion 5. *If the following inequality holds*

$$\rho(|I - RA_c| + |R|\Delta_A) < 1$$

for an arbitrary matrix R, then A is regular.

In particular, if R is the midpoint inverse A_c^{-1} then we get:

Corollary 1. *If A_c is regular and $\rho(|A_c^{-1}|\Delta_A) < 1$ then A is regular.*

Similarly we can give a criterion for checking singularity based on the approximate midpoint inverse.

Criterion 6. *If there exist a matrix R such that*

$$(I + |I - A_c R|)_j \leq (\Delta_A |R|)_j$$

for some $j \in \{1, \dots, n\}$, then A is singular.

Corollary 2. *If A_c is regular and $\max_j(\Delta_A |A_c^{-1}|)_{jj} \geq 1$, then A is singular.*

We give a criterion that ensures regularity at the cost of evaluating eigenvalues for symmetric matrices:

Criterion 7. *If the following inequality holds*

$$\lambda_{max}(\Delta_A^T \Delta_A) < \lambda_{min}(A_c^T A_c)$$

then A is regular.

We formalized all regularity criteria described by [20] and one singularity criterion. We did not concentrate on more singularity proofs as it is not of much practical interest.

5 Conclusion

We presented a formal development in interval analysis: we defined intervals and interval matrices, we formalized their properties and properties of real matrices, we proved correct regularity criteria for interval matrices: some basic regularity criteria as well as three criteria verifiable in practice. Our intention was to show on one hand how we implemented the basic concepts that we work with and on the other hand how far we got in the formalization, what sort of criteria we managed to verify. The source files corresponding to these proofs are available on-line : http://www-sop.inria.fr/marelle/Ioana.Pasca/interval

A big part of our effort was spent on : providing a formalization of real intervals and providing properties of real matrices. This is not very surprising as most criteria actually concern some real matrices associated to the interval matrix and their properties. In other proof assistants there are developments concerning both properties of matrices and of interval arithmetic. For example, work on matrices in Isabelle is described in [18] and in HOL Light in [10]. A development in Coq other than the one we used is presented in [14]. Interval arithmetic in Coq has been approached in [7,15], while in PVS we have [6] and in Isabelle [12].

All these developments on interval arithmetic have as primary concern doing correct computation. Our work is different in that its main purpose is to establish more involved theoretical properties. For now we are not considering the computational aspect. In the long run, however, this work should serve as a theoretical basis to verify properties of actual computation. We note that it is a commonly used approach to have properties verified on a abstract model of real numbers or intervals and use them to validate a concrete implementation of real or interval arithmetic. For example the interval arithmetic tool Gappa does computations on machine floating point numbers and uses a Coq library on abstract reals to validate these computations [4]. In [13] a description on abstract reals of properties for Newton's method is used to verify computations with Newton's method on computable reals.

The study of regularity of interval matrices was motivated by needs of researchers interested in robotics who use such criteria in their daily work. We managed to formally verify criteria like 5, 7, 4 which correspond to conditions used in practice. However, to get fully verified conditions, the algorithms performing the verification of these conditions should also be verified. Work in this direction is already being done, for example [8] describes the verification of the LUP decomposition algorithm.

We wish to continue this work in two directions. The first one is to provide the necessary tools to continue with formalizations on interval matrices. In particular, computing the inverse of an interval matrix is closely related to issues on regularity. This work will in turn open the road to verify algorithms for solving linear systems of interval equations. The other possible direction of this work is to abstract on the type of intervals. For now all proofs are done for intervals with real bounds. The interesting work will be to have intervals with rational bounds or even better, with floating point bounds. Since floats and rationals are themselves real numbers and intervals are computed by outward rounding, the results presented here should apply. For example, let us suppose we have F an interval matrix where the ends of the intervals are floating point numbers and we managed to verify a certain regularity criterion for F. This criterion says that all real matrices included in F are regular. In particular all floating point matrices included in F are regular. However, this is still an issue to investigate: to which degree criteria proved for ideal arithmetic are still suitable in the floating point world.

References

1. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. In: Coq'Art:the Calculus of Inductive Constructions. Springer, Heidelberg (2004)
2. Bertot, Y., Gonthier, G., Biha, S.O., Paşca, I.: Canonical Big Operators. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2008. LNCS, vol. 5170, pp. 86–101. Springer, Heidelberg (2008)
3. Biha, S.O.: Formalisation des mathématiques: une preuve du théorème de Cayley-Hamilton. In: Journées Francophones des Langages Applicatifs, pp. 1–14 (2008)
4. Boldo, S., Filliâtre, J.-C., Melquiond, G.: Combining Coq and Gappa for Certifying Floating-Point Programs. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) Calculemus 2009, MKM 2009. LNCS (LNAI), vol. 5625, pp. 59–74. Springer, Heidelberg (2009)
5. Coq development team. The Coq Proof Assistant Reference Manual, version 8.2 (2009)
6. Daumas, M., Lester, D., Muñoz, C.: Verified real number calculations: A library for interval arithmetic. IEEE Trans. Computers 58(2), 226–237 (2009)
7. Daumas, M., Melquiond, G., Muñoz, C.: Guaranteed proofs using interval arithmetic. In: Montuschi, P., Schwarz, E. (eds.) Proceedings of the 17th IEEE Symposium on Computer Arithmetic, Cape Cod, MA, USA, pp. 188–195 (2005)
8. Garillot, F., Gonthier, G., Mahboubi, A., Rideau, L.: Packaging mathematical structures. In: Urban, C. (ed.) TPHOLs 2009. LNCS, vol. 5674, pp. 327–342. Springer, Heidelberg (2009)
9. Gonthier, G., Mahboubi, A.: A small scale reflection extension for the coq system. INRIA Technical report, `http://hal.inria.fr/inria-00258384`
10. Harrison, J.: A HOL theory of euclidean space. In: Hurd, J., Melham, T.F. (eds.) TPHOLs 2005. LNCS, vol. 3603, pp. 114–129. Springer, Heidelberg (2005)
11. Hedberg, M.: A Coherence Theorem for Martin-Löf's Type Theory. Journal of Functional Programming 8(4), 413–436 (1998)
12. Holzl, J.: Proving Inequalities over Reals with Computation in Isabelle/HOL. In: International Workshop on Programming Languages for Mechanized Mathematics Systems, pp. 38–45 (2009)
13. Julien, N., Paşca, I.: Formal Verification of Exact Computations Using Newton's Method. In: Urban, C. (ed.) TPHOLs 2009. LNCS, vol. 5674, pp. 408–423. Springer, Heidelberg (2009)
14. Magaud, N.: Programming with Dependent Types in Coq: a Study of Square Matrices (January 2005) (Unpublished); A preliminary version appeared in Coq contributions
15. Melquiond, G.: Proving bounds on real-valued functions with computations. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 2–17. Springer, Heidelberg (2008)

16. Merlet, J.-P.: Interval analysis for certified numerical solution of problems in robotics. International Journal of Applied Mathematics and Computer Science 19, 399–412 (2009)
17. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge University Press, Cambridge (1990)
18. Obua, S.: Proving bounds for real linear programs in isabelle/hol. In: Hurd, J., Melham, T. (eds.) TPHOLs 2005. LNCS, vol. 3603, pp. 227–244. Springer, Heidelberg (2005)
19. Pasca, I.: Formal Proofs for Theoretical Properties of Newton's Method (2010), INRIA Research Report RR-7228, http://hal.inria.fr/inria-00463150/en/
20. Rex, G., Rohn, J.: Sufficient conditions for regularity and singularity of interval matrices. SIAM Journal on Matrix Analysis and Applications 20, 437–445 (1998)
21. Rohn, J.: Forty necessary and sufficient conditions for regularity of interval matrices: A survey. Electronic Journal of Linear Algebra 18, 500–512 (2009)

Reducing Expression Size Using Rule-Based Integration

David J. Jeffrey and Albert D. Rich

[1] Department of Applied Mathematics, The University of Western Ontario, London, Canada
djeffrey@uwo.ca
[2] 62-3614 Loli'i Way, Kamuela, Hawaii, USA
Albert_Rich@msn.com

Abstract. This paper describes continuing progress on the development of a repository of transformation rules relevant to indefinite integration. The methodology, however, is not restricted to integration. Several optimization goals are being pursued, including achieving the best form for the output, reducing the size of the repository while retaining its scope, and minimizing the number of steps required for the evaluation process. New optimizations for expression size are presented.

1 Introduction

The methods of integration can be conveniently divided into several categories.

– Look-up tables. These are collections or databases, such as [4], which try to list all possible integrals, each in a general form. Many special cases are also listed separately.
– Rule-based rewriting. The databases used are smaller than those for the look-up tables. They contain rules for transforming a given integral into one or more simpler integrals, together with rules for completing the evaluation in terms of known functions.
– Algorithmic methods. Under this heading, we include Risch integration, Rothstein-Trager-Rioboo integration, and others, which require extended computations.

A table of reduction rules can serve more roles than merely the database for an evaluation system; it can also serve as a repository for mathematical knowledge. Each rule can be annotated with information on its derivation, with references to the literature, and so on. An evaluation system can display transformations as they are used, for the information of users.

Here, we consider the repository of transformation rules for indefinite integrals that is described in [5,6]. We shall refer to it by the acronym RUBI: RUle-Based Integrator. We review the general state of the repository and then focus on particular aspects, namely, its efficiency, and the selection of output forms. Procedures have been written in MATHEMATICA to implement the evaluation of integrals using the repository, and these procedures have been the basis of testing and comparisons.

The role of rule-based approaches, and how they should complement algorithmic methods, can be a subject of debate. For example, Fateman wrote, as part of a review of the system MATHEMATICA [2]

> "Yet the evidence of the past several decades casts strong doubt on the idea that an efficient version of mathematical knowledge can be imparted to a symbolic system *primarily* by rule-transformations on trees."
> **Richard Fateman (1992)**

Owing to poor implementations, rule-based systems have a reputation for being inefficient and plagued by endless loops. This paper, however, describes the crafting of a rule-based repository (RUBI) that is compact, efficient, transparent and modular. We shall not address the combining of RUBI with algorithmic approaches, as would be required to arrive at a full integration system, but concentrate on the constructing of a database of knowledge, with examples of how it performs in practice.

It must be emphasized again that what is not being described is a scheme for table look-up. Such schemes were described, for example, in [3]. The approach there was to consider data structures and search techniques which would allow them to encode all the entries in reference books such as [1]. Adopting this approach for integration — or *a fortiori* for all simplification — would result in huge databases which would be unwieldy to maintain, debug and utilize. The set of rules described here is relatively compact, verifiable and efficient.

2 Basic Details of System

Here, we give a brief account of the RUBI system. At the time of testing, it consisted of 1377 reduction rules. Each rule is an entry in the database and consists of the following fields.

– Conditions under which the reduction rule is applied. These conditions result either from requirements for the validity of the transformation, or from requirements that the transformation be a reduction, meaning a step towards evaluation of the integral.
– The transformation from one expression to another.
– Comments recording the source of the rule (usually a reference to one or more standard reference books) or other useful information.

It should be noted that programming constructs, such as loops or branching statements are never used. Examples of these rules are given below in section 4 (without the comments).

The total size of the database (including comments) was 554 Kb. This is an uncompressed text file. About one third of the file consists of comment text. Procedures using the pattern-matching functions of MATHEMATICA were written to apply the database to the test problems, and no attempt is made to measure the sizes of subsystems of Mathematica used.

The construction and selection of the rules is based on the principle of mutual exclusivity. For a database of reduction rules to be properly defined, at most one of the rules can be applicable to any given expression. Mutual exclusivity is critical to ensuring that rules can be added, removed or modified without affecting the other rules. Such stand-alone, order-independent rules make it possible to build a rule-based repository of knowledge incrementally and as a collaborative effort.

3 Performance Comparison with Other Systems

In order to provide quantitative evidence of the benefits of rule-based integration, we present a comparison of the performance of various computer algebra systems on a test suite containing 7927 problems. The performance measure is based on the validity and simplicity of the expressions returned. Other performance measures, such as speed, have been measured, but direct comparisons can at present be made only with Mathematica, and so here the emphasis is on expression size, until a variety of platforms can be compared for speed[1]. We note in passing, however, that smaller expression sizes will also contribute to speed advantages.

The expression given for each integral was checked against the simplest form, obtained from published integral tables, or from integration by hand. For each problem, the integration result was differentiated by the system being tested, the derivative subtracted from the integrand, and the system asked to test whether the result was zero. Each test yielded one of the following 4 judgements:

- **Optimal:** Correct and close to the best form.
  Example: $\displaystyle\int \frac{5x^4\,\mathrm{d}x}{(1+x)^6} = \frac{x^5}{(1+x)^5}$.
- **Messy:** Correct, but the expression is overly large. E.g.
  $\displaystyle\int \frac{5x^4\,\mathrm{d}x}{(1+x)^6} = -\frac{1}{(1+x)^5} - \frac{5}{(1+x)} - \frac{10}{(1+x)^3} + \frac{5}{(1+x)^4} + \frac{10}{(1+x)^2}$.
  Note that the optimal and messy results differ by a constant, and the optimal form cannot be obtained by simplification of the messy.
- **Inconclusive:** No result was obtained in 60 seconds, or the result could not be verified, usually because the output was so large that the simplifier failed while attempting to differentiate and reduce to zero.
- **Invalid:** The difference between the derivative and integrand was not zero.

The performances on the test suite of Maple, Mathematica and the present rule-based system Rubi are presented in the tables below. Since Rubi

[1] Mathematica has been used to implement Rubi and a comparison with its built-in `Integrate` command shows Rubi to be faster on the test suite by a factor of 10. The test suite has been ported to Maple, but the different syntax for pattern matching has so far prevented Rubi from being ported. Therefore only the output forms can be compared.

was developed using the test suite, its good performance is to be expected, but even so, the favourable comparison with the other systems remains valid.

Although the test suite of 7927 problems is large, the problems themselves are all part of mainstream calculus, and therefore even the small rate of 3% invalid results for the large commercial systems is disappointing, to say the least. However, for the purposes of this paper, the emphasis of the main benefit of Rubi in this comparison lies in the simpler form of the results. Since the main advantage lies in the simplicity of the results, we concentrate here on how Rubi achieves its results, by presenting two case studies.

4 First Case Study: Alternative Strategies

The first study concerns an optimization to reduce output size. In order to obtain quantitative measures for expression size, utility functions have been written in Mathematica and Maple that count the number of nodes in the internal tree representation of a particular expression. Although there are variations in the internal representations of expressions, the functions provide comparable measures in the two systems.

We consider the problem of evaluating the integral

$$\int \frac{x^m \,\mathrm{d}x}{(a+bx)^{12}} \;, \tag{1}$$

for different values of $m \in \mathbb{Z}$. This is a special case of the more general problem

$$I(a,b,c,d,m,n) = \int (a+bx)^m (c+dx)^n \,\mathrm{d}x \;, \tag{2}$$

where $m, n \in \mathbb{Z}$, and $a, b, c, d \in \mathbb{C}$.

Our aim is to minimize the number of terms in the expression for the integral. As a starting point, we can use the standard integrators in Mathematica and Maple to evaluate the integral, and plot the expression sizes of the results as functions of m. Figures 1 and 2 show the expression counts for the two systems.

We have extracted below from Rubi the 9 transformation rules applying to the integral class (2). Each rule is presented in the following form:

N: Necessary conditions for mathematical validity.
T: The transformation rule $A \rightarrow B$.
S: Simplification conditions to ensure the transformation yields a simplification.

All rules require that a, b, c, d, m, n do not contain x, and that $b \neq 0$. The rules are:

1. **T**: $\displaystyle\int \frac{dx}{a+bx} \rightarrow \frac{\ln(a+bx)}{b}$.
2. **N**: $m+1 \neq 0$
   T: $\displaystyle\int (a+bx)^m dx \rightarrow \frac{(a+bx)^{m+1}}{(m+1)b}$.

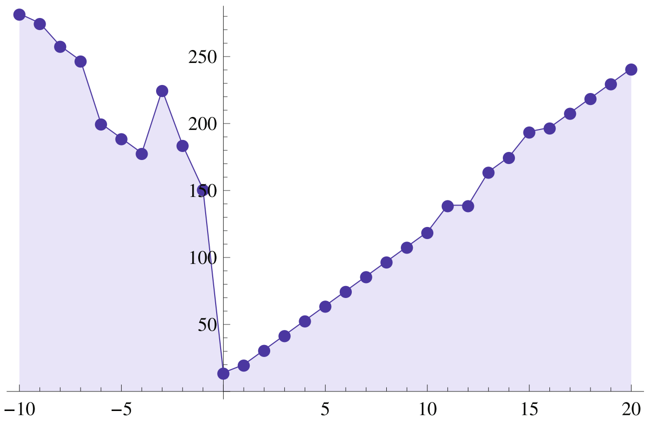


Fig. 1. The node count for expressions returned by MATHEMATICA 7 for the integral in (1). The horizontal axis shows values of the exponent m, while the vertical axis shows the node count for the corresponding expression for the integral.

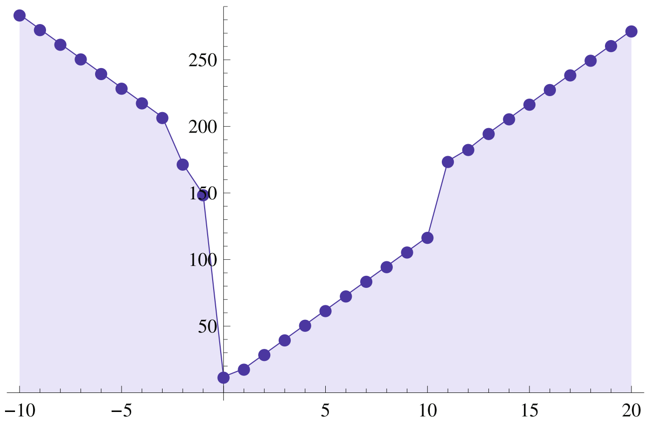


Fig. 2. The node count for expressions returned by MAPLE 13 for the integral in (1). The horizontal axis shows values of the exponent m, while the vertical axis shows the node count for the corresponding expression for the integral.

3. **N**: $bc - ad = 0$, $m + n + 1 = 0$
   T: $\int (a+bx)^m (c+dx)^n \,\mathrm{d}x \to (a+bx)^{m+1}(c+dx)^n \ln(a+bx)/b$.
4. **N**: $bc - ad = 0$, $m + n + 1 \neq 0$
   T: $\int (a+bx)^m (c+dx)^n \,\mathrm{d}x \to \dfrac{(a+bx)^{m+1}(c+dx)^n}{b(m+n+1)}$.
5. **N**: $bc - ad \neq 0$
   T: $\int (a+bx)^{-1} (c+dx)^{-1} \,\mathrm{d}x \to \dfrac{\ln(a+bx) - \ln(c+dx)}{bc-ad}$.

6. **N**: $bc - ad \neq 0$, $m + n + 2 = 0$, $n \neq -1$
   T: $$\int (a+bx)^m (c+dx)^n \, \mathrm{d}x \to -\frac{(a+bx)^{m+1}(c+dx)^{n+1}}{(n+1)(bc-ad)} \; .$$
7. **N**: $m + n + 1 = 0$, $m > 0$, $bc - ad \neq 0$
   T: $$\int (a+bx)^m (c+dx)^n \, \mathrm{d}x \to -\frac{(a+bx)^m}{dm(c+dx)^m} + \frac{b}{d}\int (a+bx)^{m-1}(c+dx)^{-m} \, \mathrm{d}x \; .$$
8. **N**: $bc - ad \neq 0$, $m + n + 1 \neq 0$, $n > 0$
   T: $$\int (a+bx)^m (c+dx)^n \, \mathrm{d}x \to \frac{(a+bx)^{m+1}(c+dx)^n}{b(m+n+1)} + \frac{n(bc-ad)}{b(m+n+1)}\int (a+bx)^m(c+dx)^{n-1} \, \mathrm{d}x \; .$$
   S: $(2n + m + 1 < 0 \vee m + n + 1 > 0) \wedge (m < 0 \vee n \leq m)$
9. **N**: $bc - ad \neq 0$, $n + 1 \neq 0$
   T: $$\int (a+bx)^m (c+dx)^n \, \mathrm{d}x \to -\frac{(a+bx)^{m+1}(c+dx)^{n+1}}{(n+1)(bc-ad)} + \frac{(m+n+2)b}{(bc-ad)(n+1)}\int (a+bx)^m(c+dx)^{n+1} \, \mathrm{d}x \; .$$
   S: $n < -1$, $m < 0 \vee 2m + n + 1 \geq 0$.

We wish to show how these rules are optimized relative to other possible sets of rules. Specifically, we shall compare these rules with a set in which the simplification conditions in rules 8 and 9 are modified. We start, however, with remarks on the rules as presented.

4.1 Remarks

1. An alternative strategy to the set of transformations shown here would be to define rules for the simpler integrand $x^m(a+bx)^n$, and then use the linear substitution $u = c+dx$ to transform expressions of the form $(a+bx)^m(c+dx)^n$ into the simpler form. This strategy was explored, but we discovered that several more rules are required when starting from the non-symmetrical form $x^m(a+bx)^n$ than when starting with the symmetrical $(a+bx)^m(c+dx)^n$. This is because two versions each of rules 7, 8 and 9 had to be given depending upon whether the exponent of the monomial or the linear factor had to be incremented or decremented. This subtle, but important, point shows that sometimes defining more general rules leads to a simpler repository.
2. It should be noted that rule 6 is in fact a special case of rule 9. It is included because it is convenient to have an explicitly non-recursive entry.
3. Rules 8 and 9 respectively increment and decrement one of the exponents of the integrand. Unlike the other rules, it is not always obvious which of these two rules should be applied to a given integrand in order to minimize the number of steps required to integrate it. This choice is the subject of our optimization.

5 Integration Strategies

The rules stated above describe a complete strategy for integration of the given class of integrals. The strategy is not unique, however, and other strategies might be more efficient. We therefore describe two other strategies and compare them with the preferred strategy.

5.1 Preliminary Strategy 1

We replace rule 8 with a rule 8a, in which the simplification conditions are removed. Thus we have

8a. **N**: $bc - ad \neq 0$, $m+n+1 \neq 0$, $n > 0$

$$\mathbf{T}\colon \int (a+bx)^m (c+dx)^n \,\mathrm{d}x \to \frac{(a+bx)^{m+1}(c+dx)^n}{b(m+n+1)} + \frac{n(bc-ad)}{b(m+n+1)} \int (a+bx)^m (c+dx)^{n-1}\,\mathrm{d}x \ .$$

The effect of removing the restrictions is that all integrals will be reduced until one of the exponents becomes zero, at which point rules 1 to 6 will terminate the reduction. When this strategy is applied to the test case (1), the sizes of the results are as shown in figure 3.

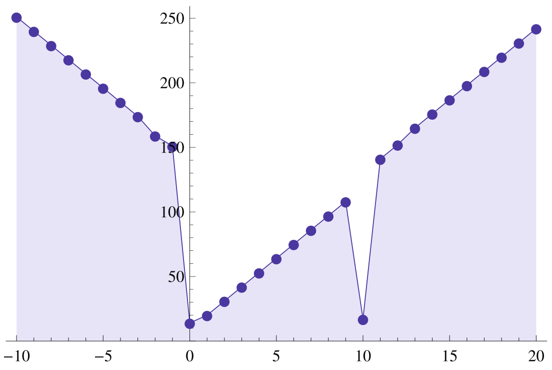


Fig. 3. The node count for expressions returned by the first alternative integration strategy for the integral in (1). The horizontal axis shows values of the exponent m, while the vertical axis shows the node count for the corresponding expression for the integral.

The dip for the case $m = 10$ is important. For this case, rule 6 provides a direct one-step integration to a very compact form:

$$\int \frac{x^{10}\,\mathrm{d}x}{(1+x)^{12}} = \frac{x^{11}}{11(1+x)^{11}} \ .$$

This possibility is not noticed by the standard integrators of Mathematica and Maple, as can be seen in figures 1 and 2.

5.2 Preliminary Strategy 2

We now remove the restrictions from rule 9, and place it above rule 8. Thus the rule becomes

9a **N**: $bc - ad \neq 0$, $n + 1 \neq 0$

T: $$\int (a+bx)^m (c+dx)^n \, \mathrm{d}x \to -\frac{(a+bx)^{m+1}(c+dx)^{n+1}}{(n+1)(bc-ad)} + \frac{(m+n+2)b}{(bc-ad)(n+1)} \int (a+bx)^m (c+dx)^{n+1} \, \mathrm{d}x$$

The effect of this is to increase one negative exponent until rule 6 can be applied. The resulting statistics on the size of integral expressions is shown in figure 4.

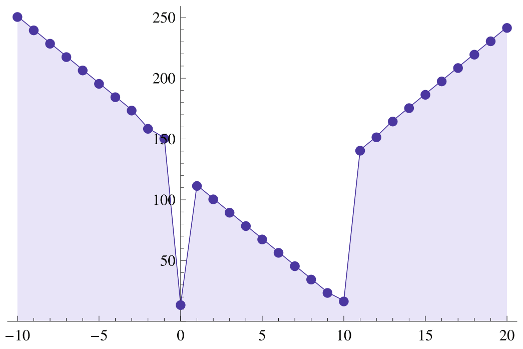


Fig. 4. The node count for expressions returned by the second alternative integration strategy for the integral in (1). The horizontal axis shows values of the exponent m, while the vertical axis shows the node count for the corresponding expression for the integral.

The dip at $m = 0$ is a result of rule 2 being applied before the general rules.

5.3 An Optimal Strategy

Clearly, one can obtain smaller expression sizes if one can switch between the two strategies just tested. This is what is done in rules 8 and 9 as presented. For the test case, the two points $m = 10$ and $m = 0$ are targets and for $m \leq 5$ the integrands are moved towards $m = 0$, while for $m > 5$ they are moved towards $m = 10$. The generalization to other powers is shown in rules 8 and 9. The resulting expression sizes are shown in figure 5.

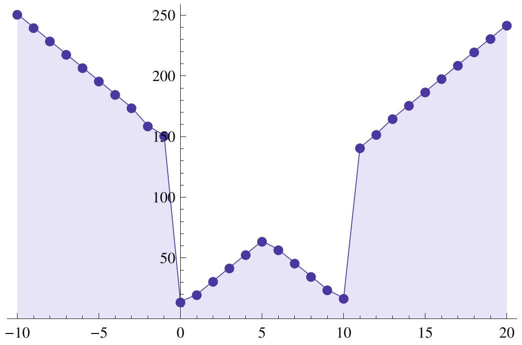


Fig. 5. The node count for expressions returned by the optimal integration strategy for the integral in (1). The horizontal axis shows values of the exponent m, while the vertical axis shows the node count for the corresponding expression for the integral.

5.4 Comparison with Other Methods

An obvious algorithmic approach to integral (1) is to expand the fraction using partial fractions, and then integrate each term. This gives results similar to those found using Maple and Mathematica. One of the advantages of Rubi is that such special cases can be identified and taken advantage of. One of the useful services that computer-algebra systems can offer mathematicians is the identification of special cases. The general algorithms preferred by Maple and Mathematica can succeed on large problems which Rubi is not yet capable of tackling. However, for smaller problems, where special cases might exist, Rubi is to be preferred.

6 Second Case Study: Two-Part Reduction

The second case study involves the class of integrals

$$J(m,n,p) = \int x^m (a+bx)^n (c+dx)^p \,\mathrm{d}x \ , \tag{3}$$

where the dependence of J on a, b, c, d has been suppressed, since we shall focus on the powers. We require $m \in \mathbb{Z}$, and $n, p \in \mathbb{Q}$. The aim is to reduce the integral in (3) to integrals with known solutions. In this case, the problems with known solutions are $J(0, N, P)$ and $J(M, N, N)$. It is straightforward to derive the equality

$$J(m,n,p) = (1/b)J(m-1,n+1,p) - (a/b)J(m-1,n,p) \tag{4}$$

An obvious strategy for $m > 0$ is to use this relation to reduce all integrals to the form $J(0, N, p)$. Thus, using the above conventions for describing a reduction rule, the rule reads

11. **T**: $J(m,n,p) \to \frac{1}{b}J(m-1,n+1,p) - (a/b)J(m-1,n,p)$
    S: $m \in \mathbb{Z}$, $m > 0$, $n, p \in \mathbb{Q}$, $n - p < 0$

At this point the algorithmically oriented person jumps to a composite rule by applying (4) m times to obtain

$$J(m,n,p) = \frac{1}{b^m} \sum_{k=0}^{m} \binom{m}{k} (-a)^k J(0, n+k, p) \ . \tag{5}$$

This, however, falls again into the trap that awaits grand algorithmic, or general-formula based, approaches. There are many special-case simplifications that will be skipped over by (5). Because the formula is derived for generic n, p, it can have no special behaviour for special cases. For example, if there exists k such that $n+k=p$ and $k<m$, then some terms can be removed from the sum and simplified separately, using the special case $J(m-k, n+k, p) = J(M, p, p)$. One of the differences between different computer systems is the extent to which they attempt intermediate simplifications. Using a step-based series of transformations (as Rubi does) each intermediate result can be tested for simplification before continuing.

For the case $m<0$, we rewrite (4) as

$$J(m,n,p) = (1/a)J(m,n+1,p) - (b/a)J(m+1,n,p) \tag{6}$$

Applying this reduction k times, we would obtain

$$J(m,n,p) = \frac{1}{a^k} \sum_{i=0}^{k} \binom{k}{i} (-b)^i J(m+i, n+k-i, p) \tag{7}$$

The terms in the sum can be evaluated whenever $m+i=0$ or $n+k-i=p$. Clearly, the latter condition requires that initially $n-p \in \mathbb{N}$. Therefore, the integral $J(m,n,p)$ will be evaluated after at most $\max(n-p, m)$ steps. As in the $m>0$ case, however, it is better to apply the reduction stepwise in order to obtain the maximum benefit from intermediate, special case, simplifications.

As an example of the above rules, we present the same integral calculated by Rubi, by Mathematica and by Maple. First, Rubi:

$$\int \frac{\sqrt{2+3x}\,\mathrm{d}x}{x^2(5-x)^{3/2}} = \frac{2\sqrt{2+3x}}{25\sqrt{5-x}} - \frac{\sqrt{5-x}\sqrt{2+3x}}{25x} - \frac{21}{25\sqrt{10}} \operatorname{arctanh} \frac{\sqrt{10-2x}}{\sqrt{10+15x}}$$

Next, Mathematica:

$$\begin{aligned} &= \frac{1}{500} \left(\frac{20(-5+3x)\sqrt{2+3x}}{x\sqrt{5-x}} + 21\sqrt{10} \ln\left(21\sqrt{10}x\right) \right. \\ &\quad \left. -21\sqrt{10} \ln\left(50\left(20+13x+2\sqrt{10}\sqrt{5-x}\sqrt{2+3x}\right)\right)\right) \end{aligned}$$

Table 1. The integration test suite, with the numbers of problems broken down in categories. The performance of the Rule-based Integrator (Rubi) is given using measures described in the text.

Test Items		Rubi: Rule-based Integrator			
Integrand	**Number**	**Optimal**	**Messy**	**Inconc.**	**Invalid**
Rational	1426	1424	1	1	0
Algebraic	1494	1483	8	3	0
Exponential	456	452	0	4	0
Logarithmic	669	667	0	2	0
Trigonometric	1805	1794	8	3	0
Hyperbolic	1386	1379	6	1	0
Inverse trig	283	281	0	2	0
Inverse hyperbolic	342	335	2	5	0
Special functions	66	66	0	0	0
Percentages		99.4%	0.3%	0.3%	0%

Table 2. The performance of Maple on the test suite, using measures described in the text

Test Items		**Maple**			
Integrand	**Number**	**Optimal**	**Messy**	**Inconc.**	**Invalid**
Rational	1426	1176	249	0	1
Algebraic	1494	1126	277	45	46
Exponential	456	351	63	37	5
Logarithmic	669	284	161	194	30
Trigonometric	1805	1054	619	83	49
Hyperbolic	1386	521	641	181	43
Inverse trig	283	206	64	5	8
Inverse hyperbolic	342	159	96	55	32
Special functions	66	38	1	25	2
Percentages		62.0%	27.4%	7.9%	2.7%

Finally, Maple:

$$= -\frac{1}{500}\left(21\sqrt{10}\,\mathrm{arctanh}\left(1/20\frac{(20+13x)\sqrt{10}}{\sqrt{10+13x-3x^2}}\right)x^2\right.$$
$$-105\sqrt{10}\mathrm{arctanh}\left(1/20\frac{(20+13x)\,\sqrt{10}}{\sqrt{10+13x-3\,x^2}}\right)x+60x\sqrt{10+13\,x-3x^2}$$
$$\left.-100\sqrt{10+13x-3x^2}\right)\sqrt{5-x}\sqrt{2+3x}\,(-5+x)^{-1}\,\frac{1}{\sqrt{10+13x-3x^2}}x^{-1}$$

There is a disadvantage, however, to stepwise application of the above reduction, a disadvantage well known in other contexts. This is the repeated evaluation of the same integral during recursive calls. The standard example of this effect is the recursive evaluation of Fibonacci numbers. This is paralleled in applications

Table 3. The performance of Mathematica on the test suite, using measures described in the text

Test Items		Mathematica			
Integrand	**Number**	**Optimal**	**Messy**	**Inconc.**	**Invalid**
Rational	1426	1239	187	0	0
Algebraic	1494	1228	246	18	2
Exponential	456	406	32	12	6
Logarithmic	669	581	84	4	0
Trigonometric	1805	1212	573	3	17
Hyperbolic	1386	911	464	6	5
Inverse trig	283	211	62	10	0
Inverse hyperbolic	342	198	140	3	1
Special functions	66	53	9	4	0
Percentages		76.2%	22.7%	0.8%	0.4%

of (4) and (6). This effect was one reason that MAPLE introduced its `option remember` early in its development. The important additional feature present here, that is not present in the Fibonacci example, is the possibility of different simplification options directing the computation to simpler results.

7 Concluding Remarks

In [5], a number of advantages of rule-based simplification were listed. These included (see reference for details).

- Human and machine readable.
- Able to show simplification steps.
- Facilitates program development.
- Platform independent.
- White box transparency.
- Fosters community development.
- An active repository.

In this paper we have shown that an additional advantage of rule-based evaluation, illustrated in the integration context, is greater simplicity of results. Finally, we wish to point out that the integration repository described here has been published on the web [6], and is available for viewing and testing by all interested people.

References

1. Abramowitz, M., Stegun, I.: Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. US Government Printing Office (1964) (10th Printing December 1972)
2. Fateman, R.J.: A Review of Mathematica. J. Symb. Computation 13(5) (1992)

3. Einwohner, T.H., Fateman, R.J.: Searching techniques for integral tables. In: Proceedings ISSAC 1995, pp. 133–139. ACM Press, New York (1995)
4. Gradshteyn, I.S., Ryzhik, I.M.: Table of Integrals, Series and Products. Academic Press, London (1965)
5. Rich, A.D., Jeffrey, D.J.: A knowledge repository for indefinite integration based on transformation rules. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) Calculemus 2009, MKM 2009. LNCS, vol. 5625, pp. 480–485. Springer, Heidelberg (2009)
6. Rich, A.D.: Indefinite Integration Rules, `http://www.apmaths.uwo.ca/RuleBasedMathematics`

A Unified Formal Description of Arithmetic and Set Theoretical Data Types

Paul Tarau

Department of Computer Science and Engineering
University of North Texas
tarau@cs.unt.edu

Abstract. We provide a "shared axiomatization" of natural numbers and hereditarily finite sets built around a polymorphic abstraction of bijective base-2 arithmetics.

The "axiomatization" is described as a progressive refinement of Haskell type classes with examples of instances converging to an efficient implementation in terms of arbitrary length integers and bit operations. As an instance, we derive algorithms to perform arithmetic operations efficiently directly with hereditarily finite sets.

The self-contained source code of the paper is available at http://logic.cse.unt.edu/tarau/research/2010/unified.hs

Keywords: formal description of arithmetic and set theoretical data types, Peano arithmetic and hereditarily finite sets, bijective base-2 arithmetic, software refinement with Haskell type classes, computational mathematics.

1 Introduction

Natural numbers and finite sets have been used as sometimes competing foundations for mathematics, logic and consequently computer science. The de facto standard axiomatization for natural numbers is provided Peano arithmetic. Finite set theory is axiomatized with the usual Zermelo-Fraenkel system (abbreviated ZF) in which the Axiom of Infinity is replaced by its negation. When the axiom of ϵ-induction, (saying that if properties proven on elements also hold on sets containing them, then they hold for all finite sets) is added, the resulting finite set theory (abbreviated ZF^*) is *bi-interpretable* with Peano arithmetic i.e. they emulate each other accurately through a bijective mapping that commutes with standard operations on the two sides ([1]).

This foundational convergence suggests a "shared axiomatization" of Peano arithmetic, hereditarily finite sets and more conventional natural number representations to be used as a unified framework for formally deriving various computational entities.

While axiomatizations of various formal systems are traditionally expressed in classic or intuitionistic predicate logic, equivalent formalisms, in particular the λ-calculus and the type theory used in modern functional languages like Haskell, can provide specifications in a sometime more readable, more concise, and more importantly, in a genuinely *executable* form.

Our incremental specification loop consists of successive refinements through a chain of Haskell *type classes* (seen as axiom systems) connected by inheritance.

Instances of the type classes (seen as *interpretations* of axiom systems) provide examples that implement various data types in this framework.

The resulting hierarchy of type classes describes incrementally *common computational capabilities* shared by bit-stacks, Peano natural numbers and hereditarily finite sets (sections 3-5).

2 Computing in Bijective Base-2

Bitstrings provide a common and efficient computational representation for both sets and natural numbers. This recommends their operations as the right abstraction for deriving, in the form of a Haskell type class, a "shared axiomatization" for Peano arithmetic and Finite Set Theory.

While the existence of such a common axiomatization can be seen as a consequence of the bi-interpretability results proven in [1], our distinct executable specification as a Haskell type class provides unique insights into the shared inductive constructions and ensures that computational complexity of operations is kept under control for a variety of instances.

We start by expressing bitstring operations as a Haskell data type:

```
data BitStack = Empty|Bit0 BitStack|Bit1 BitStack
  deriving (Eq, Show, Read)
```

We define the following operations on BitStacks

```
empty = Empty

pushBit0 xs = Bit0 xs
pushBit1 xs = Bit1 xs

popBit (Bit0 xs)=xs
popBit (Bit1 xs)=xs
```

and the predicates

```
empty_ x=Empty==x
bit0_ (Bit0 _)=True
bit0_ _ =False

bit1_ (Bit1 _)=True
bit1_ _=False
```

We remind a few basic (but possibly not widely known) concepts related to the computation mechanism we will use on bitstrings[1].

[1] We assume that bitstrings are mapped to numbers starting with the lowest exponent of 2 and ending with the highest.

Definition 1. *Bijective base-2 representation associates to $n \in \mathcal{N}$ a unique string in the regular language $\{0,1\}^*$ by removing the 1 indicating the highest exponent of 2 from the standard (complement of 2) bitstring representation of $n+1$.*

Using a list notation for bitstrings this gives: $0 = [], 1 = [0], 2 = [1], 3 = [0,0], 4 = [1,0], 5 = [0,1], 6 = [1,1]$ etc[2].

As a simple exercise in bijective base-2, arithmetic one can now implement the successor function - and therefore provide a model of Peano's axioms, as follows:

```
zero = empty
one = Bit0 empty

peanoSucc xs | empty_ xs = one
peanoSucc xs | bit0_ xs = pushBit1 (popBit xs)
peanoSucc xs | bit1_ xs = pushBit0 (peanoSucc (popBit xs))
```

For instance, 3 applications of `peanoSucc` generate $3 = [0,0]$ as follows:

```
*Unified> (peanoSucc . peanoSucc . peanoSucc) zero
Bit0 (Bit0 Empty)
```

One can verify by structural induction that:

Proposition 1. *Peano's axioms hold with the definition of the successor function provided by* `peanoSucc`.

Using the BitStack representation (by contrast with naive "base-1" successor based definitions), one can implement arithmetic operations like sum and product with low polynomial complexity in terms of the bitsize of their operands. We will defer defining these operations until the next sections, where we will provide such implementations in a more general setting.

Note that as a mild lookahead step towards abstracting away operations on our bitstacks, we have replaced reference to data constructors by the corresponding predicates and functions i.e. `bit0_` `bit1_` etc.

3 Sharing Axiomatizations with *Type Classes*

Haskell's *type classes* [2] are a good approximation of axiom systems as they allow one to describe properties and operations generically i.e. in terms of their action on objects of a parametric type. Haskell's *instances* approximate *interpretations* [1] of such axiomatizations by providing implementations of primitive operations and by refining and possibly overriding derived operations with more efficient equivalents.

We will start by defining a type class that abstracts away the operations on the `BitStack` datatype and provides an axiomatization of natural numbers first, and hereditarily finite sets later.

[2] See `http://en.wikipedia.org/wiki/Bijective_numeration` for the historical origins of the concept and the more general *bijective base-k* case.

3.1 The 5 Primitive Operations

The class `Polymath` assumes only a theory of structural equality (as implemented by the class `Eq` in Haskell) and the `Read/Show` superclasses needed for input/output.

An instance of this class is required to implement the following 5 primitive operations:

```
class (Eq n,Read n,Show n)⇒Polymath n where
  e :: n
  o_ :: n→Bool
  o :: n→n
  i :: n→n
  r :: n→n
```

We have chosen single letter names `e,o_,o,i,r` for the abstract operations corresponding respectively to `empty, bit0_, pushBit0, pushBit1, popBit` to facilitate a concise "algebraic" view needed to grasp some complex definitions that use compositions of these operations[3].

The `Polymath` type class also provides to its instances generic implementations of the following derived operations:

```
  e_ :: n→Bool
  e_ x = x==e

  i_ :: n→Bool
  i_ x = not (o_ x || e_ x)
```

Note that we use the convention that for each constructor the recognizer's name is obtained by appending "_"[4].

While not strictly needed at this point, it is convenient also to include in the `Polymath` type class some additional derived operations. As we will see later, some instances will chose to override them. We first define an object and a recognizer for `1`, the constant function `u` and the predicate `u_`.

```
  u :: n
  u = o e

  u_ :: n→Bool
  u_ x = o_ x && e_ (r x)
```

Next we implement the successor `s` and predecessor `p` functions:

```
  s :: n→n
  s x | e_ x = u
  s x | o_ x = i (r x)
  s x | i_ x = o (s (r x))
```

[3] As an ongoing analogy, the reader can interpret `o` as pushing a `0` to a bitstack, `i` as pushing a `1` and `r` as a pop operation, with `e` representing an empty bitstack.

[4] As part of the bitstack analogy, the predicates `o_` and `i_` can be seen as recognizing respectively a `0` and a `1` (in bijective base-2) at the top of the bitstack.

```
  p :: n→n
  p x | u_ x = e
  p x | o_ x = i (p (r x))
  p x | i_ x = o (r x)
```

It is convenient at this point, as we target a diversity of interpretations materialized as Haskell instances, to provide a polymorphic converter between two different instances of the type class `Polymath`. The function `view` allows converting between two different Polymath instances, generically.

```
view :: (Polymath a,Polymath b)⇒a→b
view x | e_ x = e
view x | o_ x = o (view (r x))
view x | i_ x = i (view (r x))
```

3.2 Peano Arithmetic

It is important to observe at this point that Peano arithmetic is an instance of the class `Polymath` i.e. that the class can be used to derive an "axiomatization" for Peano arithmetic through a straightforward mapping of Haskell's function definitions to Peano's axioms.

```
data Peano = Zero|Succ Peano deriving (Eq,Show,Read)

instance Polymath Peano where
  e = Zero

  o_ Zero = False
  o_ (Succ x) = not (o_ x)

  o x = Succ (o' x) where
    o' Zero = Zero
    o' (Succ x) = Succ (Succ (o' x))

  i x = Succ (o x)

  r (Succ Zero) = Zero
  r (Succ (Succ Zero)) = Zero
  r (Succ (Succ x)) = Succ (r x)
```

Finally, we can add `BitStack` - which, after all, has inspired the operations of our type class, as an instance of `Polymath`

```
instance Polymath BitStack where
  e=empty
  o=pushBit0
  o_=bit0_
  i=pushBit1
  r=popBit
```

and observe that the Peano and Bitstack interpretations behave consistently:

```
*Unified> i (o (o Empty))
Bit1 (Bit0 (Bit0 Empty))
*Unified> i (o (o Zero))
Succ (Succ (Succ (Succ (Succ (Succ (Succ (Succ Zero)))))))
*Unified> i (o (o Empty))
Bit1 (Bit0 (Bit0 Empty))
*Unified> s it
Bit0 (Bit1 (Bit0 Empty))
*Unified> view it :: Peano
Succ (Succ (Succ (Succ (Succ (Succ (Succ (Succ (Succ Zero))))))))
*Unified> p it
Succ (Succ (Succ (Succ (Succ (Succ (Succ (Succ Zero)))))))
Bit1 (Bit0 (Bit0 Empty))
```

Note also the convenience of using `:: view` to instantly morph between instances and the use of Haskell's `it` standing for the previously returned result. So far we have seen that our instances implement syntactic variations of natural numbers equivalent to Peano's axioms. We will now provide an instance showing that our "axiomatization" covers the theory of hereditarily finite sets (assuming, of course, that extensionality, comprehension, regularity, ϵ-induction etc. are implicitly provided by type classes like `Eq` and implementation of recursion in the underlying programming language).

4 Computing with *Hereditarily Finite Sets*

Hereditarily finite sets are built inductively from the empty set (denoted `S []`) by adding finite unions of existing sets at each stage. We first define a rooted tree datatype `S`:

```
data S=S [S] deriving (Eq,Read,Show)
```

To accurately represent sets, the type `S` would require a type system enforcing constraints on type parameters, saying that all elements covered by the definition are distinct and no repetitions occur in any list of type `[S]`. We will assume this and similar properties of our datatypes, when needed, from now on, and consider trees built with the constructor `S` as representing hereditarily finite sets.

We will now show that hereditarily finite sets can do arithmetic as instances of the class `Polymath` by implementing a successor (and predecessor) function. We start with the easier operations:

```
instance Polymath S where
  e = S []

  o_ (S (S []:_)) = True
  o_ _ = False

  o (S xs) = s (S (map s xs))

  i = s . o
```

Note that the `o` operation, that can be seen as pushing a `0` bit to a bitstack is implemented by applying `s` to each branch of the tree. We will now implement `r`, `s` and `p`.

```
r (S xs) = S (map p (f ys)) where
  S ys = p (S xs)
  f (x:xs) | e_ x = xs
  f xs = xs

s (S xs) = S (hLift (S []) xs) where
  hLift k [] = [k]
  hLift k (x:xs) | k==x = hLift (s x) xs
  hLift k xs = k:xs

p (S xs) = S (hUnLift xs) where
  hUnLift ((S []):xs) = xs
  hUnLift (k:xs) = hUnLift (k':k':xs) where k'= p k
```

First note that *successor* and *predecessor* operations `s,p` are overridden and that the `r` operation is expressed in terms of `p`, as `o` and `i` were expressed in terms of `s`. Next, note that the `map` combinators and the auxiliary functions `hLift` and `hUnlift` are used to delegate work between successive levels of the tree defining a hereditarily finite set.

To summarize, let us observe that the successor and predecessor operations `s,p` at a given level are implemented through iteration of the same at a lower level and that the “left shift” operation implemented by `o,i` results in initiating `s` operations at a lower level. Thus the total number of operations is within a constant factor of the size of the trees.

Let us verify that these operations mimic indeed their more common counterparts on type `Peano`.

```
*Unified> o (i (S []))
S [S [],S [S [S []]]]
*Unified> s it
S [S [S []],S [S [S []]]]
*Unified> view it :: Peano
Succ (Succ (Succ (Succ (Succ (Succ Zero)))))
*Unified> p it
Succ (Succ (Succ (Succ (Succ Zero))))
*Unified> view it :: S
S [S [],S [S [S []]]]
```

It can be proven by structural induction that:

Proposition 2. *Hereditarily finite sets as represented by the data type* `S` *implement the same successor and predecessor operation as the instance* `Peano`*.*

Note that this implementation of the class `Polymath` implicitly uses the *Ackermann interpretation* of Peano arithmetic in terms of the theory of hereditarily finite sets, i.e. the natural number associated to a hereditarily finite set is given by the function

$f(x) =$ `if` $x = \emptyset$ `then` 0 `else` $\sum_{a \in x} 2^{f(a)}$

Let us summarize what's unusual with instance `S` of the class `Polymath`: it shows that successor and predecessor operations can be performed with *hereditarily finite sets playing the role of natural numbers.* As natural numbers and finite ordinals are in a one-to-one mapping, this instance shows that hereditarily finite sets can be seen as *finite ordinals* directly, without using the simple but computationally explosive von Neumann construction (which defines ordinal n as the set $\{0, 1, \ldots, n-1\}$). We will elaborate more on this after defining a total order on our Polymath type.

5 Arithmetic Operations

Our next refinement adds key arithmetic operations in the form of a type class extending `Polymath`. We start with addition (`polyAdd`) and subtraction (`polySubtract`):

```
class (Polymath n) ⇒ PolyOrd n where
  polyAdd :: n→n→n
  polyAdd x y | e_ x = y
  polyAdd x y | e_ y = x
  polyAdd x y | o_ x && o_ y =    i (polyAdd (r x) (r y))
  polyAdd x y | o_ x && i_ y = o (s (polyAdd (r x) (r y)))
  polyAdd x y | i_ x && o_ y = o (s (polyAdd (r x) (r y)))
  polyAdd x y | i_ x && i_ y = i (s (polyAdd (r x) (r y)))

  polySubtract :: n→n→n
  polySubtract x y | e_ x && e_ y = e
  polySubtract x y | not(e_ x) && e_ y = x
  polySubtract x y | not (e_ x) && x==y = e
  polySubtract z x | i_ z && o_ x = o (polySubtract (r z) (r x))
  polySubtract z x | o_ z && o_ x = i (polySubtract (r z) (s (r x)))
  polySubtract z x | o_ z && i_ x = o (polySubtract (r z) (s (r x)))
  polySubtract z x | i_ z && i_ x = i (polySubtract (r z) (s (r x)))
```

Efficient comparison uses the fact that with our representation only sequences of distinct lengths can be different. We start by comparing lengths:

```
  lcmp :: n→n→Ordering

  lcmp x y | e_ x && e_ y = EQ
  lcmp x y | e_ x && not(e_ y) = LT
  lcmp x y | not(e_ x) && e_ y = GT
  lcmp x y = lcmp (r x) (r y)
```

Comparison can now proceed by case analysis, the interesting case being when lengths are equal (function `samelen_cmp`):

```
cmp :: n→n→Ordering
cmp x y = ecmp (lcmp x y) x y where
   ecmp EQ x y = samelen_cmp x y
   ecmp b _ _ = b

samelen_cmp :: n→n→Ordering

samelen_cmp x y | e_ x && e_ y = EQ
samelen_cmp x y | e_ x && not(e_ y) = LT
samelen_cmp x y | not(e_ x) && e_ y = GT
samelen_cmp x y | o_ x && o_ y = samelen_cmp (r x) (r y)
samelen_cmp x y | i_ x && i_ y = samelen_cmp (r x) (r y)
samelen_cmp x y | o_ x && i_ y =
  downeq (samelen_cmp (r x) (r y)) where
    downeq EQ = LT
    downeq b = b
samelen_cmp x y | i_ x && o_ y =
  upeq (samelen_cmp (r x) (r y)) where
    upeq EQ = GT
    upeq b = b
```

Finally, boolean comparison operators are defined as follows:

```
lt,gt,eq :: n→n→Bool

lt x y = LT==cmp x y

gt x y = GT==cmp x y

eq x y = EQ==cmp x y
```

After adding the instances

```
instance PolyOrd Peano
instance PolyOrd BitStack
instance PolyOrd S
```

one can see that all operations extend naturally:

```
*Unified> polyAdd (Succ Zero) (Succ Zero)
Succ (Succ Zero)
*Unified> (s.s.s.s) Empty
Bit1 (Bit0 Empty)
*Unified> take 1000 (iterate s (S []))
[S [],S [S []],....,S [S [],S [S [],S [S []]]]]]
*Unified> and (zipWith lt it (map s it))
True
```

The last example confirms, for 1000 instances, that we have a *well-ordering* of hereditarily finite sets without recurse to the von Neumann ordinal construction (used in [1] to complete the bi-interpretation from hereditarily finite sets to natural numbers). This replicates a recent result described in [3] where a lexicographic ordering is used to simplify the proof of bi-interpretability of [1].

We will proceed now with introducing more powerful operations. Needless to say, they will apply automatically to all instances of the type class `Polymath`.

6 Adding Other Arithmetic Operations

We first define multiplication.

```
class (PolyOrd n) ⇒ PolyCalc n where
  polyMultiply :: n→n→n
  polyMultiply x _ | e_ x = e
  polyMultiply _ y | e_ y = e
  polyMultiply x y = s (multiplyHelper (p x) (p y)) where
   multiplyHelper x y | e_ x = y
   multiplyHelper x y | o_ x = o (multiplyHelper (r x) y)
   multiplyHelper x y | i_ x = s (polyAdd y  (o (multiplyHelper (r x) y)))

  double :: n→n
  double = p . o

  half :: n→n
  half = r . s
```

Exponentiation by squaring follows - easier for powers of two (`exp2`), then the general case (`pow`):

```
  exp2 :: n→n -- power of 2
  exp2 x | e_ x = u
  exp2 x = double (exp2 (p x))

  pow :: n→n→n -- power y of x
  pow _ y | e_ y = u
  pow x y | o_ y = polyMultiply x (pow (polyMultiply x x) (r y))
  pow x y | i_ y = polyMultiply
    (polyMultiply x x)
    (pow (polyMultiply x x) (r y))
```

After defining instances

```
instance PolyCalc Peano
instance PolyCalc BitStack
instance PolyCalc S
```

operations can be tested under various representations

```
*Unified> polyMultiply (s (s (S []))) (s (s (s (S []))))
S [S [S []],S [S [S []]]]
*Unified> view it :: Peano
Succ (Succ (Succ (Succ (Succ (Succ Zero)))))
*Unified> pow (s (s (S []))) (s (s (s (S []))))
S [S [S [],S [S []]]]
*Unified> view it :: Peano
Succ (Succ (Succ (Succ (Succ (Succ (Succ (Succ Zero)))))))
```

7 Deriving Set Operations

We will now provide a set view of our polymorphic data type. Following [4], where Ackermann's mapping between hereditarily finite sets and natural numbers has been derived as a fold/unfold operation using a bijection between natural numbers and finite sets of natural numbers, we can write:

```
class (PolyCalc n) ⇒ PolySet n where
  as_set_nat :: n→[n]
  as_set_nat n = nat2exps n e where
    nat2exps n _ | e_ n = []
    nat2exps n x = if (i_ n) then xs else (x:xs) where
      xs=nat2exps (half n) (s x)

  as_nat_set :: [n]→n
  as_nat_set ns = foldr polyAdd e (map exp2 ns)
```

Given that natural numbers and hereditarily finite sets, when seen as instances of our generic axiomatization, are connected through Ackermann's bijections, one can shift from one side to the other at will:

```
*Unified> as_set_nat (s (s (s Zero)))
[Zero,Succ Zero]
*Unified> as_nat_set it
Succ (Succ (Succ Zero))
*Unified> as_set_nat (s (s (s (S []))))
[S [],S [S []]]
*Unified> as_nat_set it
S [S [],S [S []]]
```

Note also that, as the operations on type S show, the set associated to the number 3 is exactly the same as the first level of its expansion as a hereditarily finite set.

After defining combinators for operations of arity 1 and 2:

```
  setOp1 :: ([n]→[n])→(n→n)
  setOp1 f = as_nat_set . f . as_set_nat
  setOp2 :: ([n]→[n]→[n])→(n→n→n)
  setOp2 op x y = as_nat_set (op (as_set_nat x) (as_set_nat y))
```

we can "borrow" (with confidence!) the usual set operations (provided in the Haskell package Data.List):

```
  setIntersection :: n→n→n
  setIntersection = setOp2 intersect

  setUnion :: n→n→n
  setUnion = setOp2 union

  setDifference :: n→n→n
  setDifference = setOp2 (\\)
  setIncl :: n→n→Bool
  setIncl x y = x==setIntersection x y
```

In a similar way, we define a powerset operation conveniently using actual lists, before reflecting it into an operation on natural numbers.

```
  powset :: n→n
  powset x = as_nat_set
    (map as_nat_set (subsets (as_set_nat x))) where
      subsets [] = [[]]
      subsets (x:xs) = [zs|ys←subsets xs,zs←[ys,(x:ys)]]
```

Next, the ϵ-relation defining set membership is given as the function `inSet`, together with the `augmentSet` function used in various set theoretic constructs as a new set generator.

```
  inSet :: n→n→Bool
  inSet x y = setIncl (as_nat_set [x]) y

  augmentSet :: n→n
  augmentSet x = setUnion x (as_nat_set [x])
```

The n-th *von Neumann ordinal* is the set $\{0, 1, \ldots, n-1\}$ and it is used to emulate natural numbers in finite set theory. It is implemented by the function `nthOrdinal`:

```
  nthOrdinal :: n→n
  nthOrdinal x | e_ x = e
  nthOrdinal n = augmentSet (nthOrdinal (p n))
```

Note that as hereditarily finite sets and natural numbers are instances of the class `PolyOrd`, an order preserving bijection can be defined between the two, which makes it unnecessary to resort to von Neumann ordinals to show bi-interpretability [1,3].

After defining the appropriate instances

```
instance PolySet Peano
instance PolySet BitStack
instance PolySet S
```

we observe that set operations act naturally under the hereditarily finite set interpretation:

```
*Unified> (s.s.s.s.s.s)  (S [])
S [S [S []],S [S [S []]]]
*Unified> inSet (S [S []]) it
True

*Unified> powset (S [])
S [S []]
*Unified> powset it
S [S [],S [S []]]
```

```
*Unified> augmentSet (S [])
S [S []]
*Unified> augmentSet it
S [S [],S [S []]]
```

8 Deriving an Instance with Fast Bitstring Operations

We will now benefit from our shared axiomatization by designing an instance that takes advantage of bit operations, to implement, through a few overrides, fast versions of our arithmetic and set functions. For syntactic convenience, we will map this instance directly to Haskell's arbitrary length Integer type, to benefit in GHC from the performance of the underlying C-based GMP package. First some arithmetic operations (making use of Haskell's `Data.Bits` library):

```
instance Polymath Integer where
  e = 0
  o_ x = testBit x 0

  o x = succ (shiftL x 1)
  i  = succ . o
  r x | x>0 = shiftR (pred x) 1

  s = succ
  p n | n>0 = pred n
  u = 1
  u_ = (== 1)

instance PolyOrd Integer where
  polySubtract x y = abs (x-y)
  lt = (<)
  polyCompare=compare

instance PolyCalc Integer where
  polyMultiply = (*)
  half x = shiftR x 1
  double x = shiftL x 1
```

Next, some set operations:

```
instance PolySet Integer where
  setUnion = (.|.)
  setIntersection = (.&.)
  setDifference x y = x .&. (complement y)

  inSet x xs = testBit xs (fromIntegral x)

  powset 0 = 1
  powset x = xorL (powset (pred x)) where
    xorL n = n `xor` (shiftL n 1)
```

It is tempting to test for correctness, by computing with the "implementation" provided by the type `Integer` and then reverting to the set view:

```
*Unified> as_nat_set [1,3,4]
26
*Unified> powset it
84215045
*Unified> map as_set_nat (as_set_nat it)
[[],[1],[3],[1,3],[4],[1,4],[3,4],[1,3,4]]
```

It all adds up, but as we do not have a proof yet, we leave it as an *open problem* to show that *the* `xor` *based instance of* `powset` *in* `Integer` *does indeed implement the powerset operation as specified in section* 7.

Finally, we can observe that the von Neumann ordinal construction (used to introduce natural numbers in set theory) defines a fast growing injective function from $\mathcal{N} \rightarrow \mathcal{N}$:

```
*Unified> map nthOrdinal [0..4]
[0,1,3,11,2059]
*Unified> as_set_nat 2059
[0,1,3,11]
```

In contrast, our "shared axiomatization" defines ordinals through a trivial *bijection*: the identity function.

Note, as a more practical outcome, that one can now use arbitrary length integers as an efficient representation of hereditarily finite sets. Conversely, a computation like

```
*Unified> s (S [S [S [S [S [S [S [S [S [S []]]]]]]]]])
S [S [],S [S [S [S [S [S [S [S [S []]]]]]]]]]
```

computing easily the successor of a tower of exponents of 2, in terms of hereditarily finite sets, would overflow any computer's memory when using a conventional integer representation.

9 Related Work

The techniques described in this paper originate in the data transformation framework described in [5,4,6]. The main new contribution is that while our previous work can be seen as "an existence proof" that, for instance, arithmetic computations can be performed with symbolic objects like hereditarily finite sets, here we show it constructively. Moreover, we lift our conceptual framework to a polymorphic axiomatization which turns out to have as *interpretations* (instances in Haskell parlance) natural numbers, bitstacks and hereditarily finite sets.

Natural number encodings of hereditarily finite sets have triggered the interest of researchers in fields like Axiomatic Set Theory and Foundations of Logic [1,7]. A number of papers of J. Vuillemin develop similar techniques aiming to unify various data types, with focus on theories of boolean functions and arithmetic [8]. Binary number-based axiomatizations of natural number arithmetic are likely to be folklore, but having access to the the underlying theory of the calculus of constructions [9] and the inductive proofs of their equivalence with Peano arithmetic in the libraries of the `Coq` [10] proof assistant has been particularly enlightening to the author. On the other hand we have not found in the literature any axiomatizations in terms of hereditarily finite sets, as derived in this paper. Future work is planned in proving with Coq the equivalence of operations in Peano arithmetic with their counterparts in the set theoretic interpretation of our type classes.

10 Conclusion

In the form of a literate Haskell program, we have built "shared axiomatizations" of finite arithmetic and hereditarily finite sets using successive refinements of type classes.

We have derived some unusual algorithms, for instance, by expressing arithmetic computations symbolically, in terms of hereditarily finite sets. We have also provided a well-ordering for hereditarily finite sets that maps them to ordinals directly, without using the von Neumann construction.

This has been made possible by extending the techniques introduced in [5,4,6] that allow observing the internal working of intricate mathematical concepts through isomorphisms transporting operations between fundamental data types.

References

1. Kaye, R., Wong, T.L.: On Interpretations of Arithmetic and Set Theory. Notre Dame J. Formal Logic 48(4), 497–510 (2007)
2. Jones, S.P., Jones, M., Meijer, E.: Type classes: An exploration of the design space. In: Haskell Workshop (1997)
3. Pettigrew, R.: On Interpretations of Bounded Arithmetic and Bounded Set Theory. Notre Dame J. Formal Logic 50(2), 141–151 (2009)
4. Tarau, P.: A Groupoid of Isomorphic Data Transformations. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) Calculemus 2009, MKM 2009. LNCS (LNAI), vol. 5625, pp. 170–185. Springer, Heidelberg (2009)
5. Tarau, P.: Isomorphisms, Hylomorphisms and Hereditarily Finite Data Types in Haskell. In: Proceedings of ACM SAC 2009, Honolulu, Hawaii, pp. 1898–1903. ACM, New York (March 2009)
6. Tarau, P.: An Embedded Declarative Data Transformation Language. In: Proceedings of 11th International ACM SIGPLAN Symposium PPDP 2009, Coimbra, Portugal, pp. 171–182. ACM Press, New York (September 2009)
7. Kirby, L.: Addition and multiplication of sets. Math. Log. Q. 53(1), 52–65 (2007)
8. Vuillemin, J.: Digital algebra and circuits. In: Dershowitz, N. (ed.) Verification: Theory and Practice. LNCS, vol. 2772, pp. 733–746. Springer, Heidelberg (2004)
9. Coquand, T., Huet, G.: The calculus of constructions. Information and Computation 76(2/3), 95–120 (1988)
10. The Coq development team: The Coq proof assistant reference manual. LogiCal Project, Version 8.0 (2004)

Against Rigor

Doron Zeilberger

Department of Mathematics, Rutgers University, USA
http://www.math.rutgers.edu/~zeilberg/

Abstract. The ancient Greeks gave (western) civilization quite a few gifts, but we should beware of Greeks bearing gifts. The gifts of theatre and democracy were definitely good ones, and perhaps even the gift of philosophy, but the "gift" of the so-called "axiomatic method" and the notion of "rigorous" proof did much more harm than good. If we want to maximize Mathematical Knowledge, and its Management, we have to return to Euclid this dubious gift, and give-up our fanatical insistence on perfect rigor. Of course, we should not go to the other extreme, of demanding that everything should be non-rigorous. We should encourage diversity of proof-styles and rigor levels, and remember that nothing is absolutely sure in this world, and there does not exist an absolutely rigorous proof, nor absolute certainty, and "truth" has many shades and levels.

Smart Matching

Andrea Asperti and Enrico Tassi

[1] Department of Computer Science, University of Bologna
asperti@cs.unibo.it
[2] Microsoft Research-INRIA Joint Center
enrico.tassi@inria.fr

Abstract. One of the most annoying aspects in the formalization of mathematics is the need of transforming notions to match a given, existing result. This kind of transformations, often based on a conspicuous background knowledge in the given scientific domain (mostly expressed in the form of equalities or isomorphisms), are usually implicit in the mathematical discourse, and it would be highly desirable to obtain a similar behaviour in interactive provers. The paper describes the superposition-based implementation of this feature inside the Matita interactive theorem prover, focusing in particular on the so called *smart application* tactic, supporting smart matching between a goal and a given result.

1 Introduction

The mathematical language has a deep contextual nature, whose interpretation often presupposes not trivial skills in the given mathematical discipline. The most common and typical example of these "logical abuses" is the implicit use of equalities and isomorphisms, allowing a mathematician to freely move between different incarnations of a same entity in a completely implicit way. Equipping ITP systems with the capability of reasoning up to equality yields an essential improvement of their intelligence, making the communication between the user and the machine sensibly easier.

Techniques for equational reasoning have been broadly investigated in the realm of automated theorem proving (see eg [7,22,10]). The main deductive mechanism is a *completion* technique [17] attempting to transform a given set of equations into a confluent rewriting system so that two terms are equal if and only if they have identical normal forms. Not every equational theory can be presented as a confluent rewriting system, but one can progressively approximate it by means of a refutationally complete method called *ordered completion*. The deductive inference rule used in completion procedures is called *superposition*: it consists of first unifying one side of one equation with a subterm of another, and hence rewriting it with the other side. The selection of the two terms to be unified is guided by a suitable *term ordering*, constraining inferences and sensibly pruning the search space.

Although we are not aware of any work explicitly focused on superposition techniques for interactive provers, the integration between fully automatic provers (usually covering paramodulation) and interactive ones is a major research challenge and many efforts have been already done in this direction: for instance, KIV has been integrated with the tableau prover $3T^AP$ [1]; HOL has been integrated with various first order provers, such as Gandalf [15] and Metis; Coq has been integrated with Bliksem [8]; Isabelle was first integrated with a purpose-built prover [23] and more recently with Vampire [20]. The problems of these integrations are usually of two kinds: (a) there is a *technical* difficulty in the forward and backward translation of the information between systems, due to the different underlying logics (ITP systems are usually higher-order, and some of them intuitionistic); (b) there is a *pragmatical* problem in the management of the knowledge base to be used by the automatic solver, since it can be huge (so we cannot pass it at every invocation), and it grows dynamically (hence, it cannot be exported in advance).

A good point of the superposition calculus (and not the last reason for restricting the attention to this important fragment) is that point (a), in this context, becomes relatively trivial (and the translation particularly effective). As for point (b), its main consequence is that the communication between the Interactive Prover and the Problem Solver, in order to be efficient, cannot be *stateless*: the two systems must share a common knowledge base. This fact, joined with the freedom to adapt the superposition tool to any possible specific requirement of the Matita system convinced us to rewrite our own solver, instead of trying to interface Matita with some available tool. This paper discusses our experience of implementation of a (first order) superposition calculus (Section 2), its integration within the (higher-order) Matita interactive prover [5] (Section 3), and in particular its use for the implementation of a *smart application* tactic, supporting smart matching between a goal and a given results (Section 4). We shall conclude with a large number of examples of concrete use of this tactic.

2 The Matita Superposition Tool

One of the components of the automation support provided by the Matita interactive theorem prover is a first order, untyped superposition tool. This is a quite small and compact application (little more than 3000 lines of OCaml code), well separated by the rest of the system. It was entirely rewritten during the summer 2009 starting from a previous prototype (some of whose functionalities had been outlined in [6]), with the aim to improve both its abstraction and performance. The tool took part to the 22nd CADE ATP System Competition, in the unit equality division, scoring in fourth position, beating glorious systems such as Otter or Metis [16], and being awarded as the best new entrant tool of the competion [28].

In the rest of this section we shall give an outline, as concise as possible, of the theory and the architecture of the tool. This is important in order to understand its integration with the interactive prover.

2.1 The Superposition Calculus in a Nutshell

Let $\mathcal{F}$ bet a countable alphabet of functional symbols, and $\mathcal{V}$ a countable alphabet of variables. We denote with $\mathcal{T}(\mathcal{F}, \mathcal{V})$ the set of terms over $\mathcal{F}$ with variables in $\mathcal{V}$. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ is either a 0-arity element of $\mathcal{F}$ (constant), an element of $\mathcal{V}$ (variable), or an expression of the form $f(t_1, \ldots, t_n)$ where f is a element of $\mathcal{F}$ of arity n and $t_1, \ldots, t_n$ are terms.

Let s and r be two terms. $s|_p$ denotes the subterm of s at position p and $s[r]_p$ denotes the term s where the subterm at position p has been replaced by r.

A substitution is a mapping from variables to terms. Two terms s and t are unifiable if there exists a substitution σ such that $s\sigma = t\sigma$. In the previous case, σ is called a most general unifier (mgu) of s and t if for all substitution θ such that $s\theta = t\theta$, there exists a substitution τ which satisfies $\theta = \tau \circ \sigma$.

A literal is either an abstract predicate (represented by a term), or an equality between two terms. A clause $\Gamma \vdash \Delta$ is a pair of multisets of literals: the negative literals Γ, and the positive ones Δ. If $\Gamma = \emptyset$ (resp. $\Delta = \emptyset$), the clause is said to be positive (resp. negative).

A Horn clause is a clause with at most one positive literal. A unit clause is a clause composed of a single literal. A unit equality is a unit clause where the literal is an equality.

A strict ordering $\prec$ over $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is a transitive and irreflexive (possibly partial) binary relation. An ordering is *stable* under substitution if $s \prec t$ implies $s\sigma \prec t\sigma$ for all terms t, s and substitutions σ. A well founded monotonic ordering stable under substitution is called *reduction ordering* (see [11]). The intuition behind the use of reduction orderings for limiting the combinatorial explosion of new equations during inference, is to only rewrite big terms to smaller ones.

superposition left	**superposition right**	**equality resolution**
$\dfrac{\vdash l = r \qquad t_1 = t_2 \vdash}{(t_1[r]_p = t_2 \vdash)\sigma}$	$\dfrac{\vdash l = r \qquad \vdash t_1 = t_2}{(t_1[r]_p = t_2 \vdash)\sigma}$	$\dfrac{t_1 = t_2 \vdash}{\square}$

if $\sigma = mgu(l, t_1|_p), t_1|_p \neq x, l\sigma \not\preceq r\sigma$ and $t_1\sigma \not\preceq t_2\sigma$ if $\exists\sigma = mgu(t_1, t_2)$.

Fig. 1. Inference rules

For efficiency reasons, the calculus must be integrated with a few additional optimization rules, the most important one being demodulation ([29]).

2.2 The Main Algorithm

A naive implementation of the superposition calculus could just combine (superpose) all known clauses in all (admitted) ways, and repeat that process until the desired clause (called *goal*) is resolved. To avoid useless duplication of work, it is convenient to keep clauses in two distinct sets, traditionally called

subsumption	**tautology elimination**	**demodulation**
$\dfrac{S \cup \{C, D\}}{S \cup \{C\}}$ if $\exists \sigma, D\sigma \equiv C$	$\dfrac{S \cup \{\vdash t = t\}}{S}$	$\dfrac{S \cup \{\vdash l = r, C\}}{S \cup \{\vdash l = r, C[r\sigma]_p\}}$ if $l\sigma \equiv C\|_p$ and $l\sigma \succ r\sigma$

Fig. 2. Simplification rules

active and *passive*, with the general invariant that clauses in the active set have been already composed together in all possible ways. At every step, some clauses are selected from the passive set and added to the active set, then superposed with the active set, and consequently with themselves (*inference*). Finally, the newly generated clauses are added to the passive set (possibly after a simplification).

A natural selection strategy, resulting in a very predictable behaviour, would consist in selecting the whole passive set at each iteration, in the spirit of a breadth first search. Unfortunately the number of new equations generated at each step grows extremely fast, in practice preventing the iteratation of the main loop more than a few times.

To avoid this problem, all modern theorem provers (see e.g. [24]) adopt the opposite solution. According to some heuristics, like size and goal similarity for example, they select only *one* passive clause at each step. Not to loose completeness, some fairness conditions are taken into account (i.e. every passive clause will be eventually selected). This approach falls under the name *given-clause* (Figure 3), and its main advantage is that the passive set grows much slower, allowing a more focused and deeper inspection of the search space that consequently allows to find proofs that require a much higher number of main loop iterations.

The main drawback of this approach is that it makes the procedure way more sensible to the selection heuristics, leading to an essentially unpredictable behaviour.

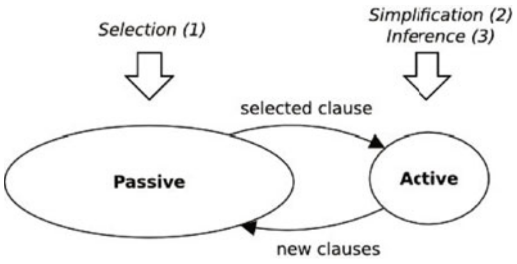


Fig. 3. Given-clause loop Numbers in parentheses reflect the steps order

2.3 Performance Issues

In order to obtain a state-of-the-art tool able to compete with the best available systems one has eventually to take into account a lot of optimizations and techniques developed for this purpose during the last thirty years.

In the following we shall shortly describe the most critical areas, and, for each of them, the approach adopted in Matita.

Orderings used to orientate rewriting rules. On complex problems (e.g. problems in the TPTP library with rating greater then 0.30) the choice of a good ordering for inference rules is of critical importance. We have implemented several orderings, comprising standard Knuth-Bendix (KBO), non recursive Knuth-Bendix (NRKBO), lexicographic path ordering (LPO) and recursive path ordering (RPO). The best suited ordering heavily depends on the kind of problem, and is hard to predict: our approach for the CADE ATP System Competition was to run in parallel different processes with different orderings.

On simpler problems (of the kind required for the smart application tactic of section 5), the given-clause algorithm is less sensitive to the term-ordering, and we may indifferently choose our preferred strategy, opportunely tuning the library (we are currently relying on LPO).

Selection strategy. The selection strategy currently implemented by Matita is a based on combination of age and weight. The weight is a positive integer that provides an estimation of the "complexity" of the clause, and is tightly related to the number of occurrences of symbols in it.

Since we are not interested in generating (counter) models of false statements, we renounced to be complete, and we silently drop inferred clauses that would slow down the main loop too much due to their excessive size.

Another similar optimization we did not implement but we could consider as a future development is Limited Resource Strategy [25], which basically allows the procedure to skip some inference steps if the resulting clauses are unlikely to be processed, mainly because of a lack of time.

Data structures and code optimization. We adopted relatively simple data structures (like discrimination [18] trees for term indexing), and a purely functional (in the sense of functional programming) implementation of them. After some code optimisation, we reached a point where very fast functions are the most expensive, because of the number of calls (implied by the number of clauses), even if they operate on simple data structures.

Since we are quite satisfied with the actual performance, we did not invest resources in adopting better data structures, but we believe that further optimizations will probably require implementing more elaborate data structures, such as substitution [14] or context trees [13], or even adopt an indexing technique that works modulo associativity and commutativity [12], that looks very promising when working on algebraic structures.

Demodulation. Another important issue for performance is demodulation: the given clause algorithm spends most of its time (up to 80%) in simplification, hence any improvement in this part of the code has a deep impact on performance. However, while reduction strategies, sharing issues and abstract machines have been extensively investigated for lambda calculus (and in general for left linear systems) less is known for general first order rewriting systems. In particular, while an innermost (eager) reduction strategy seem to work generally better than an outermost one (especially when combined with lexicographic path ordering), one could easily create examples showing an opposite behaviour (even supposing to always reduce needed redexes).

3 Integrating Superposition with Matita

3.1 Library Management

A possible approach to the integration of superposition with Matita is to solve all goals assuming that all equations part of the library lie in the passive set, augmented on the fly with the equations in the local context of the ongoing proof.

The big drawback of this approach is that, starting essentially from the same set of passive equations at each invocation on a different goal (differing only for the local context), the given clause algorithm would mostly repeat the same selection and composition operations over and over again. It is clear that, if we wish to superpose library equations, this operation should not be done at run time but in background, once and for all. Then we have to face a dual problem, namely to understand when stopping the saturation of the library with new equations, preventing an annoying pollution with trivial results that could have very nasty effects for selection and memory occupation. We would eventually like to have mechanisms to drive the saturation process.

A natural compromise is to look at library equations not as a passive set, but as the *active* one. This means that every time a new (unit) equation is added to the library it also goes through one main given-clause loop, as if it was the newly selected passive equation: it is simplified, composed with all existing active equations (i.e. all other equations in the library, up to simplification), and the newly created equations are added to the passive list. At run time, we shall then strongly privilege selection of local equations or goals.

This way, we have a natural, simple but traceable syntax to drive the saturation process, by just listing in library the selected equations. As a side effect, this approach reduces the verbosity of the library by making it unnecessary to declare (and name explicitly) trivial variants of available results that are automatically generated by superposition.

3.2 Interfacing CIC and the Superposition Engine

Our superposition tool is first order and untyped, while the Matita interactive prover is based on a variant of the Calculus of Inductive Construction (CIC), a complex higher-order intuitionistic logical systems with dependent types. The communication between the two components is hence far from trivial.

Instead of attempting a complex, faithful encoding of CIC in first order logic (that is essentially the approach adopted for HOL in [19]) we choose to follow a more naif approach, based on a forgetful translation that remove types and just keeps the first order applicative skeleton of CIC-terms.

In the opposite direction, we try to reconstruct the missing information by just exploiting the sophisticated inference capability of the Matita *refiner* [3], that is the tool in charge of transforming the user input into a machine understandable low-level CIC term.

Automation is thus a best effort service, in the sense that not only it may obviously fail to produce a proof, but sometimes it could produce an argument that Matita will fail to understand, independently from the fact if the delivered proof was "correct" or less.

The choice to deal with untyped first order equations in the superposition tool was mostly done for simplicity and modularity reasons. Moving towards a typed setting would require a much tighter integration between the superposition tool and the whole system, due to the complexity of typing and unification, but does not seem to pose any major theoretical problem.

The forgetful encoding. Equations $r =_T s$ of the calculus of constructions are translated to first order equations by merely following the applicative structure of r and s, and translating any other subterm into an opaque constant. The type T of the equation is recorded, but we are not supposed to be able to compute types for subterms.

In spite of the fact of neglecting types, the risk of producing "ill-typed" terms via superposition rules is moderate. Consider for instance the superposition left rule (the reasoning is similar for the other rules)

$$\frac{\vdash l = r \qquad t_1 = t_2 \vdash}{(t_1[r]_p = t_2 \vdash)\sigma}$$

where $\sigma = mgu(l, t_1|_p)$ and $l\sigma \not\preceq r\sigma$. The risk is that $t_1|_p$ has a different type from l, resulting into an illegal rewriting step. Note however that l and r are usually rigid terms, whose type is uniquely determined by the outermost symbol. Moreover, $t_1|_p$ cannot be a variable, hence they must share this outermost symbol. If l is not rigid, it is usually a variable x and if $x \in r$ (like e.g. in $x = x + 0$) we have (in most orderings) $l \preceq r$ that again rules out rewriting in the wrong direction.

This leads us to the following notion of *admissibility*. We say that an applicative term $f(x_1, \ldots, x_n)$ is *implicitly typed* if its type is uniquely determined by the type of f. We say that an equation $l = r$ is admissible if both l and r are implicitly typed, or $l \preceq r$ and r is implicitly typed. Non admissible equations are not taken into account by the superposition tool[1].

[1] A more liberal, but also slightly more expensive solution consists in indexing any equation and systematically try to read back each result of a superposition step in CIC, dropping it if it is not understood by the refiner.

In practice, most unit equalities are admissible. A typical counter example is an equation of the kind $\forall x, y : unit.x = y$, where *unit* is a singleton type.

On the other side, non-unit equalities are often not admissible. For instance, a clause of the kind $x \wedge y = true \vdash x = true$ could be used to rewrite any term to true, generating meaningless, ill typed clauses. Extending superposition beyond the unit equality case does eventually require to take types into consideration.

3.3 (Re)construction of the Proof Term

Translating a first-order resolution proof into a higher-order logic natural deduction proof is a notoriously difficult issue, even more delicate in case of intuitionistic systems, as the one supported by Matita. While resolution *per se* is a perfectly constructive process, skolemization and transformation into conjunctive normal forms are based on classical principles.

Our choice of focusing on the superposition calculus was also motivated by the fact it poses less difficulties, since skolemization is not needed and thus proofs have a rather simple intuitionistic interpretation.

Our technique for reconstructing a proof term relies as much as possible on the refinement capabilities of Matita, in particular for inferring implicit types. In the superposition module, each proof step is encoded as a tuple

```
Step of rule * int * int * direction * position * substitution
```

where rule is the kind of rule which has been applied, the two integers are the two *id's* of the composing equations (referring to a “bag” of unit clauses), direction is the direction the second equation is applied to the first one, position is a path inside the rewritten term and finally substitution is the mgu required for the rewriting step.

Every superposition step is encoded by one of the following terms:

$$\begin{aligned} eq\_ind\_l &: \forall A : \textsc{Type}.\forall x : A.\forall P : A \to \textsc{Prop}.P\ x \to \forall y : A.x = y \to P\ y \\ eq\_ind\_r &: \forall A : \textsc{Type}.\forall x : A.\forall P : A \to \textsc{Prop}.P\ x \to \forall y : A.y = x \to P\ y \end{aligned}$$

where left (_l) and right (_r) must be understood w.r.t. backward application, and where P is the one hole context that represents the position in which the superposition occurred.

At the end of the superposition procedure, if a proof is found, either a trivial goal has been generated, or a fact subsumes one of the active goals. In that latter case, we perform a rewriting step on the subsumed goal, so that we fall back into the previous case. Thus, when the procedure successfully stops, the selected clause is of the form $s = t$ where s and t are unifiable. We call it the meeting point, because forward steps (superposition right) and backward steps (superposition left) meet together when this trivial clause is generated, to compose the resulting proof. To generate a CIC proof term, the clauses are topologically sorted, their free variables are explicitly quantified, and nested let-in patterns are used to build the proof.

The most delicate point of the translation is closing each clause w.r.t. its free variables, since we should infer a type for them, and since CIC is an explicitly polymorphic language it is often the case that the order of abstractions does matter (e.g. variables standing for types must in general be abstracted before polymorphic variables).

The simplest solution is to generate so called "implicit" arguments leaving to the Matita *refiner* the burden of guessing them.

For instance, superposing lencat : $len\ A\ x + len\ A\ y = len\ A\ (x\underline{@}y)$ with catA : $x@(y@z) \overset{\leftarrow}{=} (x@y)@z$ at the underlined position and in the given direction gives rise to the following piece of code, where question marks stand for implicit arguments:

```
let clause_59:
   ∀w :?.∀x :?.∀y :?.∀z :?.
      len w (x@y) + len w z = len w (x@(y@z))
 :=
   λw :?.λz :?.λx :?.λy :?.
      eq_ind_r (List w) ((x@y)@z))
         (λhole : List w.len w (x@y) + len w z = len w hole)
         (lencat w (x@y) z) (x@(y@z)) (catA w x y z) in
```

Note that w *must* be abstracted first, since it occurs in the (to be inferred) types for x, y and z. Also note the one hole context expressed as an anonymous function whose abstracted variable is named *hole*, corresponding to the position of $x\underline{@}y$ in the statement of lencat.

The interesting point is that refining is a complex operation, using e.g. hints, and possibly calling back the automation itself: the interpretation of the proof becomes hence a dialog between the system and its automation components, aimed to figure out a correct interpretation out of a rough initial trace.

A more sophisticated translation, aimed to produce a really nice, human-readable output in the form of a chain of equations, is described in [6].

4 Smart Application

The most interesting application of superposition (apart from its use for solving equational goals) is the implementation of a more flexible application tactic. As a matter of fact, one of the most annoying aspects of formal development is the need of transforming notions to match a given, existing result. As explained in the introduction, most of these transformations are completely transparent to the typical mathematical discourse, and we would like to obtain a similar behaviour in interactive provers.

Given a goal $\mathcal{B}$ and a theorem t: $A \to B$, the goal is to try to match B with $\mathcal{B}$ *up to the available equational knowledge base*, in order to apply t. We call it, the *smart application* of t to G. We use superposition in the most direct way, exploiting on one side the higher-order features of CIC, and on the other the fact that the translation to first order terms does not make any difference between predicates and functions: we simply generate a goal $B = \mathcal{B}$ and pass it to the superposition tool (actually, it was precisely this kind of operation

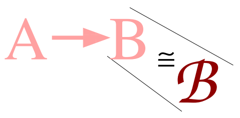


Fig. 4. Smart application

that motivated our original interest in superposition). If a proof is found, $\mathcal{B}$ is transformed into B by rewriting and t is then normally applied.

Superposition, addressing a typically undecidable problem, can easily diverge, while we would like to have a reasonably fast answer to the smart application invocation, as for any other tactic of the system. We could simply add a timeout, but we prefer to take a different, more predictable approach. As we already said, the overall idea is that superposition right steps - realising the *saturation* of the equational theory - should be thought of as background operations. Hence, at run time, we should conceptually work as if we had a *confluent* rewriting system, and the only operation worth to do is *narrowing* (that is, left superposition steps). Narrowing too can be undecidable, hence we fix a given number of narrowing operations to apply to each goal (where the new goal instances generated at each step are treated in parallel). The number of narrowing steps can be fixed by the user, but a really small number is usually enough to solve the problem if a solution exists.

5 Examples

Example 1. Suppose we wish to prove that the successor function is le-reflecting, namely

$$(*) \qquad \forall n, m. Sn \leq Sm \rightarrow n \leq m$$

Suppose we already proved that the predecessor function is monotonic:

$$monotonic\_pred : \forall n, m. n \leq m \rightarrow pred\ n \leq pred\ m$$

We would like to merely "apply" the latter to prove the former. Just relying on unification, this would not be possible, since there is no way to match $pred\ X \leq pred\ Y$ versus $n \leq m$ unless *narrowing* the former. By superposing twice with the equation $\forall n. pred(Sn) = n$ we immediately solve our matching problem via the substitution $\{X := Sn, Y := Sm\}$. Hence, the smart application of *monotonic_pred* to the goal $n \leq m$ succeeds, opening the new goal $Sn \leq Sm$ that is the assumption in $(*)$.

Example 2. Suppose we wish to prove $n \leq m * n$ for all natural numbers n, m. Suppose we already proved that multiplication is left-monotonic, namely

$$monotonic\_le\_times\_l : \forall n, a, b. a \leq b \rightarrow a * n \leq b * n$$

In order to apply this result, the system has to find a suitable $?_a$ such that $?_a * n = n$, that is easily provided by the identity law for times.

Example 3. In many cases, we just have local equational variants of the needed results. Suppose for instance we proved that multiplication in le-reflecting in its right parameter:

$$le\_times\_to\_le\_times\_r : \forall a, n, m. a * n \leq a * m \rightarrow n \leq m$$

Since times is commutative, this also trivially implies the left version:

$$monotonic\_le\_times\_l : \forall a, n, m. n * a \leq m * a \rightarrow n \leq m$$

Formally, suppose to have the goal $n \leq m$ under the assumption $(H)\ n{*}a \leq m{*}a$. By applying *le_times_to_le_times_r* we obtain a new goal $?_a * n \leq ?_a * m$ that is a smart variant of H.

Example 4. Suppose we wish to prove that $(H)\ a * (Sn) \leq a * (Sm)$ implies $a{*}n \leq a{*}m$, where S is the successor function (this is a subcase in the inductive proof that the product by a positive constant a is le-reflecting). Suppose we already proved that the sum is le-reflecting in its second argument:

$$le\_plus\_to\_le\_plus\_r : \forall a, n, m. a + n \leq a + m \rightarrow n \leq m$$

By applying this result we obtain the new goal $?_a + a * n \leq ?_a + a * m$, and if we have the expected equations for times, we can close the proof by a smart application of H.

Example 5. Consider the goal $n < 2 * m$ under the assumptions $(H)\ 0 < m$ and $(H1)\ n \leq m$. Suppose that we defined $x < y$ as $x + 1 \leq y$. Morevoer, by the defining equation of times we should know something like $2 * m = m + (m + 0)$.[2] Hence the goal is equal to $n + 1 \leq m + (m + 0)$, and the idea is to use again the monotonicity of plus (in both arguments):

$$le\_plus\ n\ m : \forall a, b. n \leq m \rightarrow a \leq b \rightarrow n + a \leq m + b$$

The smart application of this term to the goal $n < 2 * m$ succeeds, generating the two subgoals $n \leq m$ and $1 \leq m + 0$. The former one is the assumption $H1$, while the latter is a *smart* variant of H.

Example 6. Let us make an example inspired by the theory of programming languages. Suppose to have a typing relation $\Gamma \vdash M : N$ stating that in the environment Γ the term M has type N. If we work in De Bruijn notation, the weakening rule requires lifting[3]

$$\text{weak} : \Gamma \vdash M : N \rightarrow \Gamma, A \vdash \uparrow^1(M) : \uparrow^1(N)$$

Suppose now we have an axiom stating that $\vdash * : \Box$ where $*$ and $\Box$ are two given sorts. We would like to generalize the previous result to an arbitrary (legal) context Γ. To prove this, we have just to apply weakenings (reasoning by induction

[2] The precise shape depends by the specific equations available on times.

[3] The lifting operation $\uparrow^n(M)$ is meant to relocate the term M under n additional levels of bindings: in other words, it increases by n all free variables in M.

on Γ). However, the normal application of *weak* would fail, since the system should be able to guess two terms M and N such $\uparrow^1(M) = *$ and $\uparrow^1(N) = \square$. If we know that for any constant c, $\uparrow^1(c) = c$ (that comes from the definition of lifting) we may use such an equation to enable the smart application of *weak*.

Performance. In Figure 5 we give the execution times for the examples of smart applications discussed in the previous section (in bytecode). Considering these times, it is important to stress again that the smart application tactics does not take any hint about the equations it is supposed to use to solve the matching problem, but exploits all the equations available in the (imported sections of the) library.

The important point is that smart application is fast enough to not disturb the interactive dialog with the proof assistant, while providing a much higher degree of flexibility than the traditional application.

example	applied term	execution time
1	*momonotonic_pred*	$0.16s.$
2	*momonotonic_le_times_l*	$0.23s.$
3	$H : a * n \leq a * m$	$0.22s.$
4	$H : a * (Sn) \leq a * (Sm)$	$0.15s.$
5	*le_plus n m*	$0.57s.$
6	*weak*	$0.15s.$

Fig. 5. Smart application execution times

6 Related Works and Systems

Matita was essentially conceived as a light version of Coq [9], sharing the same foundational logic (the Calculus of Inductive Constructions) and being partially compatible with it (see [4] for a discussion of the main differences between the two systems at kernel level). Hence, Coq is also the most natural touchstone for our work. The `auto` tactic of Coq does not perform rewriting; this is only done by a couple of specialized tactics, called `auto rewrite` and `congruence`. The first tactic carries out rewritings according to sets of oriented equational rules explicitly passed as arguments to the tactic (and previously build by the user with suitable vernacular commands). Each rewriting rule in some base is applied to the goal until no further reduction is possible. The tactic does not perform narrowing, nor any form of completion. The `congruence` tactic implements the standard Nelson and Oppen congruence closure algorithm [21], which is a decision procedure for *ground* equalities with uninterpreted symbols; the Coq tactic only deals with equalities in the local context. Both Coq tactics are sensibly weaker than superposition that seems to provide a good surrogate for several decision procedures for various theories, as well as a simple framework for composing them (see e.g [2]).

Comparing the integration of superposition in Matita with similar functionalities provided by Isabelle is twofold complex, due not only to the different approaches, but also to the different underlying logics.

In Isabelle, equational reasoning can be both delegated to external tools or dealt with internally by the so called *simplifier*. Some of the the external tools Isabelle is interfaced with provide full support to paramodulation (and hence superposition), but the integration with them is stateless, possibly requiring to pass hundreads of theorems (all the current visible environment) at each invocation. In Matita, the active set is *persistent*, and grows as the user proves new equations.

Of more interest is the comparison with Isabelle's internal simplifier. The integration of this tool with the library is manual: only lemmas explicitly labelled and oriented by the user are taken into account by the simplifier. Moreover, these lemmas are only used to demodulate and are not combined together to infer new rewriting rules. Nevertheless, a pre-processing phase allows the user to label theorems whose shape is not an equation. For example a conjunction of two equations is interpreted as two distinct rewriting rules, or a negative statement $\neg A$ is understood as $A = False$. The simplifier is also able to take into account guarded equations as long as their premises can be solved by the simplifier itself. Finally it detects equations that cannot be oriented by the user, like commutativity, and restricts their application according to the demodulation rule using a predefined lexicographic order.

Anyway, the main difference from the user's perspective comes from a deep reason that has little to do with the simplifier or any other implemented machinery. Since Isabelle is based on classical logic, co-implication can be expressed as an equality. Hence, in Isabelle we can prove much more equations at the prosisitional level and use them for rewriting. Any concrete comparison between the two provers with respect to equational reasoning is thus inherently biased, since many problems encountered in one system would look meaningless, artificial or trivial when transposed into the other one.

7 Conclusions

We described in this paper the "smart" application tactic of the Matita interactive theorem prover. The tactics allow the backward application of a theorem to a goal, where matching is done up to the data base of all equations available in the library. The implementation of the tactics relies on a compact superposition tool, whose architecture and integration within Matita have been described in the first sections. The tool is already performant (it was awarded best new entrant tool at the 22nd CADE ATP System Competition) but many improvements can still be done for efficiency, such as the implementation of more sophisticated data structures for indexes (we currently use discrimination trees).

Another interesting research direction is to extend the management of equality to setoid rewriting [27]. Indeed, the current version of the superposition tool just works with an intensional equality, and it would be interesting to try to figure out how to handle more general binary relations. The hard problem is proof reconstruction, but again it seems possible to exploit the sophisticated capabilities of the Matita refiner [3] to automatically check the legality of the rewriting operation (i.e. the monotonicity of the context inside which rewriting has to be performed), exploiting some of the ideas outlined in [26].

One of the most promising uses of smart application is inside the backward-based automation tactic of Matita. In fact, smart application allows a smooth integration of equational reasoning with the prolog-like backward applicative mechanisms that, according to our first experimentations looks extremely promising. As a matter of fact, the weakest point of smart application is that it does not relieve the user form the effort of finding the "right" theorems in the library or of guessing/remembering their names (although it allows to sensibly reduce the need of variants of a given statement in the repository). A suitably constrained automation tactic could entirely replace the user in the quest of candidates for the smart application tactic. Since searching is a relatively expensive operation, the idea is to ask the automation tactic to return an explicit trace of the resulting proof (essentially, a sequence of smart applications) to speed-up its re-execution during script development.

Acknowledgements. We would like to thank Alberto Griggio and Maxime Dénès for their contribution to the implementation of the superposition tool.

References

1. Ahrendt, W., Beckert, B., Hähnle, R., Menzel, W., Reif, W., Schellhorn, G., Schmitt, P.H.: Integrating automated and interactive theorem proving. In: Automated Deduction — A Basis for Applications. Systems and Implementation Techniques of Applied Logic Series, vol. II(9), pp. 97–116. Kluwer, Dordrecht (1998)
2. Armando, A., Ranise, S., Rusinowitch, M.: Uniform derivation of decision procedures by superposition. In: Fribourg, L. (ed.) CSL 2001 and EACSL 2001. LNCS, vol. 2142, pp. 513–527. Springer, Heidelberg (2001)
3. Asperti, A., Ricciotti, W., Sacerdoti Coen, C., Tassi, E.: Hints in unification. In: Urban, C. (ed.) TPHOLs 2009. LNCS, vol. 5674, pp. 84–98. Springer, Heidelberg (2009)
4. Asperti, A., Ricciotti, W., Sacerdoti Coen, C., Tassi, E.: A compact kernel for the Calculus of Inductive Constructions. Sadhana 34(1), 71–144 (2009)
5. Asperti, A., Sacerdoti Coen, C., Tassi, E., Zacchiroli, S.: User interaction with the Matita proof assistant. Journal of Automated Reasoning 39(2), 109–139 (2007)
6. Asperti, A., Tassi, E.: Higher order proof reconstruction from paramodulation-based refutations: The unit equality case. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, pp. 146–160. Springer, Heidelberg (2007)
7. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. 4(3), 217–247 (1994)
8. Bezem, M., Hendriks, D., de Nivelle, H.: Automated proof construction in type theory using resolution. J. Autom. Reasoning 29(3-4), 253–275 (2002)
9. The Coq proof-assistant (2009), http://coq.inria.fr

10. Degtyarev, A., Voronkov, A.: Equality reasoning in sequent-based calculi. In: Handbook of Automated Reasoning, pp. 611–706. Elsevier and MIT Press (2001)
11. Dershowitz, N.: Orderings for term-rewriting systems. Theor. Comput. Sci. 17, 279–301 (1982)
12. Dershowitz, N., Hsiang, J., Josephson, N.A., Plaisted, D.A.: Associative-commutative rewriting. In: IJCAI, pp. 940–944 (1983)
13. Ganzinger, H., Nieuwenhuis, R., Nivela, P.: Fast term indexing with coded context trees. J. Autom. Reasoning 32(2), 103–120 (2004)
14. Graf, P.: Substitution tree indexing. In: Hsiang, J. (ed.) RTA 1995. LNCS, vol. 914, pp. 117–131. Springer, Heidelberg (1995)
15. Hurd, J.: Integrating gandalf and hol. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin, C., Théry, L. (eds.) TPHOLs 1999. LNCS, vol. 1690, pp. 311–322. Springer, Heidelberg (1999)
16. Hurd, J.: First-order proof tactics in higher-order logic theorem provers. Technical Report NASA/CP-2003-212448, Nasa technical reports (2003)
17. Knuth, D., Bendix, P.: Simple word problems in universal algebras. In: Leech, J. (ed.) Computational problems in Abstract Algebra, pp. 263–297 (1970)
18. McCune, W.: Experiments with discrimination tree indexing and path indexing for term retrieval. Journal of Automated Reasoning 9(2), 147–167 (1992)
19. Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. J. Autom. Reasoning 40(1), 35–60 (2008)
20. Meng, J., Quigley, C., Paulson, L.C.: Automation for interactive proof: First prototype. Inf. Comput. 204(10), 1575–1596 (2006)
21. Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. J. ACM 27(2), 356–364 (1980)
22. Nieuwenhuis, R., Rubio, A.: Paramodulation-based thorem proving. In: Handbook of Automated Reasoning, pp. 443–471. Elsevier/MIT Press (2001)
23. Paulson, L.C.: A generic tableau prover and its integration with isabelle. J. UCS 5(3), 73–87 (1999)
24. Riazanov, A., Voronkov, A.: The design and implementation of vampire. AI Communications 15(2-3), 91–110 (2002)
25. Riazanov, A., Voronkov, A.: Limited resource strategy in resolution theorem proving. J. Symb. Comput. 36(1-2), 101–115 (2003)
26. Sacerdoti Coen, C., Tassi, E.: A constructive and formal proof of Lebesgue's dominated convergence theorem in the interactive theorem prover Matita. Journal of Formalized Reasoning 1, 51–89 (2008)
27. Sozeau, M.: A new look at generalized rewriting in type theory. Journal of Formalized Reasoning 2(1), 41–62 (2009)
28. Sutcliffe, G.: The 4th ijcar automated theorem proving system competition - casc-j4. AI Commun. 22(1), 59–72 (2009)
29. Wos, L., Robinson, G.A., Carson, D.F., Shalla, L.: The concept of demodulation in theorem proving. J. ACM 14(4), 698–709 (1967)

Electronic Geometry Textbook: A Geometric Textbook Knowledge Management System

Xiaoyu Chen

LMIB – SKLSDE – School of Mathematics and Systems Science,
Beihang University, Beijing 100191, China

Abstract. Electronic Geometry Textbook is a knowledge management system that manages geometric textbook knowledge to enable users to construct and share dynamic geometry textbooks interactively and efficiently. Based on a knowledge base organizing and storing the knowledge represented in specific languages, the system implements interfaces for maintaining the data representing that knowledge as well as relations among those data, for automatically generating readable documents for viewing or printing, and for automatically discovering the relations among knowledge data. An interface has been developed for users to create geometry textbooks with automatic checking, in real time, of the consistency of the structure of each resulting textbook. By integrating an external geometric theorem prover and an external dynamic geometry software package, the system offers the facilities for automatically proving theorems and generating dynamic figures in the created textbooks. This paper provides a comprehensive account of the current version of Electronic Geometry Textbook.

1 Introduction

1.1 Motivation

When we speak about managing knowledge, we may start by thinking about textbooks, where knowledge is organized systematically and presented hierarchically according to its internal logical relations. Since textbooks provide a well-arranged structure of domain knowledge, they play an important role in education and research; they record knowledge and impart it to new learners. The Electronic Geometry Textbook (EGT) is a knowledge management system for geometric knowledge, built so that users may construct and publish dynamic geometry textbooks interactively and efficiently. The objective of our textbook project is to explore the approaches to managing knowledge by integrating available software tools and providing a system that assists human authors to create dynamic, interactive, and machine-processable textbooks (instead of the traditional static textbooks). EGT is motivated by the following three considerations:

(1) Textbooks are a standard form for the storage, organization, and presentation of systematic domain knowledge. For different pedagogical purposes, the same knowledge may be adopted by different textbooks as a part of the theories involved. In order to share and reuse sophisticated knowledge, we need to

build up a standard knowledge base that stores and organizes data describing the textbook knowledge. Authors can contribute knowledge (as data encoded in some knowledge representation format) to the knowledge base and *construct* textbooks by reusing pieces of knowledge already in the knowledge base as constituents in a new textbook.

EGT offers such an environment that maintains (i.e., creates, removes, modifies, and queries, etc.) and shares knowledge data with an appropriate granularity, constructs textbooks by interactively arranging the knowledge data selected and retrieved from the knowledge base, and automatically generates styled documents for browsing and printing the textbooks produced.

(2) When creating a textbook for learners, one needs to determine an appropriate narrative structure so as to arrange the contents involved in the textbook. Although one can make one's own decision as to what knowledge is to be chosen, there are common practices and implicit conventions in a scientific community as to how knowledge should be organized, formulated, and presented. For example, it is commonly accepted that a proved proposition is a lemma only if it is used in the proof of a theorem and a corollary is a true proposition that follows from a theorem. The domain knowledge presented in a textbook should be structured systematically, hierarchically, and logically, i.e., from the simplest to the most complicated and from the basic to the advanced. For example, the definition for each concept in a statement (such as a theorem, exercise, or example statement) should have been given before the statement. In order to produce such a sound and usable textbook, we need to be given feedback if the narrative structure disobeys the conventional rules during the process of construction.

EGT offers such a facility that assists a user to automatically check, in real time, whether the constructed textbook has a satisfiable and reasonable narrative structure.

(3) In recent years, many creative methods have been proposed for automated geometry theorem proving, such as algebraic approaches (the most powerful, although just decision methods) which convert the problem of geometrical reasoning to that of solving algebraic systems, coordinate-free approaches which convert the problem to the counterpart of algebraic calculation with respect to some geometric quantities, and traditional AI approaches. [6] Many geometry software tools have implemented these approaches and provided the functionality of automated reasoning, such as GEOTHER [12], Geometry Expert [10], and GCLC [8], and of interactive proving, such as GeoProof [11]. These tools not only help geometry researchers to discover new and more valuable and complex theorems, but also support geometry education. Geometry textbooks include many interesting and complicated theorems whose proofs are given and checked by the authors. As the traditional formal logical methods do not work very efficiently in automated geometry deduction, the techniques for automated geometric proof checking have not been well developed. However, it is still helpful to make use of the automated theorem provers to assist an author to determine whether a proposition written in a textbook is logically true in order to ensure the correctness of the textbook.

In addition, geometry deals with graphical objects abstracted from the real visual world. Intuitive figures are indispensable constituents of textbooks. With the help of a computer, one can draw high-resolution and accurate figures interactively by using a mouse and following the instructions provided during the construction. For instance, after selecting two points and an instruction “make the mid-point of two points” by using a mouse, the mid-point will be constructed and displayed in the figure. An even more important enhancement resulting from this interactive facility is that the steps of constructing a figure can be recorded and redone quickly. For example, as a result one drags a free point from one place to another and the figure will be updated immediately. One can explore a figure and experience what happens when components are moved. This dynamic feature makes geometry more vivid. Dynamic geometry software has been developed to implement these features and applied in geometry education and research, such as Cabri [2], SketchPad [19], Cinderella [7], and GeoGebra [9]. It is useful to apply dynamic geometry software to make the figures in textbooks more intuitive and explorable than the traditional static ones.

EGT offers interfaces to knowledge data and to selected external geometry software packages for proving theorems and generating dynamic figures automatically.

1.2 Originality

The idea of designing and developing such an integrated software system in the form of a textbook for systematic and interactive management of geometric knowledge originates from Dongming Wang [4] who has been working on automated geometric reasoning for the last two decades. The author has been stimulated to elaborate the idea and to undertake the implementation of a system himself. We consider geometry a unique and rich subject of mathematics that should be chosen for study in the context of knowledge management. In such a study, the full power of computers for symbolic, numeric, and graphical computing and data processing may be used and our ideas may be effectively tested.

Several e-learning and intelligent tutoring systems for mathematics have been proposed and developed, such as LeActiveMath [16], ActiveMath [1], and MathDox [17]. They offer facilities for generating courseware which adapts to students, tutoring students interactively with diagnoses of mistakes and adaptive feedback, analyzing and evaluating students’ abilities, etc. These systems are learner-centred and support the learner’s initiative. However, EGT is designed mainly to assist human authors in constructing dynamic textbooks. The process is mostly author-driven and manipulations of the textbook are allowed and may lead to new, modified, or improved versions of a textbook. EGT’s innovations may be seen in the following three aspects:

1. EGT products can be viewed or printed as traditional textbooks (static documents) and also run as dynamic software on a computer. Textbook knowledge is shared at an appropriate granularity and textbooks can be constructed and

maintained interactively. For example, a textbook can be seen as an arrangement of nodes that refer to the corresponding textbook contents. One can perform a series of manipulations adding, inserting, removing, modifying, and restructuring the nodes involved, and meanwhile the generated documents for browsing and printing can be updated automatically;
2. EGT can assist users to analyze the narrative structure of the textbooks constructed and automatically find the parts inconsistent with the conventional rules for writing textbooks, in real time. We call this process *consistency-checking* of the structure of the textbook. For example, the definition of a `median` of a triangle can be created only if the definition of `midpoint` of a segment has already been introduced in the textbook;
3. EGT integrates stand-alone geometry software packages for automated theorem proving and dynamic figure generation. This provides the constructed textbooks with dynamic features. For example, the theorems in the textbooks can be automatically proved by invoking geometric theorem provers, and the figures are automatically constructed by applying dynamic geometry software.

This paper describes the relevant design principles of EGT including architectural issues, the structure of the geometric knowledge base, knowledge representation, and the communication with available geometry software packages. We present the main features of the current version of EGT including maintaining geometric knowledge data for constructing textbooks interactively, rendering the textbooks in readable documents both in English and Chinese, proving the theorems and drawing the dynamic figures automatically by interfacing with the selected geometry software packages. While plane Euclidean geometry is the target of our current investigation, the ideas also apply to, or invite the attempt to apply them to, other geometries.

2 Design Principles of Electronic Geometry Textbook

We describe the architecture of the system and present the main design principles for a geometric textbook knowledge base and the representation of knowledge. More details about the design methodology have been discussed in [5].

2.1 Architecture and Communication

Now we give a bird's eye view of how the system works and which components carry out which tasks. In what follows, we refer to Fig. 1 which gives an overview over the EGT components and their communications. The *textbook knowledge base* is the kernel component of the system, storing and organizing the shared knowledge data. Via the *user interface*, users can construct textbooks by invoking the *manipulation module* to perform the manipulations of creating new knowledge data, retrieving needed knowledge data, and modifying knowledge data on the textbook knowledge base. Meanwhile, the *consistency-checking module* will check the consistency of the constructed textbooks in real

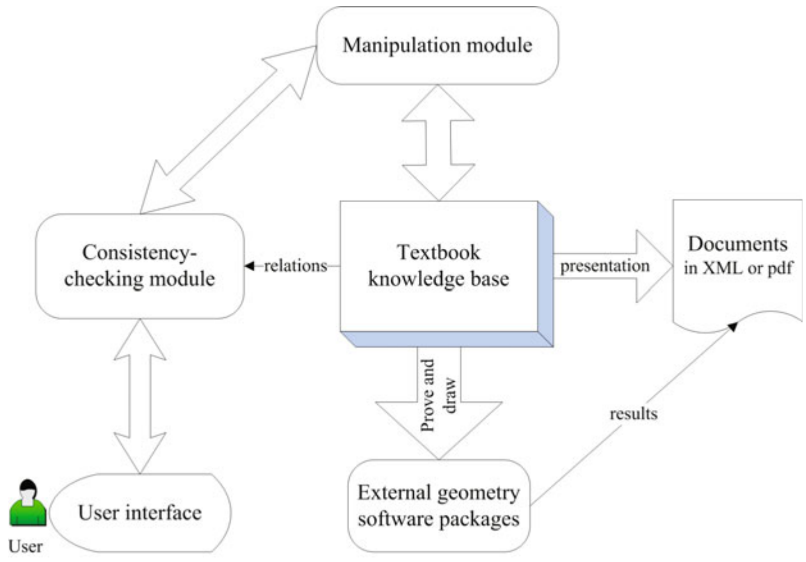


Fig. 1. Architecture of the Electronic Geometry Textbook system

time and provide feedback to the user interface. The textbooks constructed can be presented in readable documents for rendering in a browser or printing on paper. The theorems in the constructed textbooks can be proved and dynamic figures can be drawn automatically by interfacing with external geometry software packages.

From the description of the system, we can conclude that the system has as its foundation a textbook knowledge base. To manipulate knowledge efficiently and appropriately, one main task is creating a well-structured, manageable, and suitable knowledge base. On the other hand, to communicate knowledge, formalizing and representing knowledge as data in a processable way is the other main task.

2.2 Design of Geometric Textbook Knowledge Base

Generally speaking, the design of a knowledge base involves the following two aspects.

Knowledge Data Granularity. While creating a textbook, the author should identify the objects of domain knowledge, categorize them, and rank them according to their relationships. In order to support the manipulations of constructing dynamic textbooks interactively, we need to encapsulate knowledge data at an appropriate granularity. If the granularity is too fine, the process of constructing textbooks may be too complicated to manage. If the granularity is not fine enough, the process may be trivial and not subject to manipulation.

We use the notion of a *knowledge object* to represent the unit of textbook knowledge which can be recognized, differentiated, understood, and manipulated while constructing textbooks. For example, the definition of a concept is a knowledge object which gives meaning to the concept; a theorem is a knowledge object which is a true proposition in the domain; a proof demonstrates that a proposition is true; an exercise or example needs to be solved by applying some knowledge. Working from the common or implicit conventions in traditional textbooks, we classify the knowledge objects into the following types: Concept (Definition), Axiom, Lemma, Theorem, Corollary, Conjecture, Proof, Problem, Example, Exercise, Solution, Algorithm, Introduction, and Remark. Although this classification may be argued over and needs to be justified, what is essential in our approach is to encapsulate knowledge data into certain knowledge objects with the same structure. Within different types of knowledge objects, certain data items are created to store knowledge data for different applications on the objects. For example, the data stored in a data item *naturalRepresentation* is used for presentation, the data stored in *algebraicRepresentation* is used for automatic proving by algebraic methods, the data stored in *diagramInstruction* is used for automatic dynamic figure drawing, etc. One may refer to [3] for the details of the design of the structure within each type of knowledge object.

For a textbook, the index is an important component; it allows a reader to see what is included and to navigate within the textbook. We use *category* to represent such a hierarchical structure so that a *category object* has a group of subcategories or knowledge objects as its members. For example, each chapter in the textbook is a category which usually has subcategories of sections, and each section may include a group of knowledge objects. The categories should usually be contributed by the authors using their comprehensive understanding of the domain knowledge.

The textbook can be viewed as a linear arrangement of knowledge objects and categories. The process of constructing textbooks can be viewed as a series of manipulations (adding, inserting, removing, modifying, and restructuring) of these knowledge objects and categories.

Relations. Geometric knowledge is accumulated step by step, e.g., by introducing new concepts using already defined concepts, deriving useful properties about new concepts, and proving or discovering theorems relating old and new concepts. It does not lie flat but is piled up with a certain intrinsic structure of hierarchy. Some knowledge pieces serve as preliminaries for higher-level knowledge. The conventional rules for writing textbooks depend on the relationships of the knowledge involved. Therefore, the relations among category objects and knowledge objects must be captured to define the structure of the geometric textbook knowledge base, and then to assist users to perform consistency-checking of the structure of textbooks.

The relations we are interested in may involve the consideration of, and abstractions from, pedagogical rules and textbook writing conventions. We have identified 17 types of relations among knowledge objects and category objects: Inclusion ($\rightarrow_{\text{include}}$), Context ($\rightarrow_{\text{contextOf}}$), Inheritance ($\rightarrow_{\text{inherit}}$), Derivation

($\rightarrow_{\text{deriveFrom}}$), Implication ($\rightarrow_{\text{imply}}$), Property ($\rightarrow_{\text{hasProperty}}$), Decision ($\rightarrow_{\text{decide}}$), Justification ($\rightarrow_{\text{justify}}$), Introduction ($\rightarrow_{\text{introduce}}$), Remark ($\rightarrow_{\text{remarkOn}}$), Complication ($\rightarrow_{\text{complicate}}$), Solution ($\rightarrow_{\text{solve}}$), Application ($\rightarrow_{\text{applyOn}}$), Equality ($\leftrightarrow_{\text{equal}}$), Exercise ($\rightarrow_{\text{exerciseOf}}$), Example ($\rightarrow_{\text{exampleOf}}$), Association ($\leftrightarrow_{\text{associate}}$).

The conventional rules for writing textbooks can be written with these relations. For example, a relation $D \rightarrow_{\text{contextOf}} T$ (where D is a definition and T is a theorem) means D provides the context for T. The rule that D should be presented before T in the textbook can be derived from the meaning of the Context relation. Therefore, if D is arranged after T when a user constructs a textbook, then the structure of the textbook is inconsistent and needs to be restructured. In [15], F. Kamareddine et al. present an ontology and an associated markup system for annotating mathematical documents so that the graph of logical precedences (the conventional rules in our context) of the annotated parts of text can be acquired and analyzed automatically. However, we are concerned with not only how to acquire these rules but also how to make use of them to decide whether a textbook is constructed in an appropriate and soundly presented structure, i.e., whether the structure of the textbook is consistent. The inconsistencies found by the current system are limited to those disobeying the rules derived from the existing relations.

The geometric textbook knowledge base is then created to store textbook knowledge data, with well-defined structures for the types of, and relations between, the knowledge data stored.

2.3 Knowledge Representation

The knowledge data stored in the data items of knowledge objects will be applied in different situations. One important application is to communicate with external geometry software packages. It is necessary to represent the geometric statements of the involved knowledge objects in a formal language and to transform them automatically into equivalent ones that the target geometry software packages can identify and manipulate via specific interfaces. The Intergeo project [13] is an ongoing European project, one of whose objectives is to attack the barrier of lack of interoperability by offering a common file format for specifying dynamic figures. However, we have designed a geometry programming language in which one can easily specify geometric statements of definitions, theorems, axioms, and problems, etc. by using customized concepts. We have also implemented automatic translation of this language into the native languages of the geometry software packages targeted for communication. We present some examples using this language here and describe how to process this language in Section 3.4.

`Simson's theorem` in English is "The feet of the perpendiculars from a point to the three sides of a triangle are collinear if and only if the point lies on the circumcircle."

The formal representation of `Simson's theorem` is "A := point(); B := point(); C := point(); D := point(); incident(D, circumcircle(triangle(A, B, C))) $\Leftrightarrow$ collinear(foot(D, line(A, B)), foot(D, line(B, C)), foot(D, line(A, C)))."

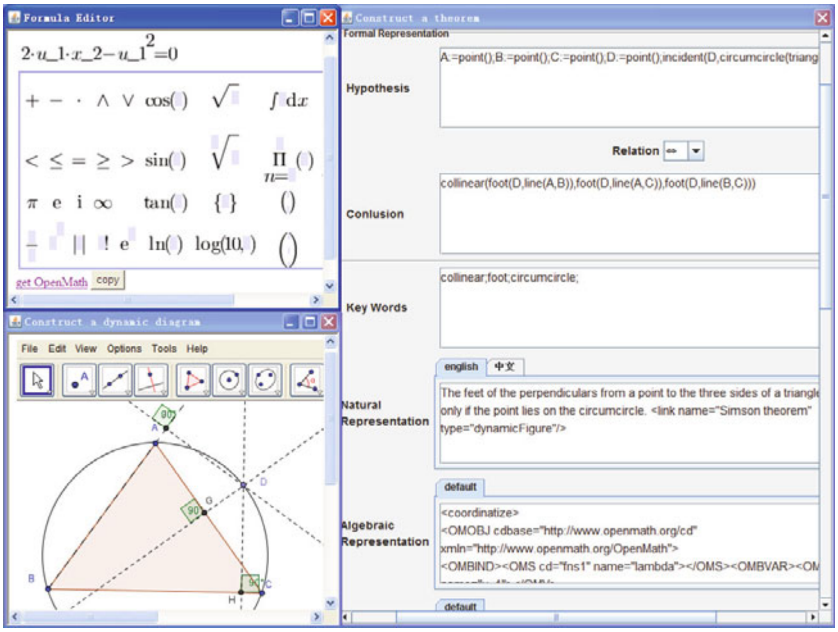
The intersection point of two lines l and m is defined as "intersection (l, m) $\triangleq$ [A::Point where incident(A, l) $\wedge$ incident(A, m)]."

3 Technical Realization of Electronic Geometry Textbook

In this section, we present the technical details of implementing the components of the Electronic Geometry Textbook system.

3.1 Creation of the Geometric Textbook Knowledge Base

The textbook knowledge base stores the knowledge data of knowledge objects and category objects, as well as their relations. These objects are the units that may be managed, retrieved, and processed by the other modules. In order to identify and distinguish them, the system automatically assigns each object a unique *objectID*. Then relational tables are defined that specify how data items are related with the objects and what the relations among these objects are. [3] We have created a database containing these tables in MS SQL Server and chosen Java as the programming language to develop the interfaces for users to maintain the knowledge data of the Concept (Definition), Axiom, Lemma, Theorem, Corollary, Conjecture, Problem, Example, Exercise, Proof, Solution, Introduction, Remark, and



$2\cdot u\_1\cdot x\_2-u\_1^2=0$

Hypothesis: A:=point();B:=point();C:=point();D:=point();incident(D,circumcircle(triang

Relation: ⇔

Conlusion: collinear(foot(D,line(A,B)),foot(D,line(A,C)),foot(D,line(B,C)))

Key Words: collinear;foot;circumcircle;

Natural Representation: The feet of the perpendiculars from a point to the three sides of a triangl only if the point lies on the circumcircle. <link name="Simson theorem" type="dynamicFigure"/>

Algebraic Representation:
```
<coordinatize>
<OMOBJ cdbase="http://www.openmath.org/cd"
xmln="http://www.openmath.org/OpenMath">
<OMBIND><OMS cd="fns1" name="lambda"></OMS><OMBVAR><OM
```

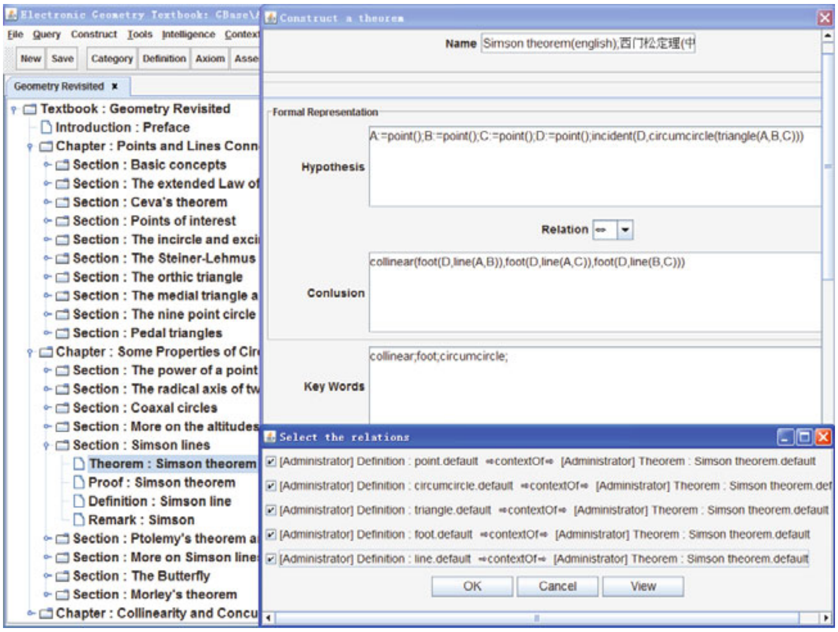
Fig. 2. Constructing `Simson's theorem` object

Category objects as well as their relations. Our system employs several external packages for editing specific data. The dynamic mathematics software GeoGebra is used for producing dynamic figures. The MathDox formula editor [18] is used to create expressions encoded in OpenMath for the algebraic representations. One can construct, for example, `Simson's theorem` as in Fig. 2.

Currently, the system provides simple query services for users who may input search commands and view the results using keywords and relations. The queries through relations work at the level of knowledge objects and category objects. This means that the queries need to be described by using the *objectIDs* of the objects but not simple natural texts. We explain the commands for queries below.

– `keyWords[`$word_1, \ldots, word_n$`]` returns the set of knowledge objects and category objects with keywords $word_1$ and ... and $word_n$.
– `relation[*,`$objectID$`,`$relationType$`]` returns the set of knowledge objects that are each in the relation of *relationType* to the knowledge object identified by *objectID*;
– `relation[`$objectID$`,*,`$relationType$`]` returns the set of knowledge objects such that the knowledge object identified by *objectID* is in the relation of *relationType* with each of them.

The relations among knowledge objects and category objects are very important for structuring the knowledge base. One way to acquire the relations is manual annotation through reference to the *objectIDs* of the corresponding objects. We



Electronic Geometry Textbook: GBase\
File Query Construct Tools Intelligence Context
New Save Category Definition Axiom Asse
Geometry Revisited
Construct a theorem
Select the relations
OK Cancel View

Textbook : Geometry Revisited
Introduction : Preface
Chapter : Points and Lines Conn
Section : Basic concepts
Section : The extended Law of
Section : Ceva's theorem
Section : Points of interest
Section : The incircle and exci
Section : The Steiner-Lehmus
Section : The orthic triangle
Section : The medial triangle a
Section : The nine point circle
Section : Pedal triangles
Chapter : Some Properties of Cir
Section : The power of a point
Section : The radical axis of tw
Section : Coaxal circles
Section : More on the altitudes
Section : Simson lines
Theorem : Simson theorem
Proof : Simson theorem
Definition : Simson line
Remark : Simson
Section : Ptolemy's theorem a
Section : More on Simson line
Section : The Butterfly
Section : Morley's theorem
Chapter : Collinearity and Concu

Name: Simson theorem(english);西门松定理(中

Formal Representation

Hypothesis: A:=point();B:=point();C:=point();D:=point();incident(D,circumcircle(triangle(A,B,C)))

Relation: ⇔

Conlusion: collinear(foot(D,line(A,B)),foot(D,line(A,C)),foot(D,line(B,C)))

Key Words: collinear;foot;circumcircle;

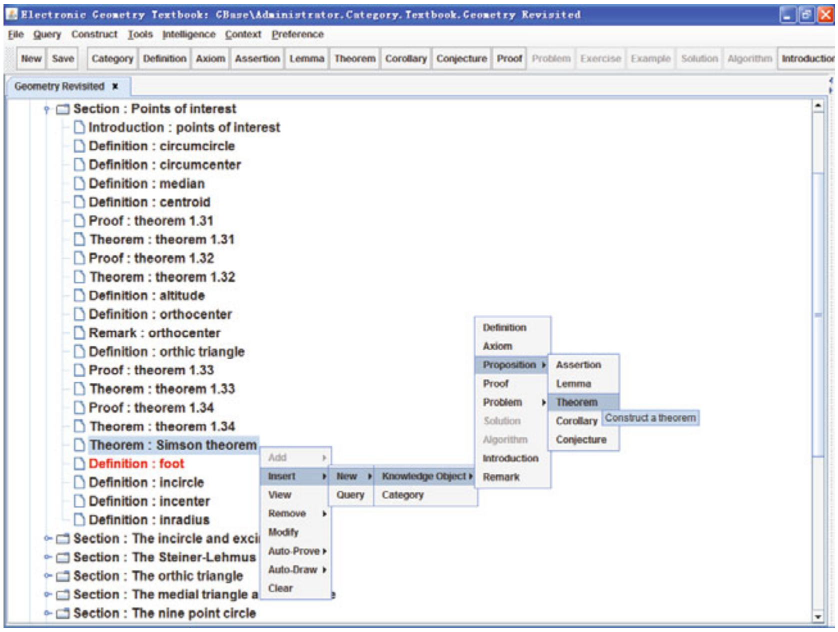
✓ [Administrator] Definition : point.default ⇒contextOf⇒ [Administrator] Theorem : Simson theorem.default
✓ [Administrator] Definition : circumcircle.default ⇒contextOf⇒ [Administrator] Theorem : Simson theorem.def
✓ [Administrator] Definition : triangle.default ⇒contextOf⇒ [Administrator] Theorem : Simson theorem.default
✓ [Administrator] Definition : foot.default ⇒contextOf⇒ [Administrator] Theorem : Simson theorem.default
✓ [Administrator] Definition : line.default ⇒contextOf⇒ [Administrator] Theorem : Simson theorem.default

Fig. 3. Discovering the Context relations automatically

have implemented another way that the system can automatically discover the Context and Inheritance relations by matching the concept declarations with the instances used in the formal representations of knowledge objects. For example, in the process of constructing `Simson's theorem`, the definitions of `point`, `line`, `foot`, `triangle`, `circumcircle` are automatically found that provide the context for `Simson's theorem`. The system will list the relations discovered for the user to select. (see Fig. 3)

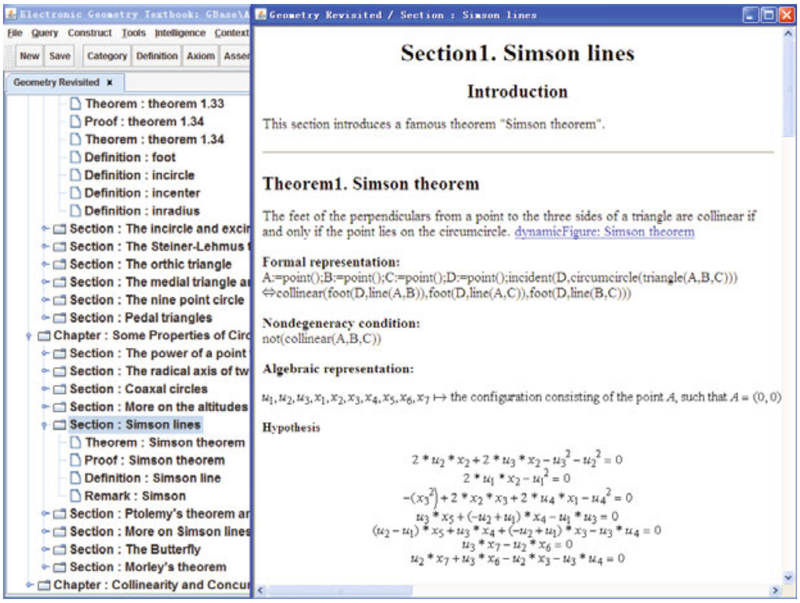
3.2 User Interface of Electronic Geometry Textbook

With the textbook knowledge base created, a user interface is implemented for users to construct dynamic textbooks, which are rendered as trees. The category objects are rendered as branch nodes and knowledge objects are rendered as tree leaves. Via dialogs, one can construct textbooks interactively by adding, inserting, removing, modifying, and rearranging the category objects and knowledge objects, and annotating their relations one by one. These objects may be newly created in, or fetched from, the knowledge base. While performing these manipulations, the system can check automatically whether the structure of the current textbook is consistent. The user may be given tips if it is inconsistent and the textbook should be restructured until it becomes consistent. For example, if one places `Simson's theorem` before the definition of `foot` (which provides the context for `Simson's theorem`), then the system will highlight the node of `foot`. (see Fig. 4)


Electronic Geometry Textbook: GBase\Administrator.Category.Textbook.Geometry Revisited
File Query Construct Tools Intelligence Context Preference
New Save Category Definition Axiom Assertion Lemma Theorem Corollary Conjecture Proof Problem Exercise Example Solution Algorithm Introduction
Geometry Revisited
Add
Insert
View
Remove
Modify
Auto-Prove
Auto-Draw
Clear
New
Query
Knowledge Object
Category
Definition
Axiom
Proposition
Proof
Problem
Solution
Algorithm
Introduction
Remark
Assertion
Lemma
Theorem
Corollary
Conjecture
Construct a theorem

Section : Points of interest
Introduction : points of interest
Definition : circumcircle
Definition : circumcenter
Definition : median
Definition : centroid
Proof : theorem 1.31
Theorem : theorem 1.31
Proof : theorem 1.32
Theorem : theorem 1.32
Definition : altitude
Definition : orthocenter
Remark : orthocenter
Definition : orthic triangle
Proof : theorem 1.33
Theorem : theorem 1.33
Proof : theorem 1.34
Theorem : theorem 1.34
Theorem : Simson theorem
Definition : foot
Definition : incircle
Definition : incenter
Definition : inradius
Section : The incircle and exci
Section : The Steiner-Lehmus
Section : The orthic triangle
Section : The medial triangle a
Section : The nine point circle

Fig. 4. The definition of `foot` is highlighted when `Simson's theorem` is placed before it


Electronic Geometry Textbook: GBase\A
File Query Construct Tools Intelligence Context
New Save Category Definition Axiom Asser
Geometry Revisited
Geometry Revisited / Section : Simson lines

- Theorem : theorem 1.33
- Proof : theorem 1.34
- Theorem : theorem 1.34
- Definition : foot
- Definition : incircle
- Definition : incenter
- Definition : inradius
- Section : The incircle and excir
- Section : The Steiner-Lehmus t
- Section : The orthic triangle
- Section : The medial triangle ar
- Section : The nine point circle
- Section : Pedal triangles
- Chapter : Some Properties of Circ
- Section : The power of a point
- Section : The radical axis of tw
- Section : Coaxal circles
- Section : More on the altitudes
- Section : Simson lines
- Theorem : Simson theorem
- Proof : Simson theorem
- Definition : Simson line
- Remark : Simson
- Section : Ptolemy's theorem ar
- Section : More on Simson lines
- Section : The Butterfly
- Section : Morley's theorem
- Chapter : Collinearity and Concur

Section1. Simson lines

Introduction

This section introduces a famous theorem "Simson theorem".

Theorem1. Simson theorem

The feet of the perpendiculars from a point to the three sides of a triangle are collinear if and only if the point lies on the circumcircle. dynamicFigure: Simson theorem

Formal representation:
A:=point();B:=point();C:=point();D:=point();incident(D,circumcircle(triangle(A,B,C))) ⇔collinear(foot(D,line(A,B)),foot(D,line(A,C)),foot(D,line(B,C)))

Nondegeneracy condition:
not(collinear(A,B,C))

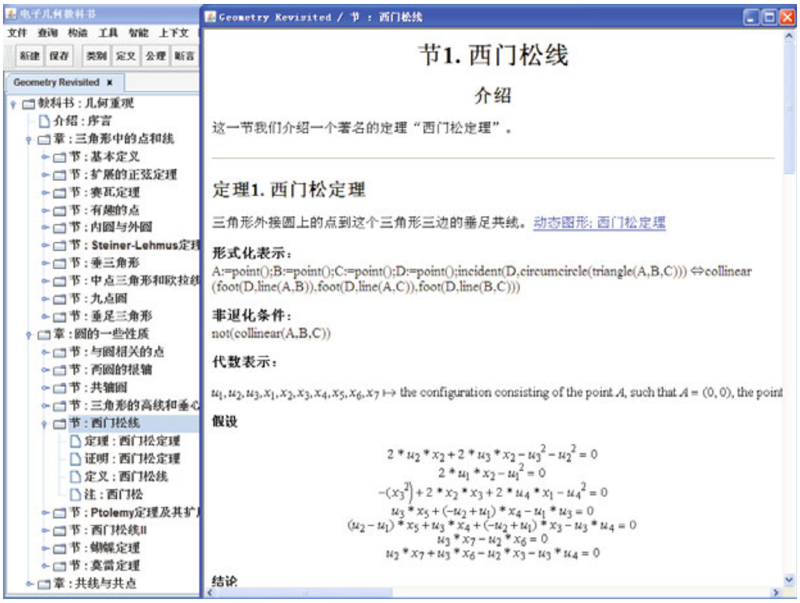
Algebraic representation:

$u_1,u_2,u_3,x_1,x_2,x_3,x_4,x_5,x_6,x_7 \mapsto$ the configuration consisting of the point A, such that $A=(0,0)$

Hypothesis

$$2*u_2*x_2+2*u_3*x_2-u_3^2-u_2^2=0$$
$$2*u_1*x_2-u_1^2=0$$
$$-(x_3^2)+2*x_2*x_3+2*u_4*x_1-u_4^2=0$$
$$u_3*x_5+(-u_2+u_1)*x_4-u_1*u_3=0$$
$$(u_2-u_1)*x_5+u_3*x_4+(-u_2+u_1)*x_3-u_3*u_4=0$$
$$u_3*x_7-u_2*x_6=0$$
$$u_2*x_7+u_3*x_6-u_2*x_3-u_3*u_4=0$$

Fig. 5. Rendering the Section "Simson lines" in English


电子几何教科书
文件 查询 构造 工具 智能 上下文
新建 保存 类别 定义 公理 断言
Geometry Revisited
Geometry Revisited / 节 : 西门松线

- 教科书 : 几何重观
- 介绍 : 序言
- 章 : 三角形中的点和线
- 节 : 基本定义
- 节 : 扩展的正弦定理
- 节 : 赛瓦定理
- 节 : 有趣的点
- 节 : 内圆与外圆
- 节 : Steiner-Lehmus定理
- 节 : 垂三角形
- 节 : 中点三角形和欧拉线
- 节 : 九点圆
- 节 : 垂足三角形
- 章 : 圆的一些性质
- 节 : 与圆相关的点
- 节 : 两圆的根轴
- 节 : 共轴圆
- 节 : 三角形的高线和垂心
- 节 : 西门松线
- 定理 : 西门松定理
- 证明 : 西门松定理
- 定义 : 西门松线
- 注 : 西门松
- 节 : Ptolemy定理及其扩
- 节 : 西门松线II
- 节 : 蝴蝶定理
- 节 : 莫雷定理
- 章 : 共线与共点

节1. 西门松线

介绍

这一节我们介绍一个著名的定理“西门松定理”。

定理1. 西门松定理

三角形外接圆上的点到这个三角形三边的垂足共线。动态图形: 西门松定理

形式化表示:
A:=point();B:=point();C:=point();D:=point();incident(D,circumcircle(triangle(A,B,C))) ⇔collinear (foot(D,line(A,B)),foot(D,line(A,C)),foot(D,line(B,C)))

非退化条件:
not(collinear(A,B,C))

代数表示:

$u_1,u_2,u_3,x_1,x_2,x_3,x_4,x_5,x_6,x_7 \mapsto$ the configuration consisting of the point A, such that $A=(0,0)$, the point

假设

$$2*u_2*x_2+2*u_3*x_2-u_3^2-u_2^2=0$$
$$2*u_1*x_2-u_1^2=0$$
$$-(x_3^2)+2*x_2*x_3+2*u_4*x_1-u_4^2=0$$
$$u_3*x_5+(-u_2+u_1)*x_4-u_1*u_3=0$$
$$(u_2-u_1)*x_5+u_3*x_4+(-u_2+u_1)*x_3-u_3*u_4=0$$
$$u_3*x_7-u_2*x_6=0$$
$$u_2*x_7+u_3*x_6-u_2*x_3-u_3*u_4=0$$

结论

Fig. 6. Rendering the Section "Simson lines" in Chinese

3.3 Presentation of Geometric Textbook Knowledge

For a textbook once constructed, it is necessary to provide a view that presents the knowledge objects and category objects in readable styles. From the textbook (or part of one), the system automatically generates corresponding XML documents by assembling the data of the selected objects and renders them (both in English and Chinese) via JDesktop Integration Components (JDIC [14]), which provide Java applications with access to functionalities and facilities furnished by the native desktop (see Figs. 5, 6). The generated XML documents can easily be styled and transformed into other document formats (MathDox [17], or PDF, etc.) by using XSLT.

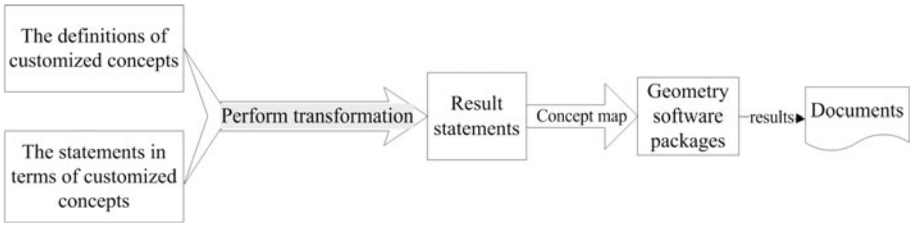
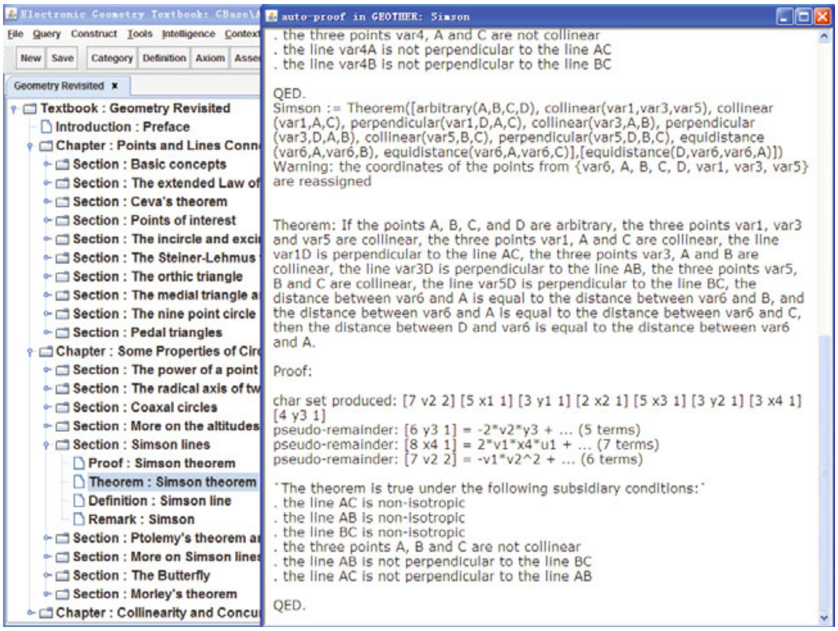

The definitions of customized concepts
The statements in terms of customized concepts
Perform transformation
Result statements
Concept map
Geometry software packages
results
Documents

Fig. 7. The workflow of communication with geometry software packages


Electronic Geometry Textbook: GBase\
File Query Construct Tools Intelligence Context
New Save Category Definition Axiom Asse
Geometry Revisited ×
auto-proof in GEOTHER: Simson

Textbook : Geometry Revisited
Introduction : Preface
Chapter : Points and Lines Conn
Section : Basic concepts
Section : The extended Law of
Section : Ceva's theorem
Section : Points of interest
Section : The incircle and exci
Section : The Steiner-Lehmus
Section : The orthic triangle
Section : The medial triangle a
Section : The nine point circle
Section : Pedal triangles
Chapter : Some Properties of Cir
Section : The power of a point
Section : The radical axis of tw
Section : Coaxal circles
Section : More on the altitudes
Section : Simson lines
Proof : Simson theorem
Theorem : Simson theorem
Definition : Simson line
Remark : Simson
Section : Ptolemy's theorem a
Section : More on Simson lines
Section : The Butterfly
Section : Morley's theorem
Chapter : Collinearity and Concu

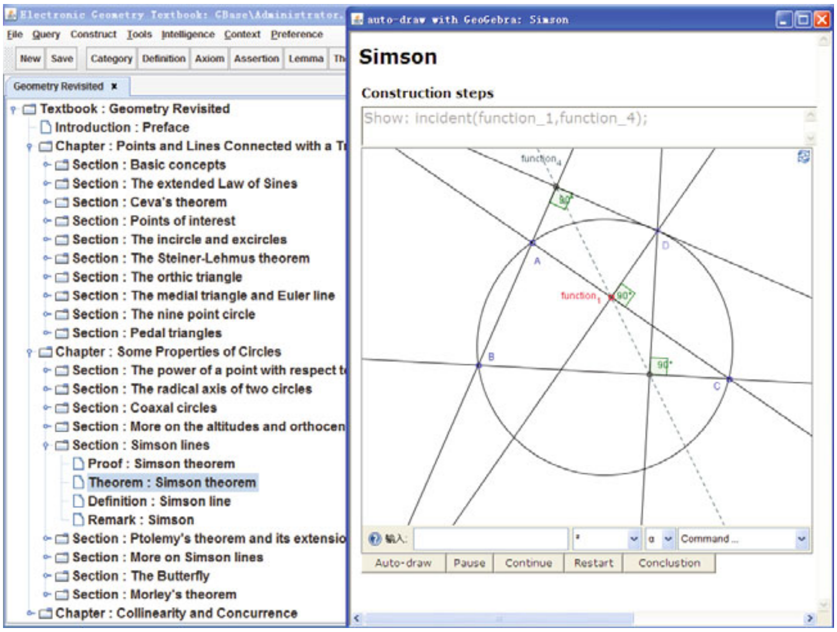
```
. the three points var4, A and C are not collinear
. the line var4A is not perpendicular to the line AC
. the line var4B is not perpendicular to the line BC

QED.
Simson := Theorem([arbitrary(A,B,C,D), collinear(var1,var3,var5), collinear
(var1,A,C), perpendicular(var1,D,A,C), collinear(var3,A,B), perpendicular
(var3,D,A,B), collinear(var5,B,C), perpendicular(var5,D,B,C), equidistance
(var6,A,var6,B), equidistance(var6,A,var6,C)],[equidistance(D,var6,var6,A)])
Warning: the coordinates of the points from {var6, A, B, C, D, var1, var3, var5}
are reassigned

Theorem: If the points A, B, C, and D are arbitrary, the three points var1, var3
and var5 are collinear, the three points var1, A and C are collinear, the line
var1D is perpendicular to the line AC, the three points var3, A and B are
collinear, the line var3D is perpendicular to the line AB, the three points var5,
B and C are collinear, the line var5D is perpendicular to the line BC, the
distance between var6 and A is equal to the distance between var6 and B, and
the distance between var6 and A is equal to the distance between var6 and C,
then the distance between D and var6 is equal to the distance between var6
and A.

Proof:

char set produced: [7 v2 2] [5 x1 1] [3 y1 1] [2 x2 1] [5 x3 1] [3 y2 1] [3 x4 1]
[4 y3 1]
pseudo-remainder: [6 y3 1] = -2*v2*y3 + ... (5 terms)
pseudo-remainder: [8 x4 1] = 2*v1*x4*u1 + ... (7 terms)
pseudo-remainder: [7 v2 2] = -v1*v2^2 + ... (6 terms)

`The theorem is true under the following subsidiary conditions:`
. the line AC is non-isotropic
. the line AB is non-isotropic
. the line BC is non-isotropic
. the three points A, B and C are not collinear
. the line AB is not perpendicular to the line BC
. the line AC is not perpendicular to the line AB

QED.
```

Fig. 8. `Simson's theorem` in the textbook is automatically proved by using GEOTHER

3.4 Automatic Problem Solving

As presented in Section 2.3, geometric statements of knowledge objects are formalized and represented by using customized geometric concepts. However, most geometry software tools only implement some of them. For communicating with the available stand-alone packages, it is indispensable to transform these statements into semantically equivalent ones employing the concepts that the target geometry software packages are able to identify and manipulate. Inspired by the idea of expression simplification in functional programming, we have implemented this transformation automatically by applying definitions of customized geometric concepts stored in the knowledge base. The process of communication with geometry software packages is diagrammed in Fig. 7.

We have implemented communication with GEOTHER for automated theorem proving (see Fig. 8) and with GeoGebra for drawing dynamic figures automatically (see Fig. 9).


Electronic Geometry Textbook: CBase\Administrator
File Query Construct Tools Intelligence Context Preference
New Save Category Definition Axiom Assertion Lemma
Geometry Revisited
auto-draw with GeoGebra: Simson
function4
function1
A
B
C
D
90°
输入:
Command ...
Auto-draw Pause Continue Restart Conclustion

Textbook : Geometry Revisited
- Introduction : Preface
- Chapter : Points and Lines Connected with a T
  - Section : Basic concepts
  - Section : The extended Law of Sines
  - Section : Ceva's theorem
  - Section : Points of interest
  - Section : The incircle and excircles
  - Section : The Steiner-Lehmus theorem
  - Section : The orthic triangle
  - Section : The medial triangle and Euler line
  - Section : The nine point circle
  - Section : Pedal triangles
- Chapter : Some Properties of Circles
  - Section : The power of a point with respect t
  - Section : The radical axis of two circles
  - Section : Coaxal circles
  - Section : More on the altitudes and orthocen
  - Section : Simson lines
    - Proof : Simson theorem
    - Theorem : Simson theorem
    - Definition : Simson line
    - Remark : Simson
  - Section : Ptolemy's theorem and its extensio
  - Section : More on Simson lines
  - Section : The Butterfly
  - Section : Morley's theorem
- Chapter : Collinearity and Concurrence

Simson

Construction steps

Show: incident(function_1,function_4);

Fig. 9. The dynamic figure of `Simson's theorem` is automatically drawn by using GeoGebra

4 Conclusion and Future Work

This paper describes the design principles and, briefly, the technical realization of the first version of Electronic Geometry Textbook. The system provides an integrated environment for users to manage and share textbook knowledge objects, construct dynamic geometry textbooks interactively and efficiently, and publish styled geometric documents easily. The knowledge objects encapsulate multiple forms of knowledge data for different applications, such as presentation in natural languages, processing by selected external geometry software packages for automated theorem proving and dynamic figure drawing, etc. The textbooks constructed can be manipulated easily with automatic consistency-checking in real time. The system can be viewed as a geometry-textbook-authoring assistant.

Currently, the development of Electronic Geometry Textbook is still at an early stage. It is far from its ultimate goal of seeing that the dynamic geometry textbooks constructed can be used in practice with students. For instance, communications with geometry software packages lack interactions with users. Aside from a series of experiments on the system in the near future, we are preparing to explore approaches to the design and development of interactive exercises in geometry and to enhance the usability of the textbooks.

Acknowledgments. The author wishes to thank Professor Dongming Wang for his kind supervision and Professor Arjeh M. Cohen for inviting him to visit and work at the DAM group of Technische Universiteit Eindhoven. His work has benefited from communication with Hans Cuypers and other members of the DAM group during his stay in Eindhoven. The author also wishes to thank the referees for their insightful comments and Patrick Ion for the copy-editing which have helped bring this paper to its present form. It has been supported by the Chinese National Key Basic Research (973) Project 2005CB321901/2 and the SKLSDE Open Fund BUAA-SKLSDE-09KF-01.

References

1. Activemath home, `http://www.activemath.org/`
2. Cabri home, `http://www.cabri.com/`
3. Chen, X., Huang, Y., Wang, D.: On the design and implementation of a geometric knowledge base. In: Kauers, M., Wu, M., Zeng, Z. (eds.) ADG 2008, pp. 62–78. East China Normal University, China (2008)
4. Chen, X., Wang, D.: Towards an electronic geometry textbook. In: Botana, F., Recio, T. (eds.) ADG 2006. LNCS (LNAI), vol. 4869, pp. 1–23. Springer, Heidelberg (2007)
5. Chen, X., Wang, D.: Management of geometric knowledge in textbooks. Preprint, LMIB – Beihang University, China (November 2009)
6. Chou, S., Gao, X.: Automated reasoning in geometry. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, ch. 11, vol. I, pp. 709–749. Elsevier, Amsterdam (2001)
7. Cinderella home, `http://www.cinderella.de/`
8. Gclc home, `http://www.emis.de/misc/software/gclc/`
9. Geogebra home, `http://www.geogebra.org/cms/`
10. Geometry expert home, `http://www.mmrc.iss.ac.cn/gex/`
11. Geoproof home, `http://home.gna.org/geoproof/`
12. Geother home, `http://www-calfor.lip6.fr/~wang/GEOTHER/index.html`
13. Intergeo home, `http://i2geo.net/xwiki/bin/view/Main`
14. Jdic home, `https://jdic.dev.java.net/`

15. Kamareddine, F., Maarek, M., Retel, K., Wells, J.B.: Narrative structure of mathematical texts. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, pp. 296–312. Springer, Heidelberg (2007)
16. Leactivemath home, http://www.leactivemath.org/
17. Mathdox home, http://www.mathdox.org/
18. Mathdox formula editor, http://www.mathdox.org/formulaeditor/
19. Sketchpad home, http://www.dynamicgeometry.com/

An OpenMath Content Dictionary for Tensor Concepts

Joseph B. Collins

Naval Research Laboratory
4555 Overlook Ave, SW
Washington, DC 20375-5337

Abstract. We introduce a new OpenMath content dictionary named "tensor1" containing symbols for the expression of tensor formulas. These symbols support the expression of non-Cartesian coordinates and invariant, multilinear expressions in the context of coordinate transformations. While current OpenMath symbols support the expression of linear algebra formulas using matrices and vectors, we find that there is an underlying assumption of Cartesian, or *standard*, coordinates that makes the expression of general tensor formulas difficult, if not impossible. In introducing these new OpenMath symbols for the expression of tensor formulas, we attempt to maintain, as much as possible, consistency with prior OpenMath symbol definitions for linear algebra.

1 Introduction

In scientific and engineering disciplines there are many uses of tensor notation. A principal reason for the need for tensors is that the laws of physics are best formulated as tensor equations. Tensor equations are used for two reasons: first, the physical laws of greatest interest are those that may be stated in a form that is independent of the choice of coordinates, and secondly; expressing the laws of physics differently for each choice of coordinates becomes cumbersome to maintain. While from a theoretical perspective it is desirable to be able to express the laws of physics in a form that is independent of coordinate frame, the application of those laws to the prediction of the dynamics of physical objects requires that we do ultimately specify the values in some coordinate frame. Much of this discussion might be moot were scientists and engineers to confine themselves to one frame, e.g., Cartesian coordinates, but such is not the case. Non-Cartesian coordinates are useful for curved geometries and because closed form solutions to applied models in classical physics, which rely on the separation of variables method of solving partial differential equations, often exist in them. For example, the Laplace equation is separable in thirteen coordinate systems [1]. One may also take as a definition of need that these concepts are included in the ISO standards defining the necessary mathematical symbols in the International System (SI) of Quantities and Units [2].

While we specify the new OpenMath symbols for tensor concepts, we attempt to maintain consistency with pre-existing OpenMath symbols [3]. The symbols within OpenMath content dictionaries support the expression of a wealth of mathematical concepts. Determining whether or not additional symbols are needed requires consideration based upon both necessity and convenience. Advancing new symbols using arguments based upon mathematical necessity only implies that a proof is at hand showing that a particular concept cannot be expressed using existing OpenMath symbols. Since such proofs would be difficult, if not impossible, in practice, convincing arguments for new symbols are more likely to be made based on a combination of practical economy, practical necessity, and convenience: this is certainly the case here. The symbols we introduce are motivated by the need for easily and directly capturing the relevant semantics in the expression of tensor formulas.

OpenMath symbols exist for the specification of matrices and vectors. These are documented in the content dictionaries linalg1, linalg2, linalg3, linalg4, linalg5, and two dictionaries named linalg6. Within these dictionaries there are two representations: one for row vectors and one for column vectors, with the row representation being labeled "official" in preference to the column representation. In the row vector representation a matrix is a column of rows, and in the column vector representation it is a row of columns. We find that the two representations, i.e., the row representation of vectors and the column representation of vectors, appear to be alternative, equivalent representations, related by a transpose operation, rather than dual representations, such as vectors and covectors where row and column representations of a vector are related via a general metric tensor. We also find, from the few examples given and from the general lack of reference to bases, that the row and column vector semantics appear to assume use of the *standard*, or Cartesian, basis only, and, in particular, with a simple Euclidean metric. For example, the scalar product is given as $\mathbf{u}\cdot\mathbf{v} = \sum_i u_i v_i$. In the row representation, the vector components resulting from a matrix-vector multiplication appear as the results of scalar products between matrix rows and a row vector, i.e., a scalar product takes as its arguments two vectors from the same vector space. Given these observations, it is not clear that in using these existing representations it is easy, or even possible, to express the semantics of tensors as they are typically used by engineers and scientists. For these reasons we introduce symbols that are expressly to be used for specifying tensor formulas.

2 Tensor Review

To motivate our choice of OpenMath symbols for specifying tensors, we briefly review some tensor basics. By definition, a tensor is a multilinear mapping that maps vectors and covectors to a scalar field. A tensor is itself an element of the space defined by a tensor product of covector and vector spaces. Among scientists and engineers, tensor formulas are commonly written using their indexed *components*.

2.1 Coordinate Frames

To begin the discussion, we note that an arbitrary point in n-dimensional space, $\mathbf{R}^n$, is typically specified by its n Cartesian, or *standard*, coordinates, x^i. The point's *position vector*, is then written as $\mathbf{r} = \sum_i x^i \mathbf{e}_i$, where the $\mathbf{e}_i$ are orthonormal Cartesian basis vectors and are constant, i.e., not a function of the coordinates, for a given Cartesian frame. Using the original Cartesian frame, alternative coordinates may be defined as functions of the Cartesian coordinates in the original frame, e.g., $x'^i = x'^i(x^1, ..., x^n)$, which may be nonlinear in the x^i.

Spatial coordinates are sometimes expressed as indexed quantities, such as (x^1, x^2, x^3), or having individual names, such as (x, y, z). In presentation, different kinds of indexes may appear much the same, but in content markup we must be more discriminating. For example, vector components are one kind of indexed quantity. It would be a mistake, however, to consider the tuple of spatial coordinates to be a vector in the general case. While it may seem to be a vector in Cartesian coordinates, i.e., a position vector, this is not the case in, for example, polar coordinates. For general coordinates the vector addition of coordinate position tuples does not appear to have a defined meaning, i.e., the meaning of

coordinates(Point1) + coordinates(Point2) = coordinates(Point3)

is not preserved under general cordinate transformation.

Considering this, the most we should say is that the variables describing the coordinates of an arbitrary point in a space comprise an n-tuple. This appears to be similar to the notion of an OpenMath *context* [4], i.e., an n-tuple of variables. Consequently, while we need to represent x^i, it is inappropriate to do this using the vector_selector symbol, the vector component accessor defined in the OpenMath linalg1 content dictionary. For this reason we introduce the *tuple* and *tuple_selector* symbols. The symbol, tuple, is an n-ary function that returns an n-tuple of its arguments in the order that they are presented. The symbol, tuple_selector, takes two arguments, an n-tuple, and an index, a natural number less than or equal to n, and returns the indexed element.

Since the Cartesian frame is most often used, including in the definition of coordinate transformations and the definition of non-Cartesian frames, we find it useful to have symbols to express the base concepts of Cartesian coordinates. We propose the symbol *Cartesian* which takes a single argument, a natural number, and returns the Cartesian coordinate, of a right-handed Cartesian frame, corresponding to the value of the argument. The standard representation of Cartesian 3-space may then be represented by either

tuple(x, y, z) = tuple(Cartesian(1), Cartesian(2), Cartesian(3))

or as

tuple_selector(i, **x**) = Cartesian(i).

Coordinate transformations may then be defined as functions on the Cartesian coordinates.

The full meaning of the Cartesian coordinate variables comes from their combination with the basis vectors for the Cartesian frame. The commonly used orthonormal basis vectors for the Cartesian frame are given by the symbol *unit_Cartesian*, i.e., unit_Cartesian takes a single natural number as its argument and returns the corresponding unit vector, say, $\mathbf{e}_i$. Other representations are easily assigned, such as

$$\text{tuple}(\hat{\imath}, \hat{\jmath}, \hat{k}) = \text{tuple}(\text{unit\_Cartesian}(1), \text{unit\_Cartesian}(2), \text{unit\_Cartesian}(3)).$$

Basis vectors, $\mathbf{g}_i$, for transformed coordinates, x'^i, are given by

$$\mathbf{g}_i = \sum_j \frac{\partial x^j}{\partial x'^i} \mathbf{e}_j.$$

2.2 Vectors and Covectors

To describe tensors we must also give prior description to vectors and covectors. A vector, $\mathbf{v}$, may be specified by components v^i with respect to an arbitrary, ordered, vector space basis, $(\mathbf{g}_1, ..., \mathbf{g}_n)$, as $\mathbf{v} = \sum_i v^i \mathbf{g}_i$. These basis vectors, $\mathbf{g}_i$, are generally the tangent vectors with respect to the spatial coordinates, e.g., x^i. In curvilinear coordinates these general basis vectors are clearly functions of the coordinates. A dual, covector space may be defined relative to a given vector space. A dual space is defined as a set of linear functionals on the vector space and is spanned by a set of basis elements, $(\mathbf{g}^1, ..., \mathbf{g}^n)$, such that $\mathbf{g}^i(\mathbf{g}_j) = \delta^i{}_j$, where $\delta^i{}_j$ is the Kronecker tensor. The symbol, *Kronecker_tensor*, has components, $\delta^i{}_j$, equal to one when $i = j$ and zero otherwise.

The presentation of the indexes, either raised or lowered, on basis vectors, basis covectors, vector components, or covector components, generally indicates how the components transform. With a transformation of coordinates, indexed tensor quantities transform either *covariantly*, as do the basis vectors, $\mathbf{g}_i$, or they transform *contravariantly*, as do vector components, v^i, or, for example, coordinate differentials, dx^i. Transforming from coordinates x^i to coordinates x'^i, the covariant transformation is defined by the transformation of the basis vectors:

$$\mathbf{g}'_i = \sum_k \frac{\partial x^k}{\partial x'^i} \mathbf{g}_k.$$

The contravariant transformation of a vector's components is given by:

$$v'^l = \sum_k \frac{\partial x'^l}{\partial x^k} v^k.$$

The covariant transformation of the components of a covector, u, is given by

$$u'_j = \sum_k \frac{\partial x^k}{\partial x'^j} u_k.$$

Vectors whose components transform contravariantly, and their covectors, whose components transform covariantly, are tensors. Many, but not all, vector quantities are tensors. For example, the coordinates themselves, x^i, are referred to as the components of a *position vector* (in Cartesian coordinates), $\mathbf{x}$ or $\mathbf{r}$, which is *not* a tensor. (We have already noted that the position vector in Cartesian coordinates, defined as a tuple of position coordinates, does not generally preserve its meaning after coordinate transformation).

In general, tensors may be created by tensor (outer) products of vectors and covectors, contracted products of tensors, and sums of tensors of the same order. Order one tensors are contravariant or covariant vectors, while order zero tensors are scalars. The order of a higher order tensor is just the necessary number of vectors and covectors multiplied together, using the tensor product, to create it.

For the purpose of describing tensor formulas in content markup, we introduce the OpenMath symbols *tensor_selector*, *contra_index*, and *covar_index*, which are applied to a natural number, returning the appropriate index. In standard tensor notation, a contravariant index is represented as a superscripted index and a covariant index is represented as a subscripted index. The contra_index and covar_index symbols are so named because characterizing the indexes of tensor quantities as being contravariant or covariant captures the semantics.

In engineering and scientific applications standard matrix-vector multiplication is consistent with tensor notation when interpreted as a matrix multiplying a column vector from the left, resulting in a column vector. Each of the components of the result are arrived at by applying the rows of the matrix to the column vector being multiplied. It is consistent with this common usage to identify the components of a column vector using the contravariant, superscripted index as a row index, and to identify the components of a row vector using the covariant, subscripted index as a column index. The matrix-vector multiplication is then represented as $u^i = \sum_j M^i{}_j v^j$. It is common in tensor notation to suppress the explicit summation in such an expression using the Einstein Summation Convention.

While it is common to implicitly assume the use of standard, or Cartesian coordinates, in which case the distinction between superscripts and subscripts appears superfluous, this is not so with tensor notation: a vector may be specified by its components relative to some general, non-Cartesian basis. As pointed out above, the basis vectors of an arbitrary, ordered basis of a vector space transform covariantly, hence they are indexed using the symbol, covar_index. Similarly, the basis covectors, derived from the same arbitrary, ordered basis of the vector space, transform contravariantly, hence they are indexed using the symbol, contra_index.

We introduce, then, the basis_selector symbol as a binary operator, taking as its arguments:

1) an ordered basis, a tuple of linearly independent vectors that spans some vector space;

2) either a covar_index or contra_index symbol applied to a natural number. The basis_selector operator returns a basis vector of the vector space when a covar_index symbol is passed and returns a basis covector of the dual vector space when a contra_index is passed.

2.3 Higher Order Tensors

To write expressions using tensor components, we use the symbol, tensor_selector. The tensor_selector symbol returns a scalar and takes three arguments:

1) a tensor;
2) a tuple of contra_index and covar_index symbols, and, finally;
3) a frame, an ordered set of basis vectors.

The sum total of indexes used, both contra_index's and covar_index's, must be the same as the order of the tensor. The contravariant and covariant indexes, taken together, are totally ordered, and refer to a matrix of tensor components, which are assumed to be in 'row-major' order, regardless of whether the indexes are contra_index's or covar_index's. By use of the term row-major order, we do not attribute any special meaning to whether an index is considered a row index or a column index, rather we merely mean that for the serial traversal of an arbitrarily dimensioned array used to store an indexed quantity, the rightmost index varies fastest. The assumption of this convention allows the unambiguous assignment of indexed matrix component values to indexed tensor components.

The scalar returned by tensor_selector is the tensor component. For example, the contravariant components of a vector are identified by applying tensor_selector to the vector and a contra_index. Components of higher order tensors are identified by use of multiple contra_index and covar_index symbols. The final argument, the frame, is necessary when one needs to specify a tensor expression that is dependent on the basis or on multiple bases, as in a transformation expression. As tensor formulas are commonly made without regard to basis, often no basis is required, and so any single, consistent basis is sufficient in this case, such as Cartesian. It is also suggested that a special value, called "unspecified", might be used.

A general tensor is usually indicated with a capital letter. Its coordinates may be represented using a sequence of contra_index and / or covar_index symbols. The tensor itself may be represented by taking the product

$$\mathbf{T} = \sum_{ij} T^{ij} \mathbf{g}_i \mathbf{g}_j .$$

The Einstein summation convention is normally implicitly applied to the product of two tensors whose components are represented with matching contravariant and covariant indexes. In content markup this summation should be explicit since there is otherwise no content markup to indicate the fact that these indexes are bound variables.

Finally, we define a couple more symbols for specific tensor and vector quantities. First, there is the *metric_tensor* which defines the geometric features of the vector space, such as length. Its components are represented as g_{ij}, a symmetric, non-degenerate, covariant, bilinear form defined by $(ds)^2 = g_{ij} dx'^i dx'^j$, where ds is the differential length element and dx'^i are the differential changes in spatial coordinates. This is a generalization of the simple Euclidean metric given by the scalar product. Covariant components and contravariant components of a vector, or row and column representations of a vector, are related by $v_i = \sum_j g_{ij} v^j$ and squared length, or squared norm, of a vector is $|\mathbf{v}|^2 = \sum_{ij} g_{ij} v^i v^j = \sum_j v_j v^j$.

Lastly, we define the *Levi-Civita* symbol, the so-called permutation tensor. It takes one argument, the dimension of the space. Its components may be indexed with the contra_index and / or covar_index symbols.

2.4 Conclusion

We have introduced a number of OpenMath symbols for the expression of tensor formulas. They are tuple, tuple_selector, Cartesian, unit_Cartesian, Kronecker_tensor, basis_selector, tensor_selector, contra_index, covar_index, metric_tensor, and Levi-Civita. Using the tuple, tuple_selector, Cartesian, and unit_Cartesian symbols we can build finite dimensioned Cartesian frames and define differentiable coordinate transformations to define other frames. Using Kronecker_tensor, basis_selector, tensor_selector, contra_index, and covar_index, we can define tensor spaces on those frames, assign values to tensor components, and write tensor formulas. The formulas may be within a single frame or between frames. Finally, with the metric tensor we can specify non-Euclidean metrics and using the Levi-Civita symbol we can express vector cross products and the curl operation in vector component form. These symbols are being submitted as a content dictionary named tensor1 to the online OpenMath repository.

Many thanks to Weiqing Gu at Naval Research Lab for several conversations regarding tensors.

References

1. Morse, P.M., Feshbach, H.: Methods of Theoretical Physics, Part I, pp. 655–666. McGraw-Hill, New York (1953)
2. Technical Committee ISO/TC 12, Quantities and units: Part 2: Mathematical signs and symbols to be used in the natural sciences and technology, Draft International Standard ISO/DIS 80000-2, International Organization for Standardization (2008)
3. Buswell, S., Caprotti, O., Carlisle, D.P., Dewar, M.C., Gäetano, M., Kohlhase, M.: The OpenMath Standard 2.0 (2004), `http://www.openmath.org`
4. Kohlhase, M., Rabe, F.: Semantics of OpenMath and MathML3. In: 22nd OpenMath Workshop Editor: James H Davenport, July 9. Grand Bend Ontario (University of Bath Press and the OpenMath Society) (2009), ISBN: 978-1-86197-172-2

On Duplication in Mathematical Repositories

Adam Grabowski[1] and Christoph Schwarzweller[2]

[1] Institute of Mathematics, University of Białystok
ul. Akademicka 2, 15-267 Białystok, Poland
adam@math.uwb.edu.pl

[2] Department of Computer Science, University of Gdańsk
ul. Wita Stwosza 57, 80-952 Gdańsk, Poland
schwarzw@inf.ug.edu.pl

Abstract. Building a repository of proof-checked mathematical knowledge is without any doubt a lot of work, and besides the actual formalization process there is also the task of maintaining the repository. Thus it seems obvious to keep a repository as small as possible, in particular each piece of mathematical knowledge should be formalized only once.

In this paper, however, we claim that it might be reasonable or even necessary to duplicate knowledge in a mathematical repository. We analyze different situations and reasons for doing so, provide a number of examples supporting our thesis and discuss some implications for building mathematical repositories.

1 Introduction

Mathematical knowledge management aims at providing tools and infrastructure supporting the organization, development, and teaching of mathematics using modern techniques provided by computers. Consequently, large repositories of mathematical knowledge are of major interest because they provide users with a data base of — verified — mathematical knowledge. We emphasize the fact that a repository should contain verified knowledge only together with the corresponding proofs. We believe that (machine-checked or -checkable) proofs necessarily belong to each theorem and therefore are an essential part of a repository.

However, mathematical repositories should be more than collections of theorems and corresponding proofs accomplished by a prover or proof checker. The overall goal here is not only stating and proving a theorem — though this remains an important and challenging part — but also presenting definitions and theorems so that the "natural" mathematical build-up remains visible. Theories and their interconnections should be available, so that further development of the repository can rely on existing formalizations. Being not trivial as such, this becomes even harder to assure for an open repository with a large number of authors.

In this paper we deal with yet another organizational aspect of building mathematical repositories: the duplication of knowledge, by which we mean that a repository includes redundant knowledge. At first glance this may look unacceptable or at least unnecessary. Why should one include — and hence formalize

— the same thing more than once? A closer inspection, however, shows that mathematical redundancy may occur in different non-trivial facets: Different proofs of a theorem may exist or different versions of a theorem formulated in a different context. Sometimes we even have different representations of the same mathematical object serving for different purposes.

From the mathematical point of view this is not only harmless but also desirable; it is part of the mathematical progress that theorems and definitions change and evolve. In mathematical repositories, however, each duplication of knowledge causes an additional amount of work. In this paper we analyze miscellaneous situations and reasons why there could — and should — be at least some redundancy in mathematical repositories. These situations range from the above mentioned duplication of proofs, theorems and representations to the problem of generalizing knowledge. Even technical reasons due to the progress of a repository may lead to duplication of knowledge.

2 Different Proofs of a Theorem

In the following we present different proofs of the Chinese Remainder Theorem (CRT) and briefly sum up the discussion from [Sch09]. The "standard" version of the CRT reads as follows.

Theorem 1. Let $m_1, m_2, \ldots, m_r$ be positive integers such that m_i and m_j are relatively prime for $i \neq j$. Let $m = m_1 m_2 \cdots m_r$ and let $u_1, u_2, \ldots, u_r$ be integers. Then there exists exactly one integer u with

$$0 \leq u < m \text{ and } u \equiv u_i \bmod m_i \text{ for all } 1 \leq i \leq r. \qquad \diamond$$

We consider three different proofs of the theorem and discuss their relevance to be included in mathematical repositories. It is very easy to show, that there exists at most one such integer u; in the following proofs we therefore focus on the existence of u. The proofs are taken from [Knu97].

First proof: Suppose integer u runs through the m values $0 \leq u < m$. Then $(u \bmod m_1, \ldots, u \bmod m_r)$ also runs through m different values, because the system of congruences has at most one solution. Because there are exactly $m_1 m_2 \cdots m_r = m$ different tuples $(v_1, \ldots, v_r)$ with $0 \leq v_i < m_i$, every tuple occurs exactly once, and hence for one of those we have $(u \bmod m_1, \ldots, u \bmod m_r) = (u_1, \ldots, u_r)$. $\diamond$

This proof is pretty elegant and uses a variant of the pigeon hole principle: If we pack m items without repetition to m buckets, then we must have exactly one item in each bucket. It is therefore valuable to include this proof in a repository for didactic or aesthetic reasons. On the other hand the proof is non-constructive, so that it gives no hints to find the value of u — besides the rather valueless "Try and check all possibilities, one will fit". A constructive proof, however, can easily be given:

Second proof: We can find integers M_i for $1 \leq i \leq r$ with

$$M_i \equiv 1 \bmod m_i \text{ and } M_j \equiv 0 \bmod m_i \text{ for } j \neq i.$$

Because m_i and m/m_i are relatively prime, we can take for example

$$M_i = (m/m_i)^{\varphi(m_i)},$$

where φ denotes the Euler function. Now,

$$u = (u_1 M_1 + u_2 M_2 + \cdots + u_r M_r) \bmod m$$

has the desired properties. ◇

This proof uses far more evolved mathematical notations — namely Euler's function — and for that reason may also be considered more interesting than the first one. Formalization requires the use of Euler's function[1] which may cause some preliminary work. From a computer science point of view, however, the proof has two disadvantages. First, it is not easy to compute Euler's function; in general one has to decompose the moduli m_i into their prime factors. Second, the M_i being multiples of m/m_i are large numbers, so that a better method for computing u is highly desirable. Such a method has indeed been found by H. Garner.

Third proof: Because we have $\gcd(m_i, m_j) = 1$ for $i \neq j$ we can find integers c_{ij} for $1 \leq i < j \leq r$ with

$$c_{ij} m_i \equiv 1 \bmod m_j$$

by applying the extended Euclidean algorithm to m_i and m_j. Now taking

$$\begin{aligned}
v_1 &:= u_1 \bmod m_1\\
v_2 &:= (u_2 - v_1)c_{12} \bmod m_2\\
v_3 &:= ((u_3 - v_1)c_{13} - v_2)c_{23} \bmod m_3\\
&\vdots\\
v_r &:= (\ldots((u_r - v_1)c_{1r} - v_2)c_{2r} - \cdots - v_{r-1})c_{(r-1)r} \bmod m_r
\end{aligned}$$

and then setting

$$u := v_r m_{r-1} \cdots m_2 m_1 + \cdots + v_3 m_2 m_1 + v_2 m_1 + v_1$$

we get the desired integer u. ◇

When constructing the v_i the application of the modulo operation in each step ensures that the occurring values remain small. The proof is far more technical than the others in constructing $\binom{r}{2} + r$ additional constants, the v_i in addition being recursively defined. On the other hand, however, this proof includes an efficient method to compute the integer u from Theorem 1.

[1] Actually a mild modification of the proof works without Euler's function.

3 Different Versions of Theorems

There are quite a number of reasons why different versions of the same theorem exist and may be included in mathematical repositories. Besides mathematical issues we also identified reasons justified by formalization issues or the development of repositories itself. For illustration we again use the CRT as an example.

3.1 Restricted Versions

Theorems are not always shown with a proof assistant to be included in a repository in the first place: Maybe the main goal is to illustrate or test a new implemented proof technique or just to show that this special kind of mathematics can be handled within a particular system. In this case it is often sufficient — or simply easier — to prove a weaker or restricted version of the original theorem from the literature.

In HOL Light [Har10], for example, we find the following theorem.

```
# INTEGER_RULE
  '!a b u v:int. coprime(a,b) ==>
                          ?x. (x == u) (mod a) /\ (x == v) (mod b)';
```

This is a version of the CRT stating that in case of two moduli `a` and `b` only there exists a simultaneous solution `x` of the congruences. Similar versions have been shown with `hol98` ([Hur03]), the Coq proof assistant ([Mén10]) or Rewrite Rule Laboratory ([ZH92]).

From the viewpoint of mathematical repositories it is of course desirable to have included the full version of the theorem also. Can we, however, in this case easily set the restricted version aside? Note that the above theorem in HOL Light also serves as a rule for proving divisibility properties of the integers. Erasing the restricted version then means that the full version has to be used instead. It is hardly foreseeable whether this will work for all proofs relying on the restricted version. So, probably both the restricted and the full version belong to the repository.

3.2 Different Mathematical Versions

The most natural reason for different versions of theorems is that mathematicians often look at the same issue from different perspectives. The CRT presented in Section 2 deals with congruences over the integers: it states the existence of an integer solving a given system of congruences. Looking from a more algebraic point of view we see that the moduli m_i can be interpreted as describing the residue class rings $\mathcal{Z}_{m_i}$. The existence and uniqueness of the integer u from the CRT then gives rise to an isomorphism between rings [GG99]:

Theorem 2. Let $m_1, m_2, \ldots, m_r$ be positive integers such that m_i and m_j are relatively prime for $i \neq j$ and let $m = m_1 m_2 \cdots m_r$. Then we have the ring isomorphism

$$\mathcal{Z}_m \;\cong\; \mathcal{Z}_{m_0} \times \cdots \times \mathcal{Z}_{m_r}. \qquad \diamond$$

This version of the CRT has been formalized in `hol98` [Hur03]. Here we find a two-moduli version that in addition is restricted to multiplicative groups. Technically, the theorem states that for relatively prime moduli p and q the function $\lambda x.(x \bmod p, x \bmod q)$ is a group isomorphism between $\mathcal{Z}_{pq}$ and $\mathcal{Z}_p \times \mathcal{Z}_q$.

$$\begin{array}{l} \vdash \forall p, q. \\ \quad 1 < p \wedge 1 < q \wedge \mathbf{gcd}\ p\ q = 1 \Rightarrow \\ \quad (\lambda x.(x \bmod p, x \bmod q)) \in \\ \qquad \mathbf{group\_iso}\ (\mathbf{mult\_group}\ pq) \\ \qquad (\mathbf{prod\_group}\ (\mathbf{mult\_group}\ p)\ (\mathbf{mult\_group}\ q)) \end{array}$$

Note that, in contrast to Theorem 2, the isomorphism is part of the theorem itself and not hidden in the proof.

It is not easy to decide which version of the CRT may be better suited for inclusion in a mathematical repository. Theorem 2 looks more elegant and in some sense contains more information than Theorem 1: It does not state the existence of a special integer, but the equality of two mathematical structures. The proof of Theorem 2 uses the homomorphism theorem for rings and is therefore interesting for didactic reasons, too. On the other hand, Theorem 1 uses integers and congruences only, so that one needs less preliminaries to understand it. Theorem 1 and its proof also give more information than theorem 2 concerning computational issues[2] — at least if not the first proof only has been formalized.

3.3 Different Technical Versions

Another reason for different versions of a theorem may be originated in the mathematical repository itself. Here again open repositories play an important role: Different authors, hence different styles of formalizing and different kinds of mathematical understanding and preferences meet in one repository. So, it may happen that two authors formalize the same (mathematical) theorem, but choose a different formulation and/or a different proof. We call this technical versions.

Especially in evolving systems such versions may radically differ just because the system's language improved over the years. In the Mizar Mathematical Library, for example, we find the following CRT [Sch08]

```
theorem
for u being integer-yielding FinSequence,
    m being CR_Sequence st len u = len m
ex z being Integer
st 0 <= z & z < Product(m) & for i being natural number
   st i in dom u holds z,u.i are_congruent_mod m.i;
```

[2] To apply the homomorphism theorem in the proof of Theorem 2 one needs to show that the canonical homomorphism is a surjection with kernel (m). This sometimes is done by employing the extended Euclidean algorithm, so that this proof gives an algorithm, too.

Here, a CR_Sequence is a sequence of natural numbers, which are pair wise relatively prime. Note that this formulation of the theorem is very close to the textbook version of theorem 1.

In another Mizar article [Kon97], however, we find a different formulation of the CRT:

```
theorem :: WSIERP_1:44
len fp>=2 &
(for b,c st b in dom fp & c in dom fp & b<>c holds (fp.b gcd fp.c)=1)
implies for fr st len fr=len fp holds ex fr1 st (len fr1=len fp &
for b st b in dom fp holds (fp.b)*(fr1.b)+(fr.b)=(fp.1)*(fr1.1)+(fr.1));
```

In this version no attributes are used. The condition that the m_i are pair wise relatively prime is here stated explicitly using the gcd functor for natural numbers. Also the congruences are described arithmetically: $u \equiv u_i \bmod m_i$ means that there exists a x_i such that $u = u_i + x_i * m_i$, so the theorem basically states the existence of $x_1, \ldots, x_r$ instead of u.

Since the article has been written more than 10 years ago, a reason is hard to estimate. It may be that at the time of writing Mizar's attribute mechanism was not so far developed as today, i.e. the author reformulated the theorem in order to get it formalized at all. Another explanation for this version might be that the author when formalizing the CRT already had in mind a particular application and therefore chose a formulation better suited to prove the application.

In the Coq Proof Assistant [Coq10] the CRT has been proved for a bit vector representation of the integers [Mén10], though as a restricted version of Theorem 1 with two moduli a and b.

```
Theorem chinese_remaindering_theorem :
 forall a b x y : Z,
 gcdZ a b = 1%Z -> {z : Z | congruentZ z x a /\ congruentZ z y b}.
```

In fact this theorem and its proof are the result of rewriting a former proof of the CRT in Coq. So in Coq there exist two versions of the CRT — though the former one has been declared obsolete.

We see that in general the way authors use open systems to formalize theorems has a crucial impact on the formulation of a theorem, and may lead to different versions of the same theorem. Removing one — usually the older one — version is a dangerous task: In large repositories it is not clear whether all proofs relying on the deleted version can be easily changed to work with the other one. So often both versions remain in the repository.

4 Abstract and Concrete Mathematics

Practically every mathematical repository has a notion of groups, rings, fields and many more abstract structures. The advantage is obvious: A theorem shown in an abstract structure holds in every concretion of the structure also. This helps to keep a repository small: Even if concrete structures are defined there is no need to repeat theorems following from the abstract structure. If necessary in a proof one can just apply the theorem proved for the abstract structure.

Nevertheless authors tend to prove theorems again for the concrete case. We can observe this phenomenon in the Mizar Mathematical Library (MML). There we find, for example, the following theorem about groups.

```
theorem
for V being Group
for v being Element of V holds v - v = 0.V;
```

For a number of concrete groups (rings or fields) this theorem, however, has been proved and stored in MML again, among them complex numbers and polynomials.

```
theorem
for a being complex number holds a - a = 0;

theorem
for L be add-associative right_zeroed right_complementable
        (non empty addLoopStr)
for p be Polynomial of L holds p - p = 0_.(L);
```

One reason might be that authors are not aware of the abstract theorems they can use and therefore believe that it is necessary to include these theorems in the concrete case. This might be especially true, if authors work on applications rather than on "core" mathematics. On the other hand it might just be more comfortable for authors to work solely in the concrete structure rather than to switch between concrete and abstract structures while proving theorems in a concrete structure.

Constructing new structures from already existent ones sometimes causes a similar problem: Shall we formalize a more concrete or a more abstract construction? Multivariate polynomials, for example, can be recursively constructed from univariate polynomials using $R[X, Y] \cong (R[X])[Y]$; or more concrete as functions from Terms in X and Y into the ring R. Which version is better suited for mathematical repositories? Hard to say, from a mathematical point of view the first version is the more interesting construction. The second one, however, seems more intuitive and may be more convenient to apply in other areas where polynomials are used. So, it might be reasonable to include both constructions in a repository. In this case, however, theorems about polynomials will duplicate also.

We close this section with another example: rational functions. Rational functions can be constructed as pairs of polynomials or as the completion $K(X)$ of the polynomial ring $K[X]$. As in the case of multivariate polynomials both constructions have its right in its own, so again both may be included in a repository. Note that this eventually might result in another (two) concrete version(s) of the theorem about groups from above, e.g.

```
theorem
for L being Field
for z being Rational_Function of L holds z - [0_.(L),1_.(L)] = z;
```

5 Representational Issues

In the majority of cases it does not play a major role how mathematical objects are represented in repositories. Whether the real numbers, for example, are introduced axiomatically or are constructed as the Dedekind-completion of the rational numbers, has actually no influence on later formalizations using real numbers. Another example are ordered pairs: Here we can apply Kuratowski's or Wiener's definition that is

$$(a,b) = \{\{a\},\ \{a,b\}\}$$

or

$$(a,b) = \{\{\{a\},\emptyset\},\{\{b\}\}\}$$

or even again the axiomatic approach

$$(a_1,b_1) = (a_2,b_2) \text{ if and only if } a_1 = a_2 \text{ and } b_1 = b_2.$$

Once there is one of the notions included in a repository formalizations relying on this notion can be carried out more or less the same.

There are, however, mathematical objects having more than one relevant representation. The most prominent example are polynomials. Polynomials can be straightforwardly constructed as sequences (of coefficients) over a ring

$$p = (a_n, a_{n-1}, \dots a_0)$$

or as functions from the natural numbers into a ring

$$p = f : I\!N \longrightarrow R \text{ where } |\{x|f(x) \neq 0\}| < \infty.$$

Note that both representations explicitly mention all zero coefficients of a polynomial, that is provide a dense representation.

There is an alternative seldom used in mathematical repositories: sparse polynomials. In this representation only coefficients not equal to 0 are taken into account — at the cost that exponents e_i have to be attached. We thus get a list of pairs:

$$p = ((e_1,a_1),(e_2,a_2),\dots(e_m,a_m)).$$

Though more technical to deal with — that probably being the reason for usually choosing a dense representation for formalization — there exist a number of efficient algorithms based on a sparse representation, for example interpolation and computation of integer roots. Therefore it seems reasonable to formalize both representations in a repository, thus reflecting the mathematical treatment of polynomials.

Another example is the representation of matrices, also a rather basic mathematical structure. The point here is that there exist many interesting subclasses of matrices, for example block matrices for which a particular multiplication algorithm can be given or triangular matrices for which equations are much easier to solve. Hence it might be reasonable to include different representations of matrices, that is different (re-) definitions, in a repository to provide support for particular applications of matrices.

6 Generalization of Theorems

Generalization of theorems is everyday occurrence in mathematics. In the case of mathematical repositories generalization is a rather involved topic: It is not obvious whether the less general theorem can be eliminated. Proofs of other theorems using the original version might not work automatically with the more general theorem instead. The reason may be that a slightly different formulation or even a different version of the original theorem has been formalized. Then the question is: Should one rework all these proofs or keep both the original and the more general theorem in the repository? To illustrate that this decision is both not trivial and important for the organization of mathematical repositories we present in this section some generalizations of the CRT taken from [Sch09].

A rather uncomplicated generalization of Theorem 1 is based on the observation that the range in which the integer u lies, does not need to be fixed. It is sufficient that it has the width $m = m_1 m_2 \cdots m_r$. This easily follows from the properties of the congruence $\equiv$.

Theorem 3. Let m_1, m_2, $\ldots, m_r$ be positive integers such that m_i and m_j are relatively prime for $i \neq j$. Let $m = m_1\, m_2 \cdots m_r$ and let a, u_1, u_2, $\ldots, u_r$ be integers. Then there exists exactly one integer u with

$$a \leq u < a + m \text{ and } u \;\equiv\; u_i \bmod m_i$$

for all $1 \leq i \leq r$. $\diamond$

It is trivial that for $a = 0$ we get the original Theorem 1. Old proofs can very easily be adapted to work with this generalization of the theorem. Maybe the system checking the repository even automatically infers that Theorem 3 with $a = 0$ substitutes the original theorem. If not, however, even the easy changing all the proofs to work with the generalization can be an extensive, unpleasant, and time-consuming task.

A second generalization of the CRT is concerned with the underlying algebraic structure. The integers are the prototype example for Euclidean domains. Taking into account that the residue class ring $\mathcal{Z}_n$ in fact is the factor ring of $\mathcal{Z}$ by the ideal $n\mathcal{Z}$, we get the following generalization.[3]

[3] Literally this is a generalization of Theorem 2, but of course Theorem 1 can be generalized analogously.

Theorem 4. Let R be a Euclidean domain. Let $m_1, m_2, \ldots, m_r$ be positive integers such that m_i and m_j are relatively prime for $i \neq j$ and let $m = m_1 m_2 \cdots m_r$. Then we have the ring isomorphism

$$R/(m) \;\cong\; R/(m_0) \times \cdots \times R/(m_r). \qquad \diamond$$

This generalization may cause problems: In mathematical repositories it is an important difference whether one argues about the set of integers (with the usual operations) or the ring of integers: They have just different types. Technically, this means that in mathematical repositories we often have two different representations of the integers. In the mathematical setting theorems of course hold for both of them. However, proofs using one representation will probably not automatically work for the other one. Consequently, though Theorem 4 is more general, it might not work for proofs using integers instead of the ring of integers; for that a similar generalization of Theorem 1 could be necessary. So in this case in order to make all proofs work with a generalization, we need to provide generalizations of different versions of the original theorem — or just change the proofs with the "right" representation leading to an unbalanced organization of the repository.

We close this section with a generalization of the CRT that abstracts away even from algebraic structures. The following theorem [Lün93] deals with sets and equivalence relations only and presents a condition whether the "canonical" function σ is onto.

Theorem 5. Let α and β be equivalence relations on a given set M. Let $\sigma : M \longrightarrow M/\alpha \times M/\beta$ be defined by $\sigma(x) := (\alpha(x), \beta(x))$. Then we have $\ker(\sigma) = \alpha \cap \beta$ and σ is onto if and only if $\alpha \circ \beta = M \times M$. $\diamond$

Here almost all of the familiar CRT gets lost. There are no congruences, no algebraic operations, only the factoring (of sets) remains. Therefore, it seems hardly possible to adapt proofs using any of the preceding CRTs to work with this generalization. Any application will rely on much more concrete structures, so that too much effort has to be spent to adapt a proof. Theorem 5 in some sense is too general to reasonably work with. However, even though hardly applicable, the theorem stays interesting from a didactic point of view.[4] It illustrates how far — or even too far — we can generalize and may provide the starting point of a discussion whether this is — aside from mathematical aesthetics — expedient.

7 Which Way To Go?

Having seen that it's more or less unavoidable to duplicate knowledge in mathematical repositories, the question is how to deal with such situations. In the following we will mention some issues hoping to start a discussion towards guidelines for building mathematical repositories.

[4] In fact the proof of Theorem 5 has been an exercise in lectures on linear algebra.

Different mathematical versions of theorems (as discussed in section 3.2) describe the same mathematical issue from different points of view. So it would be a natural approach to actually prove that such theorems are equivalent. Though troublesome to accomplish, this also would make explicit that more than one version is present in a repository. The mathematical context in which a particular version is formulated and proven then would be clear from the proof, thus would be visible to users for further applications. Much harder to deal with is the question of which version of a theorem to use. Here, not only deep mathematical understanding and knowledge, but also fondness of particular mathematical views and techniques may play important roles.

In the case of different representations (see section 5) one idea is to provide theorems describing the connection between these. For example, we can prove that sparse and dense polynomials are isomorphic (as rings). This does not only provide information between these representation, but also gives the possibility to translate from one representation into another. Consequently, though a bit tedious, theorems for one representation can be used in the other one. One future goal could be to automate such translations.

Focusing on individual operations and not on structures as a whole, the process of translation between representations has been (partially) automated in Mizar. In the Mizar Mathematical Library we find definitions of both the integers as (just) numbers and the ring of integers. So, integers and elements of the ring of integers are different objects with different operations realizing addition, multiplication, and so on. Using a special registration `identify` the user, however, can identify terms and operations from different definitions ([Kor09]), in our example integers with elements of the ring of integers:

```
registration
  let a,b be integer number;
  let x,y be Element of INT.Ring;
  identify x+y with a+b when x=a, y=b;
  identify x*y with a*b when x=a, y=b;
end;
```

After this registration, theorems proved for integers can be applied to elements of the ring of integers without any translation issues.

We believe that the duplication of theorems in more special cases (compare section 4) can be avoided by providing more support for searching in mathematical repositories. Here, of course, we do not speak of ordinary searching: having a general theorem for an algebraic structure such as group or ring, we have to "search" for concrete structures in which this theorem holds. Or, putting it the other way round, when working in a concrete structure we'd like to find theorems true for this structure, though proved in a more general case. One possibility here, is to provide software that computes the theory of a given structure taking into account that part of this theory have to be generated from more general theories. Such a theory generator, in some sense, would transfer knowledge from a mathematical repository itself into the supporting software.

When it comes to generalizations of theorems, we have seen in section 6 that from a technical point of view they are hard to deal with: Deleting a theorem for which a more general version exists, can imply major changes of the repository. One can, however, think of a software detecting more general versions of a theorem. This would at least give the possibility to automatically identify generalizations. If more than one version is present in a repository such a software in addition can support users in their decision which version of a theorem to use best to construct their proofs.

7.1 MML — The State of the Art

In this subsection we collect some solutions of the considered issues based on the policy of the Library Committee of the Association of Mizar Users.

Ordinary repetitions. Although direct explicit repetitions of definitions or simple theorems with standard proofs are not desirable in large repositories of mathematical knowledge — they make more noise to search within — one can potentially find the pros of such approach; even if the fact is available from some other articles, we obtain complete source of all properties in a single file. In MML such freedom is not allowed and usually leads to the rejection of the submitted article (or, at least to the removal of such straight duplications).

Proof variants. In the above case the author usually lacks knowledge about formalized facts, or is unable to formulate a proper query (e.g. when using MML Query searching tool). Sometimes the situation is much more complex — submitted proofs can be better, shorter or just the original ones. As of now, the author is supposed either to delete such fact from his/her current submission or to revise the one already available in the repository. This policy met with the general criticism; potentially MML lost some valuable submissions; but the origins of such policy stemed when the software automatically removing identical theorems was quite frequently used.

Simple consequences. By a simple consequence we understand the theorem which is justified only by single theorem already present in the MML. Such consequences were automatically removed but as of now it is postponed.

Generalizations. Such activity is twofold: On the one hand after more complex revisions of the library, unused assumptions are automatically removed; hence facts are generalized by the software. On the other hand, MML is gradually divided into concrete and abstract parts (those articles which don't use the notion of the structure and the rest). The articles belonging to the concrete part are put earlier on the list of processing of the MML and are usually more general than those remaining. As an example, we can consider functions with values in the ring of complex, real or integer numbers. Here the generalization which comes to mind quite naturally is just considering complex functions with associated operations — and hence all these are special cases. The danger is here that if we go too far, all the usefulness can get lost (e.g. quaternions or two possible extensions of real numbers available in the same time: complex and extended real numbers).

Important facts. "Important" theorems are usually exception of the above rule — they remain in their original place. Although it is generally hard to measure such importance, one of the criteria is, e.g. the presence of the fact in Freek Wiedijk's Top 100 mathematical theorems list [Wie06]. Sometimes well-known concrete instantiations are proven earlier as simpler ones — as many of the lemmas proven for the Jordan Curve Theorem. To keep the library as compact as possible we should at least to hide these items (not to delete them completely as they might still be used by the main theorem), i.e. not to export them into the Mizar database; final decision was not to do this (however "obsolete" pragma could be useful). However, the proof of the irrationality of the square root of 2 is not present in the MML — but the irrationality of any prime number obviously is.

Automatic discovery. Detecting and removing identical theorems was always important problem for the MML, but when checker was strengthened, especially via attributes mechanism, such activity was less intensive. Here the work of Josef Urban (MoMM) [Urb06] are worth noticing; many connections between proven facts are "up-to-environment description" — obvious for software, but not for a human (to give a trivial but explaining example — it is automatically derived via the cluster registration mechanism that any singleton is non-empty, so all theorems true for non-empty sets can be applied to singletons; if someone does not include proper identifiers into his environment declaration, it will not be the case).

Tighter net of notions. Many of the notions within the Mizar Mathematical Library are stated in terms of adjectives, plaing a role of axioms for theories. If this structure will be more precise, the possibility of the automatic detection of the interconnections between various notions will be higher. That's what the MML referees have especially in mind.

Parallel developments. There are some exceptions of the above rules which can be easily justified. E.g., we have two approaches to lattices (based on the ordering relation and equationally defined), two alternative views for the category theory, and even four distinct approaches to Petri nets (although one of them is currently being eliminated). There is however no clear view of how to merge them, or, as in the case of lattices, the expressive power of the Mizar language does not allow for such elimination.

8 Conclusions

When building a mathematical repository it seems plausible to not duplicate knowledge in order to avoid an unnecessary blow-up of the repository. This is similar to — and may be inspired by — mathematical definitions, in which the number of axioms is kept as small as possible.

In this paper we have argued that this, however, is not true in general. We have analyzed miscellaneous situations in which it might be reasonable or even necessary to duplicate knowledge in a repository. The reasons for that are manifold: Different proofs may be interesting for didactic reasons or different representations of the same knowledge may better support different groups of users. Even improvements of a repository may lead to duplication of knowledge because e.g. a theorem, that has been generalized, cannot always be trivially erased without reworking lots of proofs.

Based on our analysis we have also outlined some ideas how to cope with or maybe avoid duplication of knowledge giving rise to further research. In general, it is hardly foreseeable in which cases which kind of knowledge should be duplicated. This strongly depends on different kind of users the repository should attract.

References

[Coq10] The Coq Proof Assistant, `http://coq.inria.fr`

[Dav03] Davenport, J.H.: MKM from Book to Computer: A Case Study. In: Asperti, A., Buchberger, B., Davenport, J.H. (eds.) MKM 2003. LNCS, vol. 2594, pp. 17–29. Springer, Heidelberg (2003)

[DeB87] de Bruijn, N.G.: The Mathematical Vernacular, a language for mathematics with typed sets. In: Dybjer, P., et al. (eds.) Proceedings of the Workshop on Programming Languages, Marstrand, Sweden (1987)

[GG99] von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, Cambridge (1999)

[GS06] Grabowski, A., Schwarzweller, C.: Translating Mathematical Vernacular into Knowledge Repositories. In: Kohlhase, M. (ed.) MKM 2005. LNCS (LNAI), vol. 3863, pp. 49–64. Springer, Heidelberg (2006)

[Har10] Harrison, J.: The HOL Light Theorem Prover, `http://www.cl.cam.ac.uk/~jrh13/hol-light`

[Hur03] Hurd, J.: Verification of the Miller-Rabin Probabilistic Primality Test. Journal of Logic and Algebraic Programming 50(1-2), 3–21 (2003)

[KZ89] Kapur, D., Zhang, H.: An Overview of Rewrite Rule Laboratory (RRL). In: Dershowitz, N. (ed.) RTA 1989. LNCS, vol. 355, pp. 559–563. Springer, Heidelberg (1989)

[KN04] Kamareddine, F., Nederpelt, R.: A Refinement of de Bruijn's Formal Language of Mathematics. Journal of Logic, Language and Information 13(3), 287–340 (2004)

[Knu97] Knuth, D.: The Art of Computer Programming. In: Seminumerical Algorithms, 3rd edn., vol. 2, Addison-Wesley, Reading (1997)

[Kon97] Kondracki, A.: The Chinese Remainder Theorem. Formalized Mathematics 6(4), 573–577 (1997)

[Kor09] Korniłowicz, A.: How to Define Terms in Mizar Effectively. Studies in Logic, Grammar and Rhetoric 18(31), 67–77 (2009)

[Lün93] Lüneburg, H.: Vorlesungen über Lineare Algebra, BI Wissenschaftsverlag (1993) (in German)

[Mén10] Ménissier-Morain, V.: A Proof of the Chinese Remainder Lemma, `http://logical.saclay.inria.fr/coq/distrib/current/contribs/ZChinese.html`

[Miz10] The Mizar Home Page, `http://mizar.org`

[NB04] Naumowicz, A., Byliński, C.: Improving Mizar Texts with Properties and Requirements. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 290–301. Springer, Heidelberg (2004)

[PSK04] Pollet, M., Sorge, V., Kerber, M.: Intuitive and Formal Representations: The Case of Matrices. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 317–331. Springer, Heidelberg (2004)

[RT01] Rudnicki, P., Trybulec, A.: Mathematical Knowledge Management in Mizar. In: Buchberger, B., Caprotti, O. (eds.) Proceedings of the 1st International Conference on Mathematical Knowledge Management, Linz, Austria (2001)

[Sch08] Schwarzweller, C.: Modular Integer Arithmetic. Formalized Mathematics 16(3), 247–252 (2008)

[Sch09] Schwarzweller, C.: The Chinese Remainder Theorem, its Proofs and its Generalizations in Mathematical Repositories. Studies in Logic, Grammar and Rhetoric 18(31), 103–119 (2009)

[Urb06] Urban, J.: MoMM — Fast Interreduction and Retrieval in Large Libraries of Formalized Mathematics. International Journal on Artificial Intelligence Tools 15(1), 109–130 (2006)

[Wie06] Wiedijk, F.: On the Usefulness of Formal Methods. In: Nieuwsbrief van de NVTI, pp. 14–23 (2006)

[ZH92] Zhang, H., Hua, X.: Proving the Chinese Remainder Theorem by the Cover Set Induction. In: Kapur, D. (ed.) CADE 1992. LNCS, vol. 607, pp. 431–455. Springer, Heidelberg (1992)

Adapting Mathematical Domain Reasoners

Bastiaan Heeren[1] and Johan Jeuring[1,2]

[1]School of Computer Science, Open Universiteit Nederland
P.O.Box 2960, 6401 DL Heerlen, The Netherlands
{bhr,jje}@ou.nl
[2] Department of Information and Computing Sciences, Universiteit Utrecht

Abstract. Mathematical learning environments help students in mastering mathematical knowledge. Mature environments typically offer thousands of interactive exercises. Providing feedback to students solving interactive exercises requires domain reasoners for doing the exercise-specific calculations. Since a domain reasoner has to solve an exercise in the same way a student should solve it, the structure of domain reasoners should follow the layered structure of the mathematical domains. Furthermore, learners, teachers, and environment builders have different requirements for adapting domain reasoners, such as providing more details, disallowing or enforcing certain solutions, and combining multiple mathematical domains in a new domain. In previous work we have shown how domain reasoners for solving interactive exercises can be expressed in terms of rewrite strategies, rewrite rules, and views. This paper shows how users can adapt and configure such domain reasoners to their own needs. This is achieved by enabling users to explicitly communicate the components that are used for solving an exercise.

1 Introduction

Mathematical learning environments and intelligent tutoring systems such as MathDox [8], the Digital Mathematics Environment (DWO) of the Freudenthal Institute [9], and the ActiveMath system [14], help students in mastering mathematical knowledge. All these systems manage a collection of learning objects, and offer a wide variety of interactive exercises, together with a graphical user interface to enter and display mathematical formulas. Sophisticated systems also have components for exercise generation, for maintaining a student model, for varying the tutorial strategy, and so on. Mathematical learning environments often delegate dealing with exercise-specific problems, such as diagnosing intermediate answers entered by a student and providing feedback, to external components. These components can be computer algebra systems (CAS) or specialized domain reasoners.

The wide range of exercise types in a mathematical learning environment is challenging for systems that have to construct a diagnosis from an intermediate student answer to an exercise. In general, CAS will have no problem calculating an answer to a mathematics question posed at primary school, high school, or undergraduate university level. However, CAS are not designed to give detailed diagnoses or suggestions to intermediate answers. As a result, giving feedback using CAS is difficult. Domain reasoners, on the other hand, are designed specifically to give good feedback.

Developing, offering, and maintaining a collection of domain reasoners for a mathematical learning environment is more than just a software engineering problem applied to domain reasoners. Mathematical learning environments usually offer topics incrementally, building upon prior knowledge. For example, solving linear equations is treated before and used in solving quadratic equations. Following Beeson's principles [4] of *cognitive fidelity* (the software solves the problem as a student does) and *glassbox computation* (you can see how the software solves the problem), domain reasoners should be organized with the same incremental and layered organization. Structuring domain reasoners should therefore follow the organization of mathematical knowledge.

Domain reasoners are used by learners, teachers, and developers of mathematical environments. Users should be able to customize a domain reasoner [16]. The different groups of users have various requirements with respect to customization. For example, a learner might want to see more detail at a particular point in an exercise, a teacher might want to enforce that an exercise is solved using a specific approach, and a developer of a mathematical environment might want to compose a new kind of exercise from existing parts. Meeting these requirements is challenging in the development of domain reasoners. It is our experience that users request many customizations, and it is highly unlikely that a static collection of domain reasoners offering exercises at a particular level will be sufficient to satisfy everyone. Instead, we propose a dynamic approach that enables the groups of users to customize the domain reasoners to their needs.

In this paper we investigate how we can offer users the possibility to adapt and configure domain reasoners. In the first part of the paper we identify the problems associated with managing a wide range of domain reasoners for mathematics, and we argue why allowing configuration and adaptation of the concepts describing domain reasoners is desirable. This is the paper's first contribution. Section 2 further motivates our research question. We then give a number of case studies in Section 3 that illustrate the need for adaptation and configuration. Most of these case studies are taken from our work on developing domain reasoners for about 150 applets from the DWO of the Freudenthal Institute.

The second part starts with an overview of the fundamental concepts by means of which we describe mathematical knowledge for solving exercises in domain reasoners. We show how these concepts interoperate, and how they are combined (Section 4). Next, we present a solution for adapting and configuring domain reasoners in Section 5, which is our second contribution. In particular, we show how our solution helps in solving the case studies. The techniques that are proposed in this paper have been implemented in our framework for developing domain reasoners[1], and we are currently changing the existing domain reasoners accordingly. We evaluate the advantages and disadvantages of our approach, and draw conclusions in the final section.

[1] For more information, visit our project webpage at `http://ideas.cs.uu.nl/`

2 Motivation

Computer algebra systems (CAS) are designed specifically for solving complex mathematical tasks, and performing symbolic computations. CAS are often used in intelligent tutoring systems as a back-end for assessing the correctness of an answer. In general, they are suitable for such a task, although different normal forms can have subtle effects on an assessment [5]. CAS are less suitable for supporting more advanced tutoring functionality, such as suggesting a meaningful next step, showing a worked-out example, or discovering a common misconception: they have not been designed to do so, and generally violate the principles of cognitive fidelity and glassbox computation.

Specialized domain reasoners are designed with excellent facilities for feedback and diagnosis in mind. Because they are specialized they often operate on a narrow class of exercises (e.g., only linear equations). Supporting more, related classes (e.g., all mathematics topics covered in high school) raises the question how the knowledge should be organized and managed. Mathematical knowledge is typically hierarchical, and according to the principle of cognitive fidelity, such hierarchies should also be present in a domain reasoner for mathematics.

2.1 Feedback Services

When a mathematical learning environment uses domain reasoners for several classes of exercises, it is important that the reasoners share a set of *feedback services*, and that these services are exercise independent. We have defined such a set of services around rewrite strategies [13,10], which produce step-wise solutions for exercises. With a strategy we can produce worked-out examples (the *derivation* service), suggest a next step (the *allfirsts* service), and diagnose a term submitted by a learner (the *diagnose* service). By collecting the rewrite rules of a strategy, we can report which rules can be applied (the *applicable* service), or recognize common misconceptions (the *findbuggyrules* service). Other services we offer are variations of the ones listed above. All services calculate feedback automatically from a strategy specification and rewrite rules.

Goguadze [11] describes a set of feedback services used by the ActiveMath learning environment to serve as an interface for calling external domain reasoners. His services are similar to ours, and also assume the presence of rewrite rules. However, they do not depend on rewrite strategies. Neither his nor our current services [10] accommodate for customizing and adapting domain reasoners.

2.2 Customization from Four Perspectives

Using a predefined collection of domain reasoners that cannot be customized limits the level of adaptivity of a learning environment. Users of an environment have many wishes about customizing a domain reasoner, and satisfying these would lead to many variants. We propose a solution in which users can adapt a domain reasoner without changing the domain reasoner's implementation. We identify four perspectives for which we consider customizability and adaptability. These perspectives correspond to the different groups of users.

- **Learners.** Learners want to customize an exercise to their own level of expertise. They expect guidance at points where they experience difficulties. Learners do not interact with a domain reasoner directly, but they send their requests by way of a learning environment.
- **Teachers.** Teachers have specific requests about how an exercise should be solved, and using which steps. They have a good understanding of the capabilities of a particular homogeneous group of learners. Teachers want to tailor exercises at a high level.
- **Mathematical learning environments.** A learning environment is the front-end for practicing mathematical problem solving, and usually offers many different classes of exercises. Advanced environments include tools for authoring exercises (for teachers), they maintain a model of a learner, and can have a component for adaptive course generation [19]. All these aspects are related to domain reasoners, and the facilities they offer for customization. Environments are the primary clients of a domain reasoner.
- **Domain reasoners.** From within a domain reasoner, the main concerns are reusability and maintainability of code and components. The major issue is how mathematical knowledge should be represented and organized, reflecting the layered structure of that knowledge.

Each of the case studies that is presented in the next section belongs to one of the perspectives.

3 Case Studies

This section presents five case studies illustrating the need for dynamic domain reasoners that are easily adaptable. Afterwards, we propose a solution, and revisit the cases in Section 5.6.

3.1 Case Study: Controlling the Solutions for an Exercise

A quadratic equation can be solved in many ways. For example, the Dutch mathematics textbook Getal & Ruimte [1], used in more than half of the high schools in the Netherlands, gives many techniques to solve an equation of the form $ax^2 + bx + c = 0$. It considers the case of a binomial ($b = 0$ or $c = 0$) and the case where its factors can be found easily. Furthermore, the book shows how $(x + 3)^2 = 16$ can be solved without reworking the term on the left-hand side. Of course, the quadratic formula is given as a general approach, although using it is discouraged because it is more involved. Figure 1 shows alternative derivations for a quadratic equation, including a derivation in which the technique of "completing the square" is used. Selecting the appropriate technique for a given equation is one of the skills that needs training.

Depending on the context, a *teacher* may want to control the way in which a particular (set of) exercise(s) is solved. For example, a certain exercise should be

$$\begin{aligned}&x^2 - 4x = 12\\&x^2 - 4x - 12 = 0\\&(x-6)(x+2) = 0\\&x = 6 \vee x = -2\end{aligned}$$

$$\begin{aligned}&x^2 - 4x = 12\\&x^2 - 4x + 4 = 16\\&(x-2)^2 = 4^2\\&x - 2 = 4 \vee x - 2 = -4\\&x = 6 \vee x = -2\end{aligned}$$

$$\begin{aligned}&x^2 - 4x = 12\\&x^2 - 4x - 12 = 0\\&\left|\begin{aligned}\;D &= (-4)^2 - 4 \cdot 1 \cdot -12\\&= 64\\\sqrt{D} &= \sqrt{64} = 8\end{aligned}\right.\\&x = \tfrac{4+8}{2} \vee x = \tfrac{4-8}{2}\\&x = 6 \vee x = -2\end{aligned}$$

Fig. 1. Three possible derivations for a quadratic equation

solved without using the quadratic formula, or without the technique of completing the square (because it may not be part of the course material). Controlling the solution space not only has an effect on the diagnosis of an intermediate term entered by a learner, it also influences the generation of hints and worked-out solutions. A strategy that combines multiple solution techniques will often not be of help, since hints and worked-out solutions might refer to techniques unknown to the learner, or techniques that should not be used.

3.2 Case Study: Changing the Level of Detail

While doing an exercise, a *learner* wants to increase the level of detail that is presented by the learning environment, i.e., the granularity of the steps. For example, the learner might find the step in which $x = \frac{1}{2}\sqrt{32}$ is simplified to $x = 2\sqrt{2}$ hard to understand, even though familiarity with simplifying roots is assumed. According to the principle of glass-box computation the learner should be able to inspect the calculations within this step. An extreme scenario in the other direction is a learner who is only interested in the final answer, not in the intermediate answers.

3.3 Case Study: Changing the Number System

A *teacher* wants to allow complex numbers in solutions for polynomial equations, instead of real numbers. In the setting with real numbers, a negative discriminant (or a squared term that has to be negative) leads to no solutions. According to the principle of cognitive fidelity, the software should solve the problem with complex numbers or with real numbers, depending on the teacher's preference. However, the approach to solve an equation, that is, the rewrite strategy, is not changed significantly. Therefore, reuse of the existing strategy is desirable. A similar scenario would be to restrict the numbers in an equation to rationals only, without introducing square roots.

3.4 Case Study: Creating New Exercises from Existing Parts

Rewrite strategies can often be extended to deal with a new class of exercises by performing some steps beforehand or afterwards. In the case of solving an equation with a polynomial of degree 3 or higher, one could try to reduce the problem to a quadratic equation. This equation can then be handled by an existing strategy for solving quadratic equations. Ideally, such a composite strategy is already defined and available. If not, a mathematical *learning environment* (or a *teacher* using it) should be able to assemble the strategy from existing parts, and use it in the same way as a predefined strategy.

Another scenario is a collection of rules that has to be applied exhaustively to solve an exercise. Although exhaustive application of rules results in a very simple rewrite strategy, many interesting problems can be solved in this way. It should therefore be possible for a *teacher* using the learning environment to take or specify such a collection, and to construct a strategy out of it.

3.5 Case Study: Customizing an Exercise with a Student Model

Advanced *learning environments*, such as ActiveMath, maintain a student model containing information about the skills and capabilities of the learner. Such a student model can be used for different purposes, including task selection and reporting the progress of a learner. Because the model contains detailed knowledge about the level of the learner, it is desirable to use this knowledge and to customize the domain reasoner accordingly. For example, a learner that understands Gaussian elimination can perform this method as a single step when determining the inverse of a matrix. On the contrary, beginners in linear algebra should see the intermediate steps.

Obviously, diagnoses from the domain reasoners should also be used to update the student model. In both cases, the domain reasoner and the learning environment need a shared understanding of the knowledge items, such as the rewrite rules and the rewrite strategies. The exchange of information in both directions suggests that the two parts should be tightly integrated.

4 Concepts and Representation of Knowledge

This section discusses the three concepts that are the foundation of our approach: rewrite rules, rewrite strategies, and views for defining canonical forms. These concepts not only assist in reasoning about exercises at a conceptual level, they are also the core abstractions in the implementation of the domain reasoners. We give a brief introduction to each of the concepts, and point out how they represent knowledge appearing in mathematical textbooks. Furthermore, we highlight the properties of the concepts. In the last part of this section we discuss how the concepts come together in defining an exercise.

4.1 Rewrite Rules

Rewrite rules specify how terms can be manipulated in a sound way, and are often given explicitly in textbooks. Well-known examples are rewriting $AB = 0$ into $A = 0 \lor B = 0$, the quadratic formula, and associativity of addition.

These rules constitute the steps in worked-out solutions. Soundness of rules can be checked with respect to some semantic interpretation of a formula. Such an interpretation can be context-specific (e.g., $x^2 = -3$ gives no solutions for x in $\mathbb{R}$).

Rewrite rules are atomic actions that can be implemented in code. Clearly, this gives the implementer of the rule the full power of the underlying programming language. An alternative is to specify rules with a left-hand side and a right-hand side, and to rely on unification and substitution of terms to do the transformation [15]. This is common practice in term rewrite systems (TRS) [3]. We allow rewrite rules to yield multiple results.

4.2 Rewrite Strategies

Simple problems can be solved by applying a set of rules exhaustively (for instance, when the set of rules is confluent), but this is generally not the case. A rewrite strategy [13] guides the process of applying rewrite rules to solve a particular class of problems. Recipes for solving a certain type of problem can be found in textbooks, but they are often not precise enough for the purpose of building a domain reasoner. Given a collection of worked-out solutions by an expert, one can try to infer the strategy that was used (although typically only one possible derivation is covered).

Rewrite strategies are built from rewrite rules, with combinators for sequences and choices (<*> and <|>, respectively). The fixed point combinator *fix* allows for repeating parts. Labels can be placed at arbitrary places in the strategy, marking substrategies of interest. From a strategy description, multiple derivations may be generated or recognized.

Since strategies only structure the order in which rewrite rules are applied, soundness of a derivation follows directly from the soundness of the rules involved. Note that a strategy not only prescribes which rule to apply, but also where (that is, to which subterm). Also, strategies are designed with a specific goal in mind. A strategy for quadratic equations, for instance, is expected to rewrite an equation until the variable is isolated. The solved form that a strategy is supposed to reach is the strategy's post-condition. Likewise, a strategy may have certain assumptions about the starting term (e.g., the equation must be quadratic, or only a single variable is involved), which is its pre-condition.

4.3 Views and Canonical Forms

Canonical forms and notational conventions are an integral part of courses on mathematics. Examples of conventions in writing down a polynomial are the order of its terms (sorted by the degree of the term), and writing the coefficient in front of the variable. Such conventions also play a role when discussing equations of the form $ax^2+bx = 0$: it is likely that $-3x+x^2 = 0$ is considered an instance of the form, although the expression $1x^2 + (-3)x$ is rather atypical. These implicit assumptions make that standard rewriting techniques do not apply directly.

Canonical forms and notational conventions can be captured in a view [12], which consists of a partial function for matching, and a (complete) function for building. Matching may result in a value of a different type, such as the pair $(-3, 5)$ for the expression $-(3-5)$. In this example, the interpretation of the pair would be addition of both parts. Having a value of a different type after matching can be useful when specifying a rewrite rule: the pair $(-3, 5)$, for instance, witnesses that an addition was recognized at top-level. Building after matching gives the canonical form, and this composed operation must therefore be idempotent. A view is assumed to preserve a term's semantics.

Primitive views can be composed into compound views, in two different ways. Firstly, views are closely related to the arrow interface [17], and its bidirectional variant. The combinators of this interface can be used for combining views, such as using views in succession. Secondly, views can be parameterized with another view. Consider a view for expressions of the form $ax + b$, returning a pair of expressions for a and b. Another view can then be used for these two parts (e.g., a view for rational numbers). Essentially, this pattern of usage corresponds to having higher-order views. Views can be used in different ways:

– as a rewrite rule, reducing a term to its canonical form (if possible);
– as a predicate, checking whether a term has a canonical form;
– as an equivalence relation, comparing the canonical forms of two terms.

4.4 Exercises

The three fundamental concepts for constructing domain reasoners discussed in this section are all we need to support a general set of feedback services (Section 2.1). Instances of the concepts are grouped together in an *exercise* containing all the domain-specific (and exercise-specific) functionality.

The most prominent component of an exercise is its rewrite strategy. In addition to the rewrite rules that are present in the strategy, more rules can be added to the exercise for the purpose of being recognized, including buggy rules for anticipating common mistakes. Predicates are needed for checking whether a term is a suitable starting term that can be solved by the strategy, and whether a term is in solved form. These two predicates can be defined as views. For diagnosing intermediate answers, we need an equivalence relation to compare a submission with a preceding term. This relation can be specified as a view. Besides checking student submissions, this view can be used as an internal consistency check, validating the soundness of the rewrite rules. One more view is needed that checks whether two terms are similar enough to be considered the same. This view is used to bring intermediate terms produced by a strategy to their canonical forms.

What remains to be supplied for an exercise is its metadata, such as an identifier that can serve as a reference, and a short description. For certain domains it is convenient to have a dedicated parser and pretty-printer for the terms involved. For external tools, however, interchanging abstract syntax (as opposed to concrete syntax), such as OpenMath objects [18] for mathematical domains, is the preferred way of communication, avoiding the need for a parser and pretty-printer. Although not of primary importance, it can be convenient to have a randomized term generator for the exercise.

5 Adaptation and Configuration

This section discusses how users can adapt and customize the exercises that are offered by a domain reasoner. A user has to be able to inspect the internals of the components of an exercise, to adapt and replace these components, and to create new exercises. We briefly discuss the consequences of applying the glassbox principle to our components. We then propose representations for rewrite rules, rewrite strategies, and views. These representations are an essential part of the communication with a domain reasoner. Strategy configurations are introduced for conveniently adapting existing strategies. We conclude by returning to our case studies, and show how they can be addressed.

5.1 The Glassbox Principle

The glassbox principle expresses that you should be able to see all steps leading to a final answer. This is possible with our current services, but you cannot query the specifics of a rule that was applied, or examine the structure of the rewrite strategy. From the perspective of a learning environment, rewrite strategies and rules are still black boxes delivering some result. Ideally, the components involved are transparent as well, and adhere to the glassbox principle.

Exposing the internals of a component has the advantage that more details become available for the learning environment, and for other external tools. These details can be communicated to learners, or to teachers writing feedback messages. The information can also be used for documentation, visualization of rewrite strategies, analyses, and much more. Once a domain reasoner supports a representation, it can be extended to interpret descriptions that are passed to it. As a result, exercises can be adapted in new, unforeseen ways.

However, there is a trade-off in making components fully transparent. The need for a representation that can be communicated restricts the way components can be specified. The developer of a domain reasoner can no longer take advantage of the facilities offered by the underlying programming language, which may negatively affect performance, for example. For our own domain reasoners, we are gradually working towards transparency.

5.2 Representing Rewrite Rules

Consider the rewrite rule $AB = AC \rightarrow A = 0 \vee B = C$. In this rule, A, B, and C are meta-variables representing arbitrary expressions. A rule that is written in this way can be seen as a Formal Mathematical Property (FMP), a concept introduced by the OpenMath standard [18] to specify properties of symbols that are defined in content dictionaries. The OpenMath standard also supports explicit quantification of meta-variables by means of the `forall` binder in the `quant1` dictionary. We can thus use FMPs to represent the rewrite rules of our domain reasoners[2]. Likewise, buggy rules can be communicated as FMPs, except

[2] Instead of using FMPs, we could have introduced our own representation, in which case we would still need quantification, meta-variables, and a pair of terms.

that the meta-variables are existentially quantified. Indeed, many of our rewrite rules can also be found in a content dictionary as an FMP.

Unfortunately, not all rules can be represented with a left and right-hand side straightforwardly. Keep in mind that the representation of a rule should closely correspond to how it is perceived by a learner. We give some examples that challenge this approach.

– Some steps correspond to primitive operations, such as replacing $3+5$ by 8, or reducing $\frac{10}{15}$ to $\frac{2}{3}$. Special support is needed for these operations.
– Rewrite rules should not have meta-variables on the right-hand side that do not appear on the left [3]. Conceptually, however, such rules do exist as an intermediate step, such as the rule for scaling a fraction ($\frac{A}{B} \rightarrow \frac{AC}{BC}$), as a preparatory step for adding it to another fraction. This rule also shows that rules can have side conditions ($C \neq 0$), which can be expressed in an FMP.
– Generalizations of rules involving a variable number of terms require special support. An example of such a rule is $A(B_1 + \ldots + B_n) \rightarrow AB_1 + \ldots + AB_n$.
– In an earlier paper [12] we have argued that rules are specified in the context of a view, yet there is no support for views in the rewrite rules.

These cases can only be circumvented partially by having explicit support for views in rewrite rules (i.e., associate a new symbol with a view, and specialize the unification procedure for that symbol), or by using strategies as a representation for rules (recall that rules can return multiple results).

With this representation for rewrite rules, learning environments can communicate new rules to the domain reasoner, thereby extending it. Essentially, this turns the domain reasoner into a rewrite rule *interpreter*. When allowing dynamic extension of a domain, it may no longer be possible to guarantee (or check) the soundness of rules. Also, care should be taken that the new rules do not result in excessive computations.

5.3 Representing Rewrite Strategies

Rewrite strategies are specified using a small set of combinators, such as <∗> for sequence, and <|> for choice. Additional combinators are defined in terms of this small set (e.g., *repeat*), resulting in a combinator library with common patterns. For example, consider the strategy specification for solving a linear equation, in which both sides of the equation are first rewritten into their basic form $ax + b$ (the preparation step).

```
lineq   = label "linear equation" (prepare <*> basic)
prepare = label "prepare equation"
              (repeat (merge <|> distribute <|> removeDivision))
basic   = label "basic equation"
              (try varToLeft <*> try conToRight <*> try scaleToOne)
```

This strategy specification is declarative and compositional, which allows for an almost literal translation into an XML equivalent. The XML fragment for the *lineq* strategy is given below:

```
<label name="linear equation">
  <sequence>
    <label name="prepare equation">
      <repeat><choice>
        <rule name="merge"/>
        <rule name="distribute"/>
        <rule name="remove division"/>
      </choice></repeat>
    </label>
    <label name="basic equation"> ... </label>
  </sequence>
</label>
```

An XML tag is introduced for each combinator, and labels and rules have attributes for storing additional information. The strategy combinators for sequence and choice are associative, and therefore we let their corresponding tags have arbitrary many children, instead of imposing a nested structure. The declarative nature of rewrite strategies makes that such a convention does not interfere with the meaning of the strategy, i.e., it is easy to reason about strategies.

An important design decision in the representation of rewrite strategies is which of the derived combinators to support in XML, and which not. For instance, *repeat* s is defined as *many* s <*> *not* s, where *many* and *not* are also derived combinators. Instead of introducing the tag `<repeat>`, we could use *repeat*'s definition, giving a `<sequence>` tag at top-level. Fewer tags make it easier for other tools to process a strategy description. On the other hand, tools can take advantage of the extra tags (e.g., a tool for visualizing strategies). Hence, we decide to support most of the combinators in our library.

Rules are referenced by name in a strategy. Similarly, known (sub)strategies can be included as well. This is particularly helpful for assembling new strategies from existing parts (both rules and strategies). Under the assumption that the parts have already been defined, we can give a concise strategy description for the running example:

```
<label name="linear equation">
  <sequence>
    <strategy name="prepare equation"/>
    <strategy name="basic equation"/>
  </sequence>
</label>
```

The XML representation paves the way for learning environments to offer their own rewrite strategies, turning the domain reasoner into an interpreter for strategy descriptions. Interpreting strategies raises issues concerning the correctness of the strategy (the post-condition it should establish), and in particular termination when rewriting with the strategy. Experience has shown that specifying rich strategies is a difficult and error-prone activity, for which offline analysis and testing capabilities are very helpful.

5.4 Configuring Rewrite Strategies

New rewrite strategies can be defined from scratch, but often a small change to an existing strategy suffices. Strategy *configurations* offer an alternative (and simpler) way to adapt strategies. With such a configuration, a sequence of transformations can be applied to a strategy.

A useful transformation is to *remove* a specific part of a strategy, such that it is not used in a derivation. This can be carried out by replacing the part (substrategy or rule) by *fail*, which is the unit element of the choice combinator. When you remove a rule, you risk that an exercise can no longer be solved. The inverse transformation is to *reinsert* a part that was marked as removed.

Another transformation is based on the fact that strategies are special instances of rewrite rules, since they can be performed in a single step. Thus, strategies can be *collapsed* into a rule, contributing to just one step in a derivation. The inverse operation is to *expand* a rewrite rule and turn it into a strategy.

The *hide* transformation makes a rule implicit, or the rules in a rewrite strategy. An implicit rule behaves normally, except that it does not show up as a step in a derivation. Implicit rules can be used to perform certain simplification steps automatically, and are comparable to so-called administrative rules [13]. The inverse of *hide* is the *reveal* transformation.

The properties *removed*, *collapsed*, and *hidden* correspond to the transformations described above, and they can be assigned to subexpressions in a strategy description. The properties are translated to attributes in an XML representation. The three inverse transformations appear as attributes set to false, which is their default value. The following XML snippet illustrates this approach:

```
<label name="basic equation" collapsed="true"> ... </label>
```

Note that this still requires the whole strategy to be communicated, including the part that is collapsed. To circumvent this, we introduce an XML tag for each transformation with a target specifying where the transformation should be applied. The following XML fragment takes the original strategy for solving a linear equation, and applies the *collapse* transformation to the substrategy labeled `"basic equation"`:

```
<collapse target="basic equation">
   <strategy name="linear equation"/>
</collapse>
```

Transformations can be combined, and if nested, the innermost is applied first. Because strategy transformations are pure functions, they can be freely mixed with the “regular” strategy combinators.

Instead of removing a part, we have seen cases where the opposite was requested by a teacher: a certain rule (or substrategy) must be used. This can be done by selectively removing parts, and making sure that the mandatory part is used in all cases. For convenience, we offer a *mustuse* transformation doing exactly that, which can be used as the other transformations. A weaker variant is to express a preference for using a rule: this boils down to replacing some choice strategy combinators by the left-biased choice combinator (written $\triangleright$, see [13]).

The *prefer* transformation guarantees that the same set of exercises can be solved by the strategy, which is not the case for *mustuse*. The final transformation we discuss is to *replace* a part of the strategy by something else. This transformation takes a target to be replaced, the replacement (first child), and the strategy in which the replacement has to take place (second child).

5.5 Representing Views

Finding a representation for a view is arguably more difficult than finding one for a rewrite rule or strategy. Since a view is just a pair of functions, it is unclear how its internal structure could be represented in general, other than its implementation in the underlying programming language. We discuss two special cases: a view defined as a confluent set of rewrite rules, and a view specified as a rewrite strategy. Compound views are represented by introducing an explicit representation for the arrow combinators (as was done for the strategy combinators), and a representation for the application of higher-order views.

Some views can be defined as a confluent set of rewrite rules, in particular views for simplifying the complete term, and not just the top-level nodes of the term. The view's function for matching applies the set of rules: its function for building is simply the identity function. Such a view can be represented by listing the rules. Note that confluence ensures that the view returns a canonical form.

Views can also be specified by a rewrite strategy for the view's match function and its build function. This is more sophisticated than providing a confluent set of rewrite rules, because the strategy can control in a precise way how the rules should be applied. The strategy language has a fixed point combinator for expressing general recursion. This makes it plausible that many views can be written as a strategy. The operations of a view must be idempotent, and this property must be checked for views that are represented by a rewrite strategy.

5.6 Case Studies Revisited

We briefly revisit the five case studies. The teacher in case study 3.1 wants to control how an exercise is solved, for example by disallowing certain rules or techniques. A strategy configuration provides this functionality by means of the *remove*, *mustuse*, and *prefer* transformations. The second case (a learner customizing the level of details) is handled likewise: parts in the strategy that have been collapsed can be expanded, or the other way around. To see yet more detail, implicit rules can be made explicit, or rules can be replaced by a rewrite strategy that is doing the same. For example, the quadratic formula introduces a square root, which is simplified immediately because it is not the focus of the exercise. Normalizing expressions involving roots is, however, a topic on its own, for which a rewrite strategy is available. We can plug-in this strategy to increase the level of detail in solving an equation with the quadratic formula.

Changing the underlying number system in an exercise (case study 3.3) is not trivial. Consider using complex numbers for solving a quadratic equation. To start with, some support for the basic operations on complex numbers is needed (e.g., addition and multiplication). This can best be captured in a view. Ideally, a view for complex numbers is already present in the domain reasoner. If not, the view can be specified as a rewrite strategy. This view can be used for bringing an expression with complex numbers to its canonical form. Furthermore, additional rewrite rules are added to the exercise, such as $i^2 \rightarrow -1$. These new rules can be inserted in the strategy for quadratic equations, whereas other rules are excluded (e.g., the rule that a square root of a negative number leads to no solution). The subtle part of this case study is that the views used in the strategy's rewrite rules may also have to change, in particular if they involve calculations with numbers. Composing higher-order views (e.g., a view for polynomials parameterized over the type of its coefficients) alleviates this issue.

Case study 3.4 is solved by interpreting rewrite strategies that are assembled by the learning environment. Not all rewrite rules are representable, which currently limits what can be done without changing the domain reasoner. The last case study involves customizing the level of detail in an exercise, which is highly desirable for adaptive learning systems. Based on the student model, a strategy configuration must be generated by the learning environment.

6 Conclusions, Related and Future Work

We have shown why adapting domain reasoners is very desirable in the context of mathematical learning environments. By explicitly representing the fundamental concepts used in domain reasoners, we can let users adapt and configure a class of exercises in a domain reasoner. We use OpenMath to represent mathematical expressions and rewrite rules, but we have designed our own XML language for specifying rewrite strategies, and transformations on these strategies. Our strategy language is very similar to the tactic languages used in theorem proving [6,2], and has the same expressive power.

Several authors discuss adaptation of various aspects in learning environments [16,19], but we are not aware of previous work on configuring and adapting domain reasoners. Hierarchical proofs [7,2], which represent proofs at different levels of abstraction, are related to turning a strategy into a rule and vice versa. As far as we found, hierarchical proofs are not used to recognize proving steps made by a student.

We have indicated some challenges in representing rewrite rules and views (sections 5.2 and 5.5), and these cases require further investigation. Even though we are striving for domain reasoners that are fully transparent (i.e., that have an explicit representation), we think that hybrid solutions, in which only certain parts can be adapted, are a conceivable compromise. We plan to investigate how the facilities for adapting domain reasoners can best be offered to a teacher or a domain expert, and what skills are reasonable to expect from such a user.

Acknowledgements. This work was made possible by the Math-Bridge project of the Community programme eContent*plus*, and by the SURF Foundation (`http://www.surf.nl`), the higher education and research partnership organization for Information and Communications Technology (ICT), for the NKBW2 project. The paper does not represent the opinion of the Community, and the Community is not responsible for any use that might be made of information contained in this paper. We acknowledge discussions with Peter Boon, George Goguadze, and Wim van Velthoven. We thank Alex Gerdes and the anonymous reviewers for commenting on an earlier version of this paper.

References

1. Admiraal, C., et al.: Getal & Ruimte. EPN, Houten, The Netherlands (2009)
2. Aspinall, D., Denney, E., Lüth, C.: Tactics for hierarchical proof. Mathematics in Computer Science 3(3), 309–330 (2010)
3. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1997)
4. Beeson, M.J.: Design principles of MathPert: Software to support education in algebra and calculus. In: Kajler, N. (ed.) Computer-Human Interaction in Symbolic Computation, pp. 89–115. Springer, Heidelberg (1998)
5. Bradford, R., Davenport, J.H., Sangwin, C.J.: A comparison of equality in computer algebra and correctness in mathematical pedagogy. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) Calculemus 2009, MKM 2009. LNCS, vol. 5625, pp. 75–89. Springer, Heidelberg (2009)
6. Bundy, A.: The use of explicit plans to guide inductive proofs. In: CADE, pp. 111–120 (1988)
7. Cheikhrouhou, L., Sorge, V.: PDS – a three-dimensional data structure for proof plans. In: ACIDCA (2000)
8. Cohen, A., Cuypers, H., Reinaldo Barreiro, E., Sterk, H.: Interactive mathematical documents on the web. In: Algebra, Geometry and Software Systems, pp. 289–306. Springer, Heidelberg (2003)
9. Doorman, M., Drijvers, P., Boon, P., van Gisbergen, S., Gravemeijer, K.: Design and implementation of a computer supported learning environment for mathematics. In: Earli 2009 SIG20 Invited Symposium Issues in Designing and Implementing Computer Supported Inquiry Learning Environments (2009)
10. Gerdes, A., Heeren, B., Jeuring, J., Stuurman, S.: Feedback services for exercise assistants. In: Remenyi, D. (ed.) ECEL, pp. 402–410. Acad. Publ. Ltd., New York (2008)
11. Goguadze, G.: Representation for interactive exercises. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) Calculemus 2009, MKM 2009. LNCS, vol. 5625, pp. 294–309. Springer, Heidelberg (2009)
12. Heeren, B., Jeuring, J.: Canonical forms in interactive exercise assistants. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) Calculemus 2009, MKM 2009. LNCS, vol. 5625, pp. 325–340. Springer, Heidelberg (2009)
13. Heeren, B., Jeuring, J., Gerdes, A.: Specifying rewrite strategies for interactive exercises. Mathematics in Computer Science 3(3), 349–370 (2010)
14. Melis, E., Siekmann, J.: Activemath: An intelligent tutoring system for mathematics. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 91–101. Springer, Heidelberg (2004)
15. van Noort, T., Rodriguez Yakushev, A., Holdermans, S., Jeuring, J., Heeren, B., Magalhães, J.P.: A lightweight approach to datatype-generic rewriting. Journal of Functional Programming (to appear 2010)

16. Pahl, C.: Managing evolution and change in web-based teaching and learning environments. Computers & Education 40(2), 99–114 (2003)
17. Paterson, R.: Arrows and computation. In: Gibbons, J., de Moor, O. (eds.) The Fun of Programming, pp. 201–222. Palgrave, Oxford (2003)
18. The OpenMath Society. The OpenMath Standard (2006), http://www.openmath.org/standard/index.html
19. Ullrich, C., Lu, T., Melis, E.: A new framework for dynamic adaptations and actions. In: Cress, U., Dimitrova, V., Specht, M. (eds.) EC-TEL 2009. LNCS, vol. 5794, pp. 67–72. Springer, Heidelberg (2009)

Integrating Multiple Sources to Answer Questions in Algebraic Topology*

Jónathan Heras, Vico Pascual, Ana Romero, and Julio Rubio

Departamento de Matemáticas y Computación, Universidad de La Rioja,
Edificio Vives, Luis de Ulloa s/n, E-26004 Logroño (La Rioja, Spain)
{jonathan.heras,vico.pascual,ana.romero,julio.rubio}@unirioja.es

Abstract. We present in this paper an evolution of a tool from a user interface for a concrete Computer Algebra system for Algebraic Topology (the Kenzo system), to a front-end allowing the interoperability among different sources for computation and deduction. The architecture allows the system not only to interface several systems, but also to make them cooperate in shared calculations.

1 Different Questions, Different Sources

When working in Mathematics, usually the researcher uses different sources of information. Typically, he can consult some papers or textbooks, make some computations with a Computer Algebra system, check the results against some known tables or, more rarely, verify some conjectures with a proof assistant tool. That is to say, Mathematical Knowledge is dispersed among several sources.

Our aim in this work is to mechanize, in some particular cases, the management of these multiple-source information systems. Since it would be too pretentious to try to solve fully this problem, we work in a very specific context. Thematically, we restrict ourselves to (a subset of) Algebraic Topology. With respect to the sources, in order to have a representation wide enough, we have chosen two Computer Algebra systems (Kenzo and GAP), a theorem prover (ACL2) and a small expert system developed by us. The objective of the expert system is computing *homotopy* groups. Kenzo and GAP can compute *homology* groups of different spaces, but the calculation of *homotopy* is in general much harder. Our homotopy expert system tries to take profit of theoretical knowledge contained in theorems (tables have been excluded up to now, since they are considered less difficult to integrate, from a technological point of view), and can ask computational results to Kenzo, if needed.

This paper is a natural continuation of [17] and [18]. There are three main contributions in the paper: an architecture based on the Broker pattern [8] (proven as an open, flexible and adaptable tool); an Homotopy Expert System (HES) that allows non-trivial computations (and explanations) interacting with Kenzo; and the automation of the interoperability between Kenzo and GAP.

* Partially supported by Ministerio de Ciencia e Innovación, project MTM2009-13842-C02-01.

From the symbolic computation literature, we looked for inspiration in different projects and frameworks such as the MathWeb software bus [5], its successor the MathServe Framework [4], the MoNET project [9,6] or the MathBroker [2] and MathBroker II [3] projects, as well as in other works as [13] or [22].

2 General View of the System

The *Broker* architectural pattern [8] can be used to structure software systems with decoupled components that interact through service invocations. The Broker pattern defines three kinds of participants: *clients*, *components*, and the *broker* itself. A scheme of our architecture based on this pattern is depicted in Figure 1. The mediator (broker) component embeds an *Internal Memory* where a strategy of *memoization* has been systematically implemented (based on the same idea used in GAP for attributes, see [16]). The system stores the results in the internal memory when a computation is executed for the first time, and if the same computation is asked again later, the result is simply looked up and returned, without further computation.

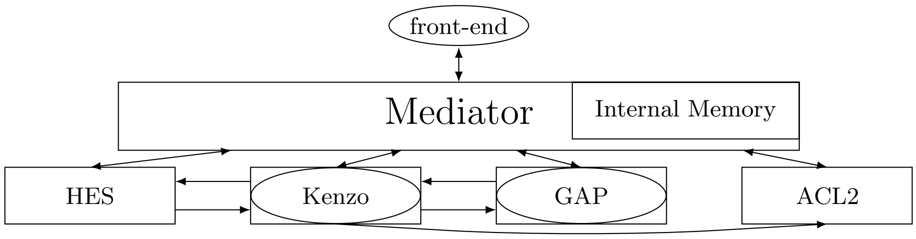


Fig. 1. Broker architectural pattern

The decorator pattern [8] is used to wrap objects of our system with information, like the type of the object (simplicial set, group,...) or the reduction degree [17] if the object is a topological space. This information guides the mediator to decide which component to use. Namely, Kenzo [11] (a Symbolic Computation system devoted to Algebraic Topology) is the core for computations related to homology groups of spaces, GAP [1] (a Computer Algebra system in the area of Computational Group Theory) and HAP [12] (a GAP homological algebra library) are the core for computations related to group homology, ACL2 [19] (a first order logic theorem prover) is the kernel for verifying the correctness of statements and, finally, the Homotopy Expert System (a small module developed by us described in the next paragraph and from now on called HES) is in charge of computing homotopy groups.

HES is a rule-based expert system. The structure of a rule-based expert system, see [15], consists of, and the HES is no exception, the following components: the *Working memory (the facts)*, the *Knowledge base (the rules)*, the *Inference engine*, a *Knowledge acquisition* module and an *Explanation facility* module. In the scope of the HES, the facts are properties associated with the objects (for instance, "$\forall n \in \mathbb{N} : \quad \Delta^n$ is a contractible space"). The current knowledge base is made up of 23 rules (such as, "`if` X is contractible `and` $n \geq 1$ `then` $\pi_n(X) = 0$"). The inference engine uses the *forward chaining* method for reasoning, see [15]. To grapple with the knowledge acquisition aspects, the HES takes profit from both the RuleML markup language [7] and the OMDoc format [20], the former one is used to specify rules in a declarative way and the second one to store concrete functionalities. Last but not least, gathering the applied rules and the facts that decorate each object, our HES is able to provide a trace containing the reasoning followed by it in order to reach a conclusion.

However, the power of our system does not lie in gathering several computer algebra systems and theorem provers and use them separately with the same front end, but interconnecting them to reach new results. The communication among modules is performed by means of the OpenMath language [10], used to represent the objects in a common language for all the systems.

In [21] an approach to coordinate GAP and Kenzo was presented. In that work GAP and Kenzo cooperate in order to compute homology groups of some spaces. These spaces with their homology can then be used in other constructions and applications. Some enhancements of that tool provided by our system are: avoidance of the installation of several programs and packages, automation of communication steps (here the SCSCP protocol [14] plays a key role) and concealment of the details to mix the systems.

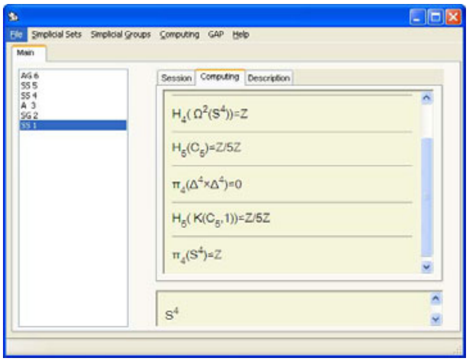
The general procedure and technology to connect with the ACL2 system explained in [18] is now applied to the context of group homology. The Common Lisp code used in Kenzo to represent a group is sent to ACL2 as an instance of an ACL2 *encapsulate* (a mechanism to introduce new functions symbols by axioms constraining them to have certain properties) by means of our broker, which is also in charge of invoking ACL2 in a way transparent for the user.

In another line, Kenzo and the HES cooperate to compute homotopy groups of spaces. In this case, the HES requests Kenzo to compute homology groups which can be used to obtain homotopy groups. Whereas Kenzo communicates with the HES in order to send it results. The idea consists of gathering the knowledge stored in the HES and chaining several tools available in Kenzo to get results which are not reachable by anyone of them working in an independent way.

3 Putting All Together

Our current front-end has evolved from the user interface for Kenzo presented in [17]. Its presentation layer is kept, but its internal mediator has been enriched to support different sources of information. Figure 2 displays some computations which took profit of the following interactions: $H_4(\Omega^2(S^4)) = \mathbb{Z}$ (computed with Kenzo), $H_5(C_5) = \mathbb{Z}/5\mathbb{Z}$ (computed with GAP), $\pi_4(\Delta^4 \times \Delta^5) = 0$ (obtained with the Homotopy Expert System), $H_5(K(C_5, 1)) = \mathbb{Z}/5\mathbb{Z}$ (computed with Kenzo + GAP), $\pi_4(S^4) = \mathbb{Z}$ (obtained with the Homotopy Expert System + Kenzo).

It is worth noting that all results are shown to the user in a unique screen and that the computations are asked from a sole menu, then the user does not know the system in charge of computing neither the collaboration among computer



$H_4(\Omega^2(S^4))=Z$

$H_5(C_5)=Z/5Z$

$\pi_4(\Delta^4 \times \Delta^4)=0$

$H_5(K(C_5,1))=Z/5Z$

$\pi_4(S^4)=Z$

S^4

Fig. 2. The front end for computations

algebra systems, he only receives the desired result. The technical details are hidden to the user. The results related to ACL2 are shown in a different tab to split the computations from the deductions.

4 Conclusions and Further Work

In this paper an architecture to integrate different tools for computing and logical reasoning in Algebraic Topology is presented. Even if our proposal has a limited extend, both thematically and from the point of view of the core systems, we think it shows a solid line of research that could be exported to other areas of mathematical knowledge management. OpenMath technologies are the essential tool ensuring the interoperability among systems (even integration in some cases). This interoperability has a vertical dimension (from the mediator to the kernel systems) as well as a horizontal axis (allowing direct interconnection of kernel systems). The modules can be taken from their sources (as in the cases of Kenzo and ACL2), invoked in a remote manner (like the GAP server, connected via the SCSCP protocol) or even developed in an ad-hoc way (as our Homotopy Expert System).

Several research lines are still open. The most important ones are related to giving more resources to the user to manage the interaction. Moreover, it would be also necessary to improve the interaction with the ACL2 system. At this moment the queries must be pre-processed; a comfortable way of introducing questions about the truth of properties of intermediary objects, dynamically generated during a computing session, should be provided. Last, and the most difficult one, a meta-language should be designed to specify how and when a new kernel system can be plugged in the framework. This capability and the necessity of orchestrating the different services suppose a real challenge, which will be explored by means of OpenMath technologies.

References

1. GAP - Groups, Algorithms, Programming - System for Computational Discrete Algebra,
`http://www.gap-system.org`

2. MathBroker: A Framework for Brokering Distributed Mathematical Services, http://www.risc.uni-linz.ac.at/projects/basic/mathbroker/
3. MathBroker II: Brokering Distributed Mathematical Services, http://www.risc.uni-linz.ac.at/projects/mathbroker2/
4. MathServe Framework, http://www.ags.uni-sb.de/jzimmer/mathserve.html
5. MATHWEB-SB:A Software Bus for MathWeb, http://www.ags.uni-sb.de/jzimmer/mathweb-sb/
6. MoNET: Mathematics on the Net, http://monet.nag.co.uk/cocoon/monet/index.html
7. Boley, H., et al.: Rule Markup Language (RuleML) version 0.91 (2004), http://ruleml.org/
8. Buschmann, F., et al.: Pattern-Oriented Software Architecture. A Pattern Language for Distributed Computing. Software Design Patterns, vol. 4. Wiley, Chichester (2007)
9. Cohen, A.M., Caprotti, O., Cuypers, H., Riem, M.N., Sterk, H.: Using OpenMath Servers for Distributing Mathematical Computations. In: Proceedings of the Fifth Asian Technology Conference in Mathematics, pp. 325–336 (2000)
10. The OpenMath Consortium. Openmath standard 2.0 (2004), http://www.openmath.org/standard/om20-2004-06-30/omstd20.pdf
11. Dousson, X., Sergeraert, F., Siret, Y.: The Kenzo program. Institut Fourier, Grenoble (1998), http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/
12. Ellis, G.: HAP package for GAP (2009), http://www.gap-system.org/Packages/hap.html
13. Freundt, S., Horn, P., Konovalov, A., Linton, S.A., Roozemond, D.: Symbolic Computation Software Composability. In: Autexier, S., Campbell, J., Rubio, J., Sorge, V., Suzuki, M., Wiedijk, F. (eds.) AISC 2008, Calculemus 2008, and MKM 2008. LNCS (LNAI), vol. 5144, pp. 285–295. Springer, Heidelberg (2008)
14. Freundt, S., Horn, P., Konovalov, A., Lindon, S., Roozemond, D.: Symbolic Computation Software Composability Protocol (SCSCP) specification, version 1.3 (2009), http://www.symbolic-computation.org/scscps
15. Giarratano, J.C., Riley, G.D.: Expert Systems: Principles and Programming. PWS Publishing Company (2005)
16. The GAP group. GAP a Tutorial, Operations and Methods, ch. 8, pp. 72–76 (2008)
17. Heras, J., Pascual, V., Rubio, J.: Mediated Access to Symbolic Computation Systems. In: Autexier, S., Campbell, J., Rubio, J., Sorge, V., Suzuki, M., Wiedijk, F. (eds.) AISC 2008, Calculemus 2008, and MKM 2008. LNCS (LNAI), vol. 5144, pp. 446–461. Springer, Heidelberg (2008)
18. Heras, J., Pascual, V., Rubio, J.: Using Open Mathematical Documents to Interface Computer Algebra and Proof Assistant Systems. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) Calculemus 2009. LNCS, vol. 5625, pp. 467–473. Springer, Heidelberg (2009)
19. Kaufmann, M., Moore, J.S.: ACL2 version 3.6 (2009), http://www.cs.utexas.edu/users/moore/acl2/
20. Kohlhase, M.: OMDoc – An open markup format for mathematical documents (Version 1.2). Springer, Heidelberg (2006)
21. Romero, A., Ellis, G., Rubio, J.: Interoperating between Computer Algebra systems: computing homology of groups with Kenzo and GAP. In: Proceedings of International Symposium on Symbolic and Algebraic Computation, pp. 303–310 (2009)
22. Smirnova, E., So, C.M., Watt, S.M.: An architecture for distributed mathematical web services. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 363–377. Springer, Heidelberg (2004)

sTeXIDE: An Integrated Development Environment for sTeX Collections

Constantin Jucovschi and Michael Kohlhase

Computer Science, Jacobs University Bremen
{c.jucovschi,m.kohlhase}@jacobs-university.de

Abstract. Authoring documents in MKM formats like OMDoc is a very tedious task. After years of working on a semantically annotated corpus of sTeX documents (GenCS), we identified a set of common, time-consuming subtasks, which can be supported in an integrated authoring environment.

We have adapted the modular Eclipse IDE into sTeXIDE, an authoring solution for enhancing productivity in contributing to sTeX based corpora. sTeXIDE supports context-aware command completion, module management, semantic macro retrieval, and theory graph navigation.

1 Introduction

Before we can manage mathematical 'knowledge' — i.e. reuse and restructure it, adapt its presentation to new situations, semi-automatically prove conjectures, search it for theorems applicable to a given problem, or conjecture representation theorems, we have to convert informal knowledge into machine-oriented representations. How exactly to support this formalization process so that it becomes as effortless as possible is one of the main unsolved problems of MKM. Currently most mathematical knowledge is available in the form of LaTeX-encoded documents. To tap this reservoir we have developed the sTeX [Koh08, sTe09] format, a variant of LaTeX that is geared towards marking up the semantic structure underlying a mathematical document.

In the last years, we have used sTeX in two larger case studies. In the first one, the second author has accumulated a large corpus of teaching materials, comprising more than 2,000 slides, about 800 homework problems, and hundreds of pages of course notes, all written in sTeX. The material covers a general first-year introduction to computer science, graduate lectures on logics, and research talks on mathematical knowledge management. The second case study consists of a corpus of semi-formal documents developed in the course of a verification and SIL3-certification of a software module for safety zone computations [KKL10a, KKL10b]. In both cases we took advantage of the fact that sTeX documents can be transformed into the XML-based OMDoc [Koh06] by the LaTeXML system [Mil10], see [KKL10a] and [DKL+10] for a discussion on the MKM services afforded by this.

These case studies have confirmed that writing sTeX is *much* less tedious than writing OMDoc directly. Particularly useful was the possibility of using the sTeX-generated PDF for proofreading the text part of documents. Nevertheless serious usability problems remain. They come from three sources:

P1 installation of the (relatively heavyweight) transformation system (with dependencies on `perl`, `libXML2`, LaTeX, the sTeX packages),
P2 the fact that sTeX supports an object-oriented style of writing mathematics, and
P3 the size of the collections which make it difficult to find reusable components.

The documents in the first (educational) corpus were mainly authored directly in sTeX via a text editor (`emacs` with a simple sTeX mode [Pes07]). This was serviceable for the author, who had a good recollection for names of semantic macros he had declared, but presented a very steep learning curve for other authors (e.g. teaching assistants) to join. The software engineering case study was a post-mortem formalization of existing (informal) LaTeX documents. Here, installation problems and refactoring existing LaTeX markup into more semantic sTeX markup presented the main problems.

Similar authoring and source management problems are tackled by Integrated Development Environments (IDEs) like Eclipse [Ecl08], which integrate support for finding reusable functions, refactoring, documentation, build management, and version control into a convenient editing environment. In many ways, sTeX shares more properties with programming languages like Java than with conventional document formats, in particular, with respect to the three problem sources mentioned above

S1 both require a build step (compiling Java and formatting/transforming sTeX into PDF/OMDoc),
S2 both favor an object-oriented organization of materials, which allows to
S3 build up large collections of re-usable components

To take advantage of the solutions found for these problems by software engineering, we have developed the sTeXIDE integrated authoring environment for sTeX-based representations of mathematical knowledge. In the next section we recap the parts of sTeX needed to understand the system. In Section 3 we present the user interface of the sTeXIDE system, and in Section 4 we discuss implementation issues. Section 5 concludes the paper and discusses future work.

2 sTeX: Object-Oriented LaTeX Markup

The main concept in sTeX is that of a "*semantic macro*", i.e. a TeX command sequence $\mathcal{S}$ that represents a meaningful (mathematical) concept or object $\mathcal{O}$: the TeX formatter will expand $\mathcal{S}$ to the presentation of $\mathcal{O}$. For instance, the command sequence `\positiveReals` is a semantic macro that represents a mathematical symbol — the set $\mathbb{R}^+$ of positive real numbers. While the use of semantic macros is generally considered a good markup practice for scientific documents[1],

[1] For example, because they allow adapting notation by macro redefinition and thus increase reusability.

regular TEX/LATEX does not offer any infrastructural support for this. sTEX does just this by adopting a semantic, "object-oriented" approach to semantic macros by grouping them into "modules", which are linked by an "imports" relation. To get a better intuition, consider the example in listing 1.

Listing 1. An sTEX module for Real Numbers

```
\begin{module}[id=reals]
  \importmodule[../background/sets]{sets}
  \symdef{Reals}{\mathcal{R}}
  \symdef{greater}[2]{#1>#2}
  \symdef{positiveReals}{\Reals^+}
  \begin{definition}[id=posreals.def,title=Positive Real Numbers]
    The set $\positiveReals$ is the set of $\inset{x}\Reals$ such that $\greater{x}0$
  \end{definition}
  ...
\end{module}
```

which would be formatted to

Definition 2.1 (Positive Real Numbers):
The set $\mathbb{R}^+$ is the set of $x \in \mathbb{R}$ such that $x > 0$

Note that the markup in the module `reals` has access to semantic macro `\inset` (membership) from the module `sets` that was imported by the document by `\importmodule` directive from the `../background/sets.tex`. Furthermore, it has access to the `\defeq` (definitional equality) that was in turn imported by the module `sets`.

From this example we can already see an organizational advantage of sTEX over LATEX: we can define the (semantic) macros close to where the corresponding concepts are defined, and we can (recursively) import mathematical modules. But the main advantage of markup in sTEX is that it can be transformed to XML via the LATEXML system [Mil10]: Listing 2 shows the OMDoc [Koh06] representation generated from the sTEX sources in listing 1.

Listing 2. An XML Version of Listing 1

```
<theory xml:id="reals">
 <imports from="../background/sets.omdoc#sets"/>
 <symbol xml:id="Reals"/>
 <notation>
   <prototype><OMS cd="reals" name="Reals"/></prototype>
     <rendering><m:mo>ℝ</m:mo></rendering>
 </notation>
  <symbol xml:id="greater"/><notation>...</notation>
  <symbol xml:id="positiveReals"/><notation>...</notation>
  <definition xml:id="posreals.def" for="positiveReals">
    <meta property="dc:title">Positive Real Numbers</meta>
    The set <OMOBJ><OMS cd="reals" name="postiveReals"/></OMOBJ> is the set ...
  </definition>
  ...
</theory>
```

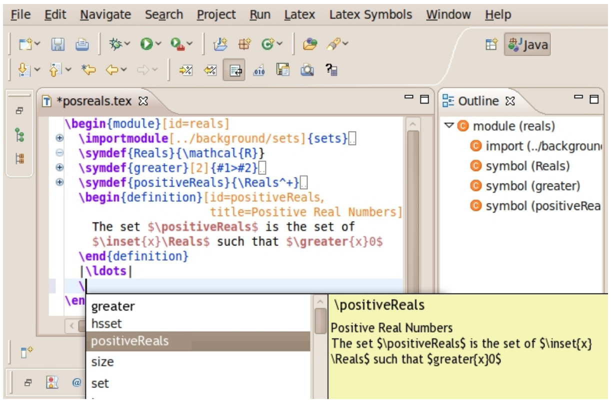
One thing that stands out from the XML in this listing is that it incorporates all the information from the sTEX markup that was invisible in the PDF produced by formatting it with TEX.

3 User Interface Features of sTeXIDE

One of the main priorities we set for sTeXIDE is to have a relatively gentle learning curve. As the first experience of using a program is running the installation process, we worked hard to make this step as automated and platform independent as possible. We aim at supporting popular operating systems such as Windows and Unix based platforms (Ubuntu, SuSE). Creating an OS independent distribution of Eclipse with our plugin preinstalled was a relatively straightforward task; so was distributing the plugin through an update site. What was challenging was getting the 3rd party software (`pdflatex`, `svn`, `latexml`, `perl`) and hence OS specific ports installed correctly.

After installation we provide a new project wizard for sTeX projects which lets the user choose the output format (`.dvi`, `.pdf`, `.ps`, `.omdoc`, `.xhtml`) as well as one of the predefined sequences of programs to be executed for the build process. This will control the Eclipse-like workflow, where the chosen 'outputs' are rebuilt after every save, and syntactic (as well as semantic) error messages are parsed, cross-referenced, and displayed to the user in a collapsible window. The wizard then creates a stub project, i.e. a file `main.tex` which has the structure of a typical sTeX file but also includes `stex` package and imports a sample module defined in `sample_mod.tex`.

sTeXIDE supports the user in creating, editing and maintaining sTeX documents or corpora. For novice users we provide templates for creating modules, imports and definitions. Later on, the user benefits from *context-aware autocompletion*, which assists the user in using valid LaTeX and sTeX macros. Here, by valid macros, we mean macros which were previously defined or imported (both directly or indirectly) from other modules. Consider the sample sTeX source in listing 1. At the end of the first line, one would only be able to autocomplete LaTeX macros, whereas at the end of the second line, one would already have macros like `\inset` from the imported `sets` module (see Fig. 1). Note that we also make use of the semantic structure of the sTeX document in listing 1 for



```
\begin{module}[id=reals]
  \importmodule[../background/sets]{sets}
  \symdef{Reals}{\mathcal{R}}
  \symdef{greater}[2]{#1>#2}
  \symdef{positiveReals}{\Reals^+}
  \begin{definition}[id=positiveReals,
                     title=Positive Real Numbers]
    The set $\positiveReals$ is the set of
    $\inset{x}\Reals$ such that $\greater{x}0$
  \end{definition}
  |\ldots|
  \
\en
```

```
\positiveReals
Positive Real Numbers
The set $\positiveReals$ is the set of $\inset{x}
\Reals$ such that $greater{x}0$
```

Fig. 1. Context aware autocompletion feature for semantic macros

explanations. Namely, the macro `\positiveReals` is linked to its definition via the key `for=positiveReals`, this makes it possible to display the text of the definition as part of macro autocompletion explanation (the yellow box).

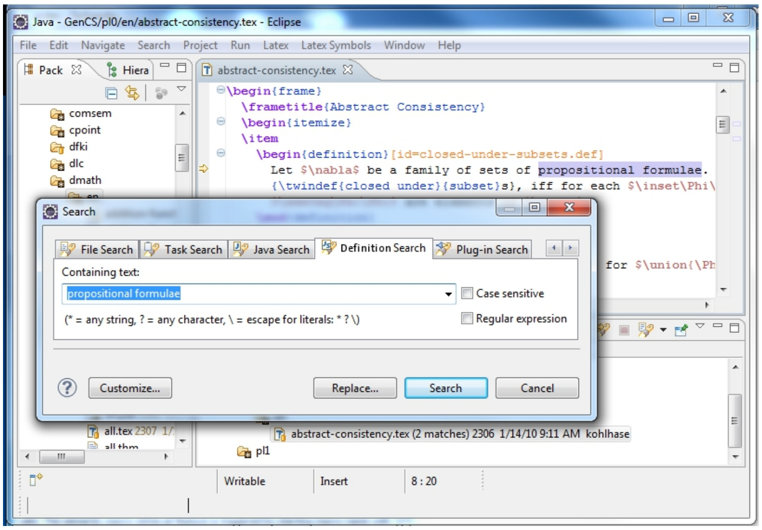
Similarly, *semantic macro retrieval* (triggered by typing '`\*`') will suggest all available macros from all modules of the current project. In case that the auto-completed macro is not valid for the current context, sTeXIDE will insert the required import statement so that the macro becomes valid.

Moreover, sTeXIDE supports several typical document/collection maintenance tasks: Supporting *symbol and module name refactoring* is very important as doing it manually is both extremely error-prone and time consuming, especially if two different modules define a symbol with the same name and only one of them is to be renamed. The *module splitting* feature makes it easier for users to split a larger module intro several semantically self contained modules which are easier to be reused. This feature ensures that imports required to make the newly created module valid are automatically inserted.

$$\begin{tikzcd} C & & B \arrow[ll, dotted] \\ & A \arrow[ul, dashed] \arrow[ur] & \end{tikzcd}$$

At last, *import minimization* creates warnings for unused or redundant `\importmodule` declarations and suggests removing them. Consider for instance the situation on the right, where modules `C` and `B` import module `A`. Now, if we add a semantic macro in `C` that needs an import from `B`, then we should replace the import of `A` in `C` with one of `B` instead of just adding the latter (i.e. we would replace the dashed by the dotted import).

Three additional features make navigation and information retrieval in big corpora easier. *Outline view* of the document (right side of figure 1) displays main semantic structures inside the current document. One can use outline tree layout to copy, cut and navigate to areas represented by the respective



```
\begin{frame}
  \frametitle{Abstract Consistency}
  \begin{itemize}
  \item
    \begin{definition}[id=closed-under-subsets.def]
      Let $\nabla$ be a family of sets of propositional formulae.
      {\twindef{closed under}{subset}s}, iff for each $\inset\Phi\
                                               for $\union{\Ph
```

Containing text: propositional formulae

☐ Case sensitive

☐ Regular expression

(* = any string, ? = any character, \ = escape for literals: * ? \)

abstract-consistency.tex (2 matches) 2306 1/14/10 9:11 AM kohlhase

Fig. 2. Macro Retrieval via Mathematical Concepts

structures. In case of imports one can navigate to imported modules. *Theory graph navigation* is another feature of sTeXIDE. It creates a graphical representation of how modules are related through imports. This gives the author a chance to get a better intuition for how concepts and modules are related. The last feature is the *semantic macro search feature*. The aim of this feature is to search for semantic macros by their mathematical descriptions, which can be entered into the search box in figure 2. The feature then searches definitions, assumptions and theorems for the query terms and reports any `\symdef`-defined semantic macros 'near' the hits. This has proved very convenient in situations where the macro names are abbreviated (e.g. `\sconcjuxt` for "string concatenation by juxtaposition") or if there are more than one name for a mathematical context (e.g. "concatenation" for `\sconcjuxt`) and the author wants to re-use semantic macros defined by someone else.

4 Implementation

The implementation of sTeXIDE is based on the TeXlipse [TeX08] plugin for Eclipse. This plugin makes use of Eclipse's modular framework (see Fig. 3 3) and provides features like syntax highlighting, code folding, outline generation, autocompletion and templating mechanisms. Unfortunately, TeXlipse uses a parser which is hardwired for a fixed set of LaTeX macros like `\section`, `\input`, etc. which made it quite challenging to generalize it to sTeX specific macros. Therefore we had to reimplement parts of TeXlipse so that sTeX macros like `\symdef` and `\importmodule` that extend the set of available macros can be treated specially. We have underlined all the parts of TeXlipse we had to extend or replace in Figure 3.

To support context sensitive autocompletion and refactoring we need to know the exact position in the source code where modules and symbols are defined.

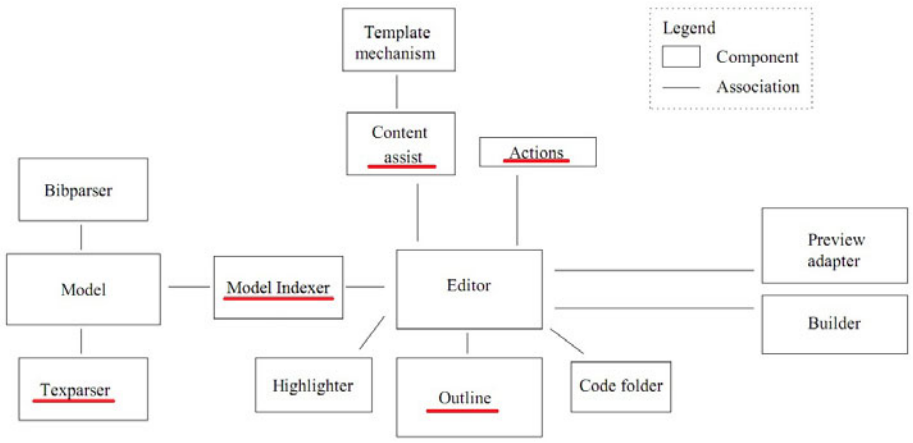


Fig. 3. Component architecture of TeXlipse (adapted from [TeX10])

Running a fully featured LaTeX parser like LaTeXML proved to be too slow (sometimes taking 5-10 sec to compile a document of 15 pages) and sensitive to errors. For these reasons, we implemented a very fast but naïve LaTeX parser which analyses the source code and identifies commands, their arguments and options. We call this parser naïve because it parses only one file a time (i.e. inclusions, and styles are not processed) and macros are not expanded. We realize the parse tree as an in-memory XML DOM to achieve format independence (see below). Then we run a set of semantic spotters which identify constructs like module and import declarations, inclusions as well as sections/subsections etc, resulting in an index of relevant structural parts of the sTeX source identified by unique URIs and line/column number ranges in the source. For example, a module definition in sTeX begins with `\begin{module}[id=module_id]` and ends in a `\end{module}`, so the structure identifying a module will contain these two ranges.

Note that the LaTeX document model (and thus that of sTeX) is a tree, so two spotted structure domains are either disjoint or one contains the other, so we implement a range tree we use for efficient change management: sTeX IDE implements a class which listens to changes made in documents, checks if they intersect with the important ranges of the spotted structures or if they introduce new commands (i.e. start with '\'). If this does not hold, the range tree is merely updated by calculating new line and column numbers. Otherwise we run the naïve LaTeX parser and the spotters again.

Our parser is entirely generated by a JavaCC grammar. It supports error recovery (essential for autocompletion) and does not need to be changed if a new macro needs to be handled: Semantic Spotters can be implemented as XQueries, and our parser architecture provides an API for adding custom made semantic spotters. This makes the parser extensible to new sTeX features and allows working around the limitation of the naïve LaTeX parser of not expanding macros.

We implemented several indexes to support features mentioned in section 3. For theory navigation we have an index called `TheoryIndex` which manages a directed graph of modules and import relationships among them. It allows *a)* retrieving a list of modules which import/are imported by module X *b)* checking if module X is directly/indirectly imported by module Y. `SymdefIndex` is another index which stores pairs of module URIs and symbols defined in those modules. It supports fast retrieving of (symbol, module) pairs where a symbol name starts with a certain prefix by using a trie data structure. As expected, this index is used for both context aware autocompletion as well as semantic macro retrieval features. The difference is that context aware autocompletion feature also filters the modules not accessible from current module by using the `TheoryIndex`. Refactoring makes use of an index called `RefIndex`. This index stores (module URI, definition module URI, symbol name) triples for all symbol occurrences (not just definitions as in `SymdefIndex`). Hence when the author wants to rename a certain symbol, we first identify where that symbol is defined (i.e. its definition module URI) and then query for all other symbols with same name and definition module URI.

5 Conclusion and Future Work

We have presented the sTeXIDE system, an integrated authoring environment for sTeX collections realized as a plugin to the Eclipse IDE. Even though the implementation is still in a relatively early state, this experiment confirmed the initial expectation that the installation, navigation, and build support features contributed by Eclipse can be adapted to a useful authoring environment for sTeX with relatively little effort. The modularity framework of Eclipse and the TeXlipse plugin for LaTeX editing have been beneficial for our development. However, we were rather surprised to see that a large part of the support infrastructure we would have expected to be realized in the framework were indeed hard-coded into the plugins. This has resulted in un-necessary re-implementation work.

In particular, system- and collection-level features of sTeXIDE like automated installation, PDF/XML build support, and context-sensitive completion of command sequences, import minimziation, navigation, and concept-based search have proven useful, and are not offered by document-oriented editing solutions. Indeed such features are very important for editing and maintaining any MKM representations. Therefore we plan to extend sTeXIDE to a general “MKM IDE”, which supports more MKM formats and their human-oriented front-end syntaxes (just like sTeX serves a front-end to OMDoc in sTeXIDE).

The modular structure of Eclipse also allows us to integrate MKM services (e.g. information retrieval from the background collection or integration of external proof engines for formal parts [ALWF06]; see [KRZ10] for others) into this envisioned “MKM IDE”, so that it becomes a “rich collection client” to *a universal digital mathematics library (UDML)*, which *would continuously grow and in time would contain essentially all mathematical knowledge* envisioned as the Grand Challenge for MKM in [Far05].

In the implementation effort we tried to abstract from the sTeX surface syntax, so that we anticipate that we will be able to directly re-use our spotters or adapt them for other surface formats that share the OMDoc data model. The next target in this direction is the modular LF format introduced in [RS09]. This can be converted to OMDoc by the TWELF system, which makes its treatment directly analogous to sTeX, this would provide a way of information sharing among different authoring systems and workflows.

References

[ALWF06] Aspinall, D., Lüth, C., Winterstein, D., Fayyaz, A.: Proof general in eclipse. In: Eclipse Technology eXchange ETX 2006, ACM Press, New York (2006)

[DKL$^+$10] David, C., Kohlhase, M., Lange, C., Rabe, F., Zhiltsov, N., Zholudev, V.: Publishing math lecture notes as linked data. In: Aroyo, L., Antoniou, G., Hyvönen, E. (eds.) ESWC 2010. LNCS. Springer, Heidelberg (June 2010)

[Ecl08] Eclipse: An open development platform (May 2008), `http://www.eclipse.org/`

[Far05] Farmer, W.M.: Mathematical Knowledge Management. In: Schwartz, D.G. (ed.) Encyclopedia of Knowledge Management, pp. 599–604. Idea Group Reference (2005)

[KKL10a] Kohlhase, A., Kohlhase, M., Lange, C.: Dimensions of formality: A case study for MKM in software engineering. In: Autexier, S., et al. (eds.) AISC/Calculemus/MKM 2010. LNCS (LNAI), vol. 6167, pp. 355–369. Springer, Heidelberg (2010)

[KKL10b] Kohlhase, A., Kohlhase, M., Lange, C.: sTeX – a system for flexible formalization of linked data. submitted to I-SEMANTICS 2010 (2010)

[Koh06] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents (Version 1.2)*. Number 4180 in LNAI. Springer Verlag, August 2006.

[Koh08] Kohlhase, M.: Using LaTeXas a semantic markup format. Mathematics in Computer Science 2(2), 279–304 (2008)

[KRZ10] Kohlhase, M., Rabe, F., Zholudev, V.: Towards mkm in the large: Modular representation and scalable software architecture. In: Autexier, S., et al. (eds.) AISC/Calculemus/MKM 2010. LNCS (LNAI), vol. 6167, pp. 370–384. Springer, Heidelberg (2010)

[Mil10] Miller, B.: LaTeXML: A LaTeXto XML converter (March 2010), `http://dlmf.nist.gov/LaTeXML/`

[Pes07] Pesikan, D.: Coping with content representations of mathematics in editor environments: nOMDoc mode. Bachelor's thesis, Computer Science, Jacobs University, Bremen (2007)

[RS09] Rabe, F., Schürmann, C.: A Practical Module System for LF. In: Proceedings of the Workshop on Logical Frameworks Meta-Theory and Practice, LFMTP (2009)

[sTe09] Semantic Markup for LaTeX (July 2009), `http://kwarc.info/projects/stex/`

[TeX08] Texlipse: Adding latex support to the eclipse ide (May 2008), `http://texlipse.sourceforge.net/`

[TeX10] T-76.115 technical specification. texlipse project (March 2010), `http://prdownloads.sourceforge.net/texlipse/texlipse-techspec-1.0.0.pdf?download`

Proofs, Proofs, Proofs, and Proofs

Manfred Kerber

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

Abstract. In logic there is a clear concept of what constitutes a proof and what not. A proof is essentially defined as a finite sequence of formulae which are either axioms or derived by proof rules from formulae earlier in the sequence. Sociologically, however, it is more difficult to say what should constitute a proof and what not. In this paper we will look at different forms of proofs and try to clarify the concept of proof in the wider meaning of the term. This has implications on how proofs should be represented formally.

1 Introduction

There is a relatively clear definition of what a *proof* is. The concept has been clarified in the last 150 years with the development of logic and in the last 50 to 60 years with the development of systems which formalize the results of these investigations in formal computer systems. Mathematicians, however, seem not to have much changed their view of proofs.[1] Sure, they have some knowledge of the results in foundations but if they work in fields such as statistics, group theory, or geometry then the formalization of proof is only of marginal interest, although the concept of proof itself is at the core of the whole of mathematics.

Is this just a matter of ignorance? Or rather of professionalism? And what are the consequences for our field which tries to offer support for mathematicians?

In order to approach these questions an account of the development of the concept of proof in different fields is given. We first take a look at the development in logic (section 2)[2]. Next we see the consequences this had on working mathematicians and their attitude towards formal proofs (section 3). The development in logic has strongly influenced the development of deduction systems. In section 4 we take a brief look at deduction systems. Then some consequences for the field of mathematical knowledge management are discussed. Essentially we argue that the representation of proofs should be flexible enough to serve different purposes in order to be able to communicate proofs at different levels: checkable proofs, abstract high-level proofs, proof ideas/plans, and false proofs.

[1] Obviously for those working on the foundations of mathematics this is different. The generalization 'mathematicians' does not mean all mathematicians, but most typical mathematicians.

[2] The claim is not that there is a single view in the different fields. Even a single person may have different views at different times or in different contexts.

2 The Logician's View

In the second half of the 19th century and in the early 20th century, a rigorous definition of the concept of proof was given. Inspired by the rigorous work of Euclid, Hilbert axiomatized geometry and developed a programme to be carried through for all of mathematics. Whitehead and Russell wrote the Principia Mathematica [23] which started to implement the grand vision to reduce all of mathematics to logic. Logicians like Boole and Frege developed propositional and predicate logic, and Gentzen the calculus of Natural Deduction.

As Frege put it, the vision was ([7] quoted from [10, p.6f]):

> *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.*

Hilbert's definition of proof as a sequence of formulae which are either axioms or generated by rules from elements coming earlier in the sequence is now quite standard in logic books. Natural Deduction calculi as introduced by Gentzen (see, e.g., [19]) are an extension of this definition. For instance, Andrews [2, p.11] defines strictly formally the notions of a proof (from hypotheses and then a proof of a well-formed formula (wff)). Then he defines *"A theorem is a wff which has a proof."*

On a more philosophical level there has been a dispute what should constitute a rigorous proof, since even with this clarification the question was not fully settled. Most notably there was a dispute between Hilbert and Brouwer on the right approach to mathematics, in which the question of constructive proofs versus 'classical' proofs played a major role. Hilbert wanted, in particular, defend the 'paradise' of (infinite) sets provided by Cantor, whereas Brouwer was wary about the concept of infinity and insisted on constructiveness. At the time, Brouwer's view was considered by many mathematicians as too restrictive (and probably is still by many today). With the advent of computers the idea of constructive proofs, however, gained great attraction since proving and programming became the same activity. For details of the dispute see [10]. There are other disputes, e.g., about the axiom of choice and about the rigour of diagrams in reasoning.

For the argument here, it suffices to state that even in the rigorous area of logical foundations there can be dispute about what should constitute a proof and what not.[3]

3 The Mathematician's View

Mathematicians seem to have ignored the development in formal logic to a large degree. The start of the rapid development of modern logic can be put roughly to the mid 19th century. However, the start of the rapid development of modern mathematics is about 200 years older.[4]

As Kline [12, p.256] notes, the "*Bourbakists expressed their position on logic in an article in the Journal of Symbolic Logic (1949): 'In other words, logic, so far as we mathematicians are concerned, is no more and no less than the grammar of the language which we use, a language which had to exist before the grammar could be constructed.' Future developments in mathematics may call for modifications of the logic. This had happened with the introduction of infinite sets and, ... it would happen again.*"

In a similar line, Bourbaki ([4] quoted from [12, p.320]) doubts that one of the goals of logicians, to make mathematics free from contradictions, is feasible:

> *Historically speaking, it is of course quite untrue that mathematics is free from contradictions; non-contradiction appears as a goal to be achieved, not as a God-given quality that has been granted to us once for all. Since the earliest times, all critical revisions of the principles of mathematics as a whole, or of any branch in it, have almost invariably followed periods of uncertainty, where contradictions did appear and had to be resolved. ... There are now twenty-five centuries during which the mathematicians have had the practice of correcting their errors and thereby seeing their science enriched, not impoverished; this gives them the right to view the future with serenity.*

Theorems with their proofs are at the core of mathematics and play a significant role in the working of mathematicians. Hardy describes in § 12 of [9] two examples of theorems (with proofs) which he calls 'first-rate': First the theorem that there are infinitely many prime numbers (the proof is indirect, assume that you have finitely many, multiply them all and add 1. The new number is not divisible by any prime number, which gives a contradiction.) and second the theorem that $\sqrt{2}$ is irrational (again an indirect proof: assume $\sqrt{2} = a/b$ with a and b two integers which have no common divisor. Then $2 \cdot b^2 = a^2$. It follows that a^2 and hence a must be even, but then b must be even as well, which gives a contradiction).

[3] It should be noted, however, that there is a clear language and it is very clear to the participants of a dispute what they are talking about.

[4] And of course there are always precursors, Aristotle as the father of logic, Archimedes who was close to inventing calculus almost 2000 years before Newton and Leibniz.

In § 18 Hardy states then:

> *In both theorems (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.*

Proofs often follow established patterns. Often they are invented, doubted by other, later generally accepted, and finally re-used, taught, and generally recognized. Examples are the ϵ-δ criterion (to establish continuity), diagonalization (to establish the impossibility of certain properties, e.g. halting problem, incompleteness, uncountability), mathematical induction (to reason about infinite structures), infinitesimals (to reason about differentiation).

From a logicians point of view mathematical proofs are more like proof plans. This is reflected in the education of mathematics. Proof principles such as the ϵ-δ criterion are taught in lectures, without a strictly formal treatment. Many of these principles are even taught in concrete proofs which have exemplary character and can be generalized later on to many other cases. Formal logic, however, is not necessarily part of the education of a mathematician. In consequence, the concept of a proof is much less strict, and 'mathematics is a MOTLEY of techniques of proof' as Wittgenstein put it [24, p. 176f].[5]

This means that the concept of proof is not fixed once and for all but requires the possibility for extension. Practically, mathematicians treat proofs and proof methods as first class objects, that is, just as they introduce new concepts they may introduce new proof principles, describe them and then use them. For this reason mathematicians focus on their special fields of expertise and consider the study of logic as one field among others. And if this field is not their specialty and particular area of expertise then they do what professionals do with fields they consider only as marginally relevant: they give it only marginal attention.

4 The Deductionist's View

In formal communities such as the theorem proving community, the logicians' view of the concept of proof has (for good reasons) been predominant, but not been the only view. Davis distinguishes two communities, the logic oriented and the cognitive oriented communities.

[5] Still there is a general assumption in mathematics that in principle it is possible to extend these mathematical proofs (proof plans) to full logic level proofs if necessary.

Automated Theorem Proving

As Davis [6, p.5] states:

With the ready availability of serious computer power, deductive reasoning, especially as embodied in mathematics, presented an ideal target for those interested in experiments with computer programs that purported to implement the "higher" human faculties. This was because mathematical reasoning combines objectivity with creativity in a way difficult to find in other domains. For this endeavor, two paths presented themselves. One was to understand what people do when they create proofs and write programs emulating that process. The other was to make use of the systematic work of the logicians reducing logical reasoning to standard canonical forms on which algorithms could be based.

Since the groundbreaking work in logic in the early 20th century was very close to implementation it led to the dream to build machines that can solve hard problems fully automatically. The invention of the resolution principle by Robinson [21] which made search spaces finitely branching was a great breakthrough and led to the possibility to prove many theorems fully automatically. In parallel there was a smaller community which was interested in the cognitive aspects of theorem proving (by Newell and others, followed up in the proof planning work by Bundy and others). At least motivationally the work is linked to psychological evidence [20] that deductive reasoning plays a very important role in human intelligence and that some proof rules like Modus ponens are universally accepted while others are accepted only by a minority. Related in this context is also the work on diagrammatic reasoning (see, e.g. [11]) which shows that reasoning falsely considered for some time as inferior, can be made very precise.

In general, however, the dream of full automation has not come true at large. There are fascinating exceptions such as the proof of the Robbins problem ([15]), but still mathematicians do not have theorem proving machines on their desks which they use to a similar degree as they use typesetting programs or computer algebra systems. And possibly not everybody would want such a machine, since as Hardy put it in [9, § 10] "*there is nothing in the world which pleases even famous men ... quite so much as to discover, or rediscover, a genuine mathematical theorem.*" and we can add "*and a genuine mathematical proof.*" (but proofs are parts of the theorem for Hardy anyway.) Why leaving the fun to a machine?

There has been a different community at least since the 1960s, namely the community interested in being able to check proofs by a machine. On first sight this looks like a much less ambitious goal, but it turned out to be actually much more difficult than anticipated. We will look at this next.

Proof Checking

The perhaps two most prominent approaches to proof checking – from which other systems have been derived – are Automath [5] and Mizar [22]. The goal is not to find proofs automatically but to check proofs. De Bruijn summarizes his dream in 1994 [16, p.210] as follows:

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.

In many ways this dream is more exciting, since firstly it looks more feasible and secondly it is something mathematicians and professionals working in related fields can appreciate more. Although proof checkers have been extended by useful extensions which allow for higher-level proofs, most notably by tactics which allow to reduce many steps in a proof to a single user interaction, even 16 years after de Bruijn's retrospective we are still not there and mathematicians do not widely use the corresponding systems. However, they can be used and some do use them, most notably there is the Flyspeck Project [8] in which Hales (and others) are formalizing his proof of the Kepler conjecture.

5 How to Make Systems More Accepted?

Systems which deal with proofs can be built for different purposes and different purposes result in different requirements. Only some of them are currently adequately supported by mathematical knowledge management systems. Let us look at the most common purposes/contexts in which proofs are communicated:

Education: In an educational context proofs will be presented and/or jointly developed in order to teach the concept. A teacher may want to teach how to find a proof but more typically will teach how to write up a proof so that it is of an acceptable standard. These are two different modes as Pólya [18, p. vi], pointed out:

We secure our mathematical knowledge by demonstrative reasoning, *but we support our conjectures by* plausible reasoning ... *Demonstrative reasoning is safe, beyond controversy, and final. Plausible reasoning is hazardous, controversial, and provisional. ... In strict reasoning the principal thing is to distinguish a proof from a guess, a valid demonstration from an invalid attempt. In plausible reasoning the principal thing is to distinguish a guess from a guess, a more reasonable guess from a less reasonable guess. ... [plausible reasoning] is the kind of reasoning on which his [a mathematician's] creative work will depend.*

Proof development: Here the scenario is that of a mathematician or a group of mathematicians developing a proof. They do not know yet the details of the proof (or even whether there is a proof), they may have some ideas which may be vague and informal. A blackboard and a piece of chalk seem to be the tools of choice and systems at best offer the functionality of a blackboard and chalk. In Pólya's words, the game is mainly about plausible reasoning at this stage. The task of providing support is particularly challenging since proof attempts, ideas, partial proof plans may have to be communicated.

Automation: If automation is the objective and proofs are found, automated theorem provers typically can document a formal proof object which can be independently checked. This object can be communicated.

Correctness: If correctness of arguments is sought then proofs must be checkable. At a calculus level the different formal systems implemented allow this. Human mathematicians, however, can check proofs at a less formal level. Support at this level is still patchy, although important steps have been made in an area which is labelled as the development of a mathematical vernacular (going back to de Bruijn and the Automath project, and continued by Nederpelt and Kamareddine [17]).

For any of the different activities there is the question: What kind of information is necessary and how should it be represented?

A proof is an argument that should convince the reader (interpreter) of the truth of a statement (certain axioms and assumptions given). That is, a proof is a relationship between the argument and the reader, and the reader has to come with some level of knowledge.

If we know a lot, then a proof can be more concise. If we know the theorem already then we do not need to be convinced. If we know little, then we need a detailed argument which convinces us beyond reasonable doubt (some may say beyond any doubt) of the correctness of the theorem. In this respect a proof is a proof only with respect to a receiver/reader. "*Nothing can be explained to a stone, the reader must understand something beforehand.*" as McCarthy formulated it (1964, p.7), quoted from [1, p.8] and analogously we can state that "*Nothing can be explained to God, since he understands everything beforehand.*" or as Ayer [3, p.85f] put it:

> *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 expression 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.*

Typically, we are in between the stone and God: We know certain theorems and proofs and are happy to accept certain arguments when they are mentioned in a new proof and others not. We can fill in certain gaps, but not others. We have intelligence which goes beyond checking substitutions and matching, which can convince us that a theorem is really a theorem. A proof should give us a good reason why we should not doubt the correctness of the theorem at an appropriate level. Going back to a logic level proof is typically like being dragged on a level on which we do not see the wood for the trees.

Indeed proofs come in various formats, they can be presented at different levels of abstraction and can be quite different in style and details. In order to represent and support them appropriately we need to know what they are needed for and have to reflect the purpose and the level of understanding and knowledge of the reader. The reader may know the proof already or know a similar proof (and would be quite quick at understanding the new one). The reader may have no intuition – possibly the statement is even counter intuitive – and would have to check steps slowly and carefully. Or the reader may not be able to understand the proof in a reasonable amount of time at all since they lack the corresponding knowledge and would require a significant course in a whole field of mathematics before they can appreciate the arguments.[6]

In a familiar area, mathematicians know which arguments to accept and where to be careful. They are well aware of fallacies to avoid, that is, we have positive and negative information at our disposal and avoid the fallacies as they are described by Maxwell in [14]. Maxwell's examples deal, for instance, with non-apparent divisions by zero, with problems with integration by parts, and with incorrectly drawn auxiliary diagrams in geometric proofs. Maxwell distinguishes between fallacies, where things go wrong on a deeper level (and proof checking on a high-level may wrongly succeed) and howlers, where the incorrectness of the argument is apparent (and a wrong argument may still give the correct result).

That the mathematical notion of a proof is subject to change, not strictly formal, and not beyond doubt has most convincingly been described by Lakatos [13] in an analysis of the history of the Euler polyhedron theorem, which had an exciting history of proofs and subsequent counterexamples, which led to improved proofs and more sophisticated counterexamples. This has not led to a general distrust in proofs. Although the theorem is not central to mathematics, still – as Hardy put it [9, § 12] – *a mathematician offers the game*, and a contradiction may cast doubt on the correctness of mathematics as a whole. However, the sequence of proof, counterexample, proof can be seen very much in the spirit of the quote in section 3 of Bourbaki that "*mathematicians have had the practice of correcting their errors and thereby seeing their science enriched, not impoverished.*"

Theorem proving and checking proofs is a social activity and in a highly specialized society there are different reasons why we believe a theorem and its proof. Only few will actually have the knowledge, the capacity, and the time

[6] Obviously the borders are not sharp. We may know a similar proof, and actually we would not remember every single step. Having a good intuition, having some intuition, and having no intuition, or a counter intuition is again fluid. The proof of the Robbins problem was so hard for humans since they did not have an intuition of the Robbins algebras.

to understand complicated proofs like that of the Fermat-Wiles theorem or the Kepler conjecture. Still most of us will accept that there are proofs and that the theorems hold. The two theorems mentioned by Hardy, however, have much simpler proofs and it belongs to the folklore to know their proofs.

We see that there is a broad spectrum of proofs. Typically natural language in combination with diagrams is used to store and communicate proofs. Some types of proof (formal logical proofs, some types of proof plans) can be represented in a format which is better suited to mechanical manipulation (e.g. to proof checking) than natural language. Other types are still difficult to formalize. Work on the mathematical vernacular is certainly useful in order to formalize the variety of proofs. An advanced approach to understand informal proofs at a linguistic level has been carried through by Zinn [25]. He analyzes the linguistic structure of proofs and builds internal structures, which reflect the inner logic of the proofs. This opens a way to understanding and checking informal mathematical discourse.

6 Summary

Proofs come at different levels and with different intentions. They are written for readers/checkers who/which must have certain competences. A human mathematician who knows a theorem very well knows and can communicate proofs of it at different levels: the gist of it, which allows other experts to reconstruct a full proof, a proof plan for a less proficient reader/checker, and a low level proof for a checker with little information in the field. Likewise an expert can understand proofs on different levels.

A system that has deep knowledge about proofs would be able to link the different levels. Achieving such a human level of expertise looks AI-hard unfortunately. On the other hand this has its attraction as Davis states, since it "*combines objectivity with creativity*." (Generalized) proof plans can offer a framework which is general enough to capture the different levels. Linking different levels and understanding different levels simultaneously will remain a hard problem for some time, and proof will remain a colourful concept.

References

1. Abrams, P.S.: An APL machine. SLAC-114 UC-32 (MISC). Stanford University, Stanford, California (1970)
2. Andrews, P.B.: An Introduction to Mathematical Logic and Type Theory: To Truth through Proof. Academic Press, Orlando (1986)
3. Ayer, A.J.: Language, Truth and Logic, 2nd edn., 1951 edn. Victor Gollancz Ltd. London (1936)
4. Bourbaki, N.: Théorie des ensembles. In: Éléments de mathématique, Fascicule 1, Hermann, Paris, France (1954)
5. de Bruijn, N.G.: A survey of the project Automath. In: Seldin, J.P., Hindley, J.R. (eds.) To H.B. Curry - Essays on Combinatory Logic, Lambda Calculus and Formalism, pp. 579–606. Academic Press, London (1980)

6. Davis, M.: The early history of automated deduction. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, pp. 5–14. Elsevier Science, Amsterdam (2001)
7. Frege, G.: Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle (1879)
8. Hales, T.: The Flyspek Project (2010), `http://code.google.com/flyspeck/`
9. Hardy, G.: A Mathematician's Apology. Cambridge University Press, London (1940)
10. van Heijenoort, J. (ed.): From Frege to Gödel – A Source Book in Mathematical Logic, 1879-1931. Harvard Univ. Press, Cambridge (1967)
11. Jamnik, M.: Mathematical Reasoning with Diagrams: From Intuitions to Automation. CSLI Press, Stanford (2001)
12. Kline, M.: Mathematics – The Loss of Certainty. Oxford University Press, New York (1980)
13. Lakatos, I.: Proofs and Refutations. Cambridge University Press, Cambridge (1976)
14. Maxwell, E.A.: Fallacies in Mathematics. Cambridge University Press, Cambridge (1959)
15. McCune, W.: Solution of the Robbins problems. Journal of Automated Reasoning 19(3), 263–276 (1997), `http://www.mcs.anl.gov/home/mccune/ar/robbins/`
16. Nederpelt, R., Geuvers, H., de Vrijer, R. (eds.): Selected Papers on Automath. Studies in Logic and the Foundations of Mathematics, vol. 133. North-Holland, Amsterdam (1994)
17. Nederpelt, R., Kamareddine, F.: An abstract syntax for a formal language of mathematics. In: The Fourth International Tbilisi Symposium on Language, Logic and Computation (2001), `http://www.cedar-forest.org/forest/papers/conference-publications/tbilisi01.ps`
18. Pólya, G.: Mathematics and Plausible Reasoning. Princeton University Press, Princeton (1954); Two volumes, Vol. 1: Induction and Analogy in Mathematics, Vol. 2: Patterns of Plausible Inference
19. Prawitz, D.: Natural Deduction – A Proof Theoretical Study. Almqvist & Wiksell, Stockholm (1965)
20. Rips, L.J.: The Psychology of Proof – Deductive Reasoning in Human Thinking. The MIT Press, Cambridge (1994)
21. Robinson, J.A.: A machine oriented logic based on the resolution principle. Journal of the ACM 12, 23–41 (1965)
22. Trybulec, A.: The Mizar logic information language. In: Studies in Logic, Grammar and Rhetoric, Białystok, Poland, vol. 1 (1980)
23. Whitehead, A.N., Russell, B.: Principia Mathematica, vol. I. Cambridge University Press, Cambridge (1910)
24. Wittgenstein, L.: Bemerkungen über die Grundlagen der Mathematik. In: Suhrkamp-Taschenbuch Wissenschaft, 3rd edn., Frankfurt, Germany, vol. 506 (1989)
25. Claus Zinn. Understanding Informal Mathematical Discourse. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany (2004)

Dimensions of Formality: A Case Study for MKM in Software Engineering

Andrea Kohlhase[1], Michael Kohlhase[2], and Christoph Lange[2]

[1] German Research Center for Artificial Intelligence (DFKI)
Andrea.Kohlhase@dfki.de
[2] Computer Science, Jacobs University Bremen
{m.kohlhase,ch.lange}@jacobs-university.de

Abstract. We study the formalization of a collection of documents created for a Software Engineering project from an MKM perspective. We analyze how document and collection markup formats can cope with an open-ended, multi-dimensional space of primary and secondary classifications and relationships. We show that RDFa-based extensions of MKM formats, employing flexible "metadata" relationships referencing specific vocabularies for distinct dimensions, are well-suited to encode this and to put it into service. This formalized knowledge can be used for enriching interactive document browsing, for enabling multi-dimensional metadata queries over documents and collections, and for exporting Linked Data to the Semantic Web and thus enabling further reuse.

1 Introduction

The field of Mathematical Knowledge Management (MKM) tries to model mathematical objects and their relationships, their creation and publication processes, and their management requirements. In [CF09, 237 ff.] Carette and Farmer analyzed "*six major lenses through which researchers view MKM*": the document, library, formal, digital, interactive, and the process lens. Quite obviously, there is a gap between the formal aspects {"library", "formal", "digital"} – related to machine use of mathematical knowledge – and the informal ones {"document", "interactive", "process"} – related to human use.

In the FormalSafe project [For08] at the German Research Center for Artificial Intelligence (DFKI) Bremen a main goal is the integration of project documents into a computer-supported software development process. MKM techniques are used to bridge the gap between informally stated user requirements and formal verification. One of the FormalSafe case studies is based on the documents of the SAMS project ("Sicherungskomponente für Autonome Mobile Systeme [Safety Component for Autonomous Mobile Systems]", see [FHL$^+$08]) at DFKI. The SAMS objective was to develop a safety component for autonomous mobile service robots and to get it certified as SIL-3 standard compliant in the course of three years. On the one hand, certification required the verification of certain safety properties in the code documents with the proof checker Isabelle [NPW02]. On the other hand, it necessitated the software development process to follow the V-Model (fig. 1). This mandates e. g. that relevant document fragments get justified and linked to corresponding fragments in a successive document refinement process (the arms of the 'V' from the upper left over the bottom to the upper right and between arms in fig. 1).

The collection of SAMS documents (we call it "**SAMSDocs**" [SAM09]) promised an interesting case study for FormalSafe as system development with respect to the V-Model regime resulted in a highly interconnected collection of design documents, certification documents, code, formal specifications, and formal proofs. Furthermore, it was supposed that adding semantics to SAMSDocs would be comparatively easy as it was developed under a strong formalization pressure.

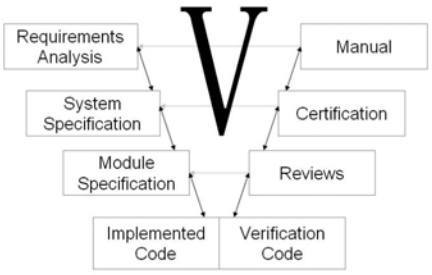


Fig. 1. Documents in the V-Model

In this paper we report on — and draw conclusions from — the SAMSDocs formalization, particularly the formalization of its LaTeX documents. In section 2, we document the process and detect inherent, distinct formality levels and the multi-dimensionality of the formalized structures. Real information needs (drawn from three use cases in the SAMS context) turn out in section 3 to be multi-dimensional. This motivates our exploration of multi-dimensional markup in section 4. Section 5 showcases the feasibility of multi-dimensional services with MKM technology enabled by multi-dimensional structured representations and section 6 concludes the paper.

2 Dimensions of Formality in SAMSDocs

In this paper, we are especially interested in the question ***"What should we sensibly formalize in a document collection and can MKM methods cope?"***. Note that we understand "to formalize" as "making implicit knowledge explicit" and not as "to make s.th. fully formal".

The SAMS project was organized as a typical Software Engineering project, its collection of documents SAMSDocs therefore has a prototypical composition of distinct document types like contract, code, or manual. Thus, SAMSDocs presents a good base for a case study with respect to our question. In fig. 2 we can see the concrete distribution over used document formats in SAMSDocs. Requirements analysis, system and module specifications, reviews, and the final manual were mainly written in LaTeX, only roughly a sixth in MS Word. The implementation in Misra-C contains Isabelle theorem prover calls.

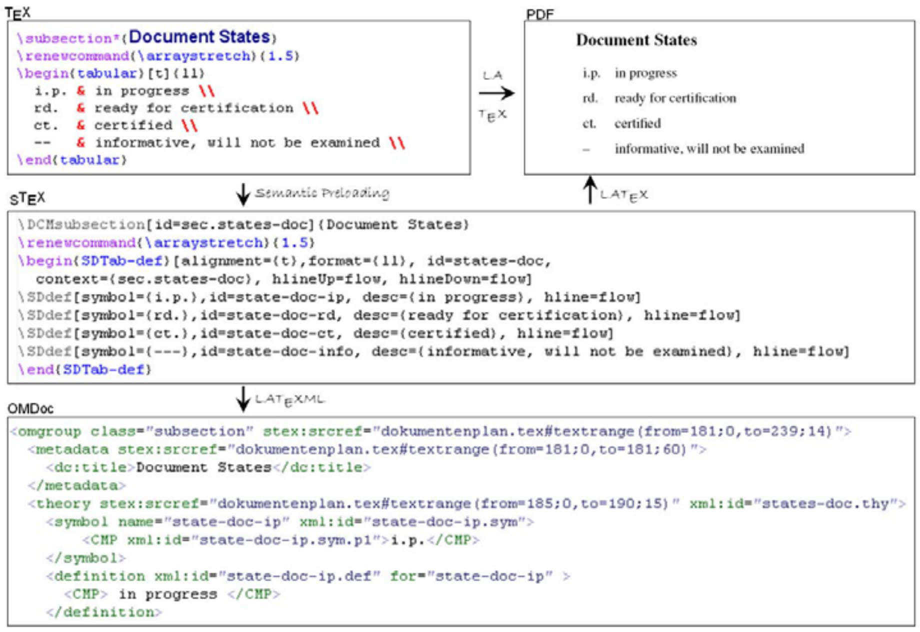
Format	**Files**	#
LaTeX	`*.tex`	251
MS Word	`*.doc`	61
Isabelle	`*.thy`	33
Misra-C Code	`*.c`	40

Fig. 2. SAMSDocs

The first stinging, but unsurprising observation was that the ***level of formality*** of the documents in SAMSDocs varies considerably — because distinct purposes create distinct formality requirements. For instance, the contract document serves as communication medium between the customer and the contractor. Here, underspecification is an important tool, whereas it is regarded harmful in the fine-granular module specifications and a fatal flaw in input logic for a theorem prover. Since this issue was already present in the set of LaTeX documents, we focused on just these.

For the formalization of this subset in SAMSDocs we used the sTeX system [Koh08], a semantic extension of LaTeX. It offers to both publish documents as high-quality human-readable PDF *and* as formal machine-processable OMDoc [Koh06] via LaTeXML [SKG+10]. Our formalization process revealed early on that previous sTeX applications (based on OMDoc 1.2) were too rigid for a stepwise semantic markup. But fortunately, sTeX also allows for the OMDoc 1.3 scheme of metadata via RDFa [ABMP08] annotations (see [Koh10]). In particular, we could 'invent' our own vocabulary for markup on demand without extending OMDoc. This new vocabulary consists of SAMSDocs-specific metadata properties and relationship types. We call the process of adding this *pre*-formal markup to SAMSDocs **(semantic) preloading**. Concretely, we extended sTeX to sTeX-SD (sTeX for SAMSDocs) by adding LaTeXML bindings for all SAMS specific TeX macros and environments used in SAMSDocs, thus enabling the conservation of the original PDF document layouts at the same time as the generation of meaningful OMDoc.

Let us look at an example for such an sTeX extension within our formalization workflow (see fig. 3). We started out with a TeX document (upper left),



```
\subsection*{Document States}
\renewcommand{\arraystretch}{1.5}
\begin{tabular}[t]{ll}
  i.p. & in progress \\
  rd.  & ready for certification \\
  ct.  & certified \\
  --   & informative, will not be examined \\
\end{tabular}
```

Document States

i.p.	in progress
rd.	ready for certification
ct.	certified
–	informative, will not be examined

```
\DCMsubsection[id=sec.states-doc]{Document States}
\renewcommand{\arraystretch}{1.5}
\begin{SDTab-def}[alignment={t},format={ll}, id=states-doc,
  context={sec.states-doc}, hlineUp=flow, hlineDown=flow]
\SDdef[symbol={i.p.},id=state-doc-ip, desc={in progress}, hline=flow]
\SDdef[symbol={rd.},id=state-doc-rd, desc={ready for certification}, hline=flow]
\SDdef[symbol={ct.},id=state-doc-ct, desc={certified}, hline=flow]
\SDdef[symbol={---},id=state-doc-info, desc={informative, will not be examined}, hline=flow]
\end{SDTab-def}
```

```
<omgroup class="subsection" stex:srcref="dokumentenplan.tex#textrange(from=181;0,to=239;14)">
  <metadata stex:srcref="dokumentenplan.tex#textrange(from=181;0,to=181;60)">
    <dc:title>Document States</dc:title>
  </metadata>
  <theory stex:srcref="dokumentenplan.tex#textrange(from=185;0,to=190;15)" xml:id="states-doc.thy">
    <symbol name="state-doc-ip" xml:id="state-doc-ip.sym">
        <CMP xml:id="state-doc-ip.sym.p1">i.p.</CMP>
    </symbol>
    <definition xml:id="state-doc-ip.def" for="state-doc-ip" >
      <CMP> in progress </CMP>
    </definition>
```

Fig. 3. The Formalization Workflow with sTeX-SD [translated by the authors]

which compiled to the PDF seen on the upper right. Here, we have a simple, two-dimensional table, which is realized with a LaTeX environment `tabular`. Semantically, this table contains a list of symbols for document states with their definitions, e. g. "i. B." for "in Bearbeitung [in progress]". As such definition tables were used throughout the project, we developed the environment `SDTab-def` and the macro `SDdef` as sTeX extensions. We determined the OMDoc output for these to be a symbol together with its definition element (for each use of `SDdef` in place of the resp. table row) and moreover, to group all of them into a theory (via using `SDTab-def`). Preloading the TeX table by employing `SDTab-def` and `SDdef` turned it into an sTeX document (middle of fig. 3) while keeping the original PDF table structure. Using LaTeXML on this sTeX document produces the OMDoc output shown in the lower area of fig. 3.

Mathematical, structural relationships have a privileged state in sTeX: their command sequence/environment syntax is analogous to the native XML element and attribute names in OMDoc. Since many objects and relationships induce formal representations for Isabelle, it seemed possible to semantically mark them up with a logic-inspired structure. But in the formalization process it soon became apparent that (important) knowledge implicit in SAMSDocs did not refer to the 'primary' structure aimed at with the use of sTeX. Instead, this knowledge was concerned with a space of less formal, 'secondary' classifications and relationships. Thus, our second observation pertains to the substance of formalizations. Even though we wanted to find out *what* we can sensibly formalize, we had assumed this to mean *how much* we can sensibly formalize. Therefore, we were rather surprised to find distinct ***formality structures*** realized in our sTeX extension. In the following we want to report on these structures.

We grouped the macros and environments of sTeX-SD in fig. 5 according to what induced them. Particularly, we distinguished the following triggers:

– "*objects*" — document fragments viewed as autonomous elements — and
– their net of relationships via the *collection*,
– *documents* and
– their *organization*al handling, and
– the *project* itself and thus, its own scheme of meaningful relationships.

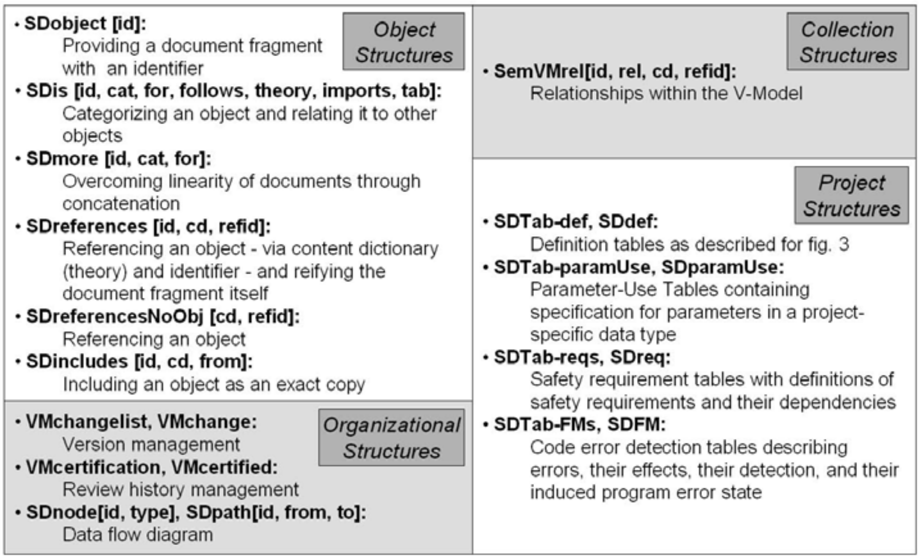
For instance, in the system specification we marked a recap of a definition of the braking distance function for straight-ahead driving s_G as an object and referenced it from within the assertion seen in fig. 4. In the module specification s_G was then meticulously specified. This document fragment is connected to the original one via a refinement-relationship from the V-Model, which determined the creation process of the collection. Documents induce layout structures like sections or subsections and they are themselves organized for example under a version management scheme. In the workflow in fig. 3 we already showcased a project-specific element, the definition table, with its meaning. Interestingly, we cannot compare formality in one group with the formality in another. For example, we cannot decide whether a document completely marked up with the

$$\boxed{\begin{gathered} q(t) = s_G(v(t)) \\ \frac{dq}{dt} = \frac{ds}{dv}\frac{dv}{dt} \end{gathered}}$$

Fig. 4. s is Braking Distance?

object-induced structures is more formal than one fully semantically enhanced by the version management markup. As these grouped structures only interact relatively lightly, we can consider them as independent **dimensions of a formality space** that is reified in the formalization process of a document collection.

Concretely, sTEX-SD covers the following dimensions and consists of the listed extension macros/environments (with attributes in [·] where sensible):



Object Structures

- **SDobject [id]:** Providing a document fragment with an identifier
- **SDis [id, cat, for, follows, theory, imports, tab]:** Categorizing an object and relating it to other objects
- **SDmore [id, cat, for]:** Overcoming linearity of documents through concatenation
- **SDreferences [id, cd, refid]:** Referencing an object - via content dictionary (theory) and identifier - and reifying the document fragment itself
- **SDreferencesNoObj [cd, refid]:** Referencing an object
- **SDincludes [id, cd, from]:** Including an object as an exact copy

Organizational Structures

- **VMchangelist, VMchange:** Version management
- **VMcertification, VMcertified:** Review history management
- **SDnode[id, type], SDpath[id, from, to]:** Data flow diagram

Collection Structures

- **SemVMrel[id, rel, cd, refid]:** Relationships within the V-Model

Project Structures

- **SDTab-def, SDdef:** Definition tables as described for fig. 3
- **SDTab-paramUse, SDparamUse:** Parameter-Use Tables containing specification for parameters in a project-specific data type
- **SDTab-reqs, SDreq:** Safety requirement tables with definitions of safety requirements and their dependencies
- **SDTab-FMs, SDFM:** Code error detection tables describing errors, their effects, their detection, and their induced program error state

Fig. 5. Formality Dimensions in sTEX-SD

Formalizing object structures is not always obvious, since many of the documents contain recaps or previews of material that is introduced in other documents/parts (e. g. to make them self-contained). Compare for example fig. 4 and fig. 6, which are actually clippings from the system specification “`KonzeptBremsmodell.pdf`”. Note the use of s resp. s_G, *both* pointing in fig. 4 to the braking distance function for straight-ahead driving (which is obvious from the local context), whereas in fig. 6 s represents the general arc length function of a circle, which is different in principle from the braking distance, but coincides here.

$$\boxed{a = s\frac{\sin\frac{\phi}{2}}{\frac{\phi}{2}} = s\,\mathrm{sinc}\,\frac{\phi}{2}}$$

Fig. 6. Yet another Braking Distance s?

We also realized that sTEX itself had already integrated another formality dimension besides the logic-inspired one, the one concerned with document layout: A typical document layout is structured into established parts like sections or modules. If we want to keep this grouping information in the formal XML document, we might use sTEX’s `DCM` package for annotating general document structures with Dublin Core (cf. [Dub08]) and similar general-purpose metadata. In the sTEX box in fig. 3 we find for example the command `DCMsubsection`

• DCM package:
Annotating general document structures with Dublin Core and similar general-purpose metadata
Document Structures

Fig. 7. The Document Formality Dimension in sTEX

with attributes containing the title of the subsection and an identifier that can be used in the usual LATEX referencing scheme.

Finally, we would like to remark that the sTEX-SD preloading process was executed as "*in-place formalization*" [SIM99]. It frequently considered several of the above dimensions for the object at hand at the same time. Therefore, the often applied metaphor of "formalization steps" does not mirror the formalization process in our case study. We found that the important aspect of the formalization was not its sequence per se, which we consider particular to the SAMSDocs collection, but the fact that the metaphor of 'steps' only worked within each single dimension of formality. In particular, there is no scale for formalization progress as distinct formality levels in distinct formality dimensions existed in a document at one point in time.

3 Multi-dimensional Information Needs

We have shown that the formalization of knowledge results in an open-ended, multi-dimensional space of primary and secondary classifications and relationships. But are multi-dimensional document formalizations beneficial for services supporting real users? Concretely, we envision potential questions in the SAMS context and services that retrieve and display answers based on the multi-dimensional markup of SAMSDocs.

Let us first take a **programmer**'s perspective. Her main information source for the programming task will stem from the module specification. But while studying it the following questions might arise:

(i) What is the definition for a certain (mathematical) symbol?[1]
(ii) How much of this specification has already been implemented?
(iii) In what state is the proof of a specific equation, has it already been formally verified so that it is safe to ground my implementation on it?
(iv) Whom can I ask for further details?

Assuming multi-dimensional markup an information retrieval system can supply useful responses. For example, it can answer (i) if technical terms in natural language are linked to the respective formal mathematical symbols they represent. For replies to (ii) and (iii) we note that, if all collection links are merged into a graph, their original placement and direction no longer makes a difference. So if we have links from the Isabelle formalization to the respective C code and links from this C code to a specification fragment, as realized in the V-Model structure of SAMSDocs, we can follow the graph from the specification through

[1] See fig. 4 and 6 for two symbols having the same appearance but different meanings.

to the state of the according proof. Drawing on the V-Model links combined with the semantic version management or the review logs, the system can deduce the answer for (iv): The code in question connects to a specification document that has authors and reviewers. This service can be as fine-grained as one is willing to formalize the granularity of the version and review management. If we admit further dimensions of markup into the picture, then the system might even find persons with similar interests (e. g. expressed in terms of the FOAF vocabulary), as has been investigated in detail for expert finder systems [SWJL10].

Now, we take a more global perspective, the one of a **project manager**. She might be concerned with the following issues:

(v) *Software Engineering Process*: How much code has been implemented to satisfy a particular requirement from the contract? Has the formal code structure passed a certain static analysis and verification? She does not want to inspect that manually by running Isabelle, thus, she needs high-level figures of, e. g., the number of mathematical statements without a formally verified proof.

(vi) *Certification*: What parts of the specification, e. g. requirements, have changed since the last certification? What other parts does that affect, and thus, what subset of the whole specification has to be re-certified?

(vii) *Human Capital*: Who is in charge of a document? How could an author be replaced if necessary, taking into account colleagues working on the same or on related documents – such as previous revisions of the same document, or its predecessor in terms of the V-Model, i. e. the document that is refined by the current one?

Exploiting the multi-dimensionality of formalized knowledge, it becomes obvious how the issues can be tackled.

Finally, we envision a **certifier**'s information needs. For inspection, she might first be interested in getting an overview, such as a list of all relevant concepts in the contract document. Then, she likes to follow the links to the detailed specification and further on to the actual implementation. For more information, she likes to contact the project investigator instead of the particular author of a code snippet. The certifier also needs to understand what parts of the whole specification are subject to a requested re-certification. Her rejection of a certain part of a document also affects all elements in the collection that depend on it. Again, a system can easily support a certifier's efficiency by combining the formalized information of distinct formality dimensions.

These use scenarios in a Software Engineering project clearly show that multi-dimensional markup is useful, since multi-dimensional queries serve natural information needs. To answer such queries, we need to represent multi-dimensional information in MKM formats.

4 Multi-dimensional Markup

Structured representations are usually realized as files marked up in formats that reflect the primary formalization intent and markup preferences of the formalizer.

In the evaluation of document formats it is thus important to realize that every representation language concentrates on only a subset of possible relationships, which it treats with specific language constructs. Note that therefore the formality space of a semantically enhanced document is very often reduced to this primary dimension. On the formal side, for example, a plethora of system-specific logics exist. Furthermore, formal systems increasingly contain custom modularization infrastructures, ranging from simple facilities for inputting external files to elaborate multi-logic theory graphs [MML07]. Collections of informal documents, on the other side, are often structured by application-specific metadata like the Math Subject Classification [Soc09] or the V-Model relations as in our case study.

No given format can natively capture *all* aspects of the domain via special-purpose markup primitives. It has to relegate some of them to other mechanisms like the sTeX-SD extension for the formalization of SAMSDocs, if more dimensions of the formality space than the primary one are to be covered. In representation formats that support fragment identifiers — e. g. XML-based ones — these relationships can be expressed as stand-off markup in RDF (Resource Description Framework [RDF04]), i. e., as subject-predicate-object triples, where subject and object are URI references to a fragment and the predicate is a reference to a relationship specified in an external vocabulary or ontology[2]. As we have XML-based formats for informal documents (e. g. XHTML+MathML+SVG) and formal specifications (OpenMath or Content MathML), we can in principle already encode multi-dimensional structured representations, if we only supply according metadata vocabularies for their structural relationships. Indeed this is the basic architecture of the "Semantic Web approach" to eScience, and much of the work of MKM can be seen as attempts to come up with good metadata vocabularies for the mathematical/scientific domain.

Since RDF stand-off markup is notoriously difficult to keep up to date, **RDFa** [ABMP08] has been developed: A set of attributes for embedding RDF annotations into XML-based languages, originally XHTML. On the one hand, RDFa serves as an enabling technology for making XML-based languages extensible by inter- and intra-document relationships. On the other hand, RDFa serves as a vehicle for document format interoperability. All relationships from a format F that cannot be natively represented in a format F' can be represented as RDFa triples, where the predicate is from an appropriately designed metadata vocabulary that describes the format F. For instance, an OMDoc `<theory>` element can be represented as `<div typeof="http://omdoc.org/ontology#Theory">` in XHTML, using the OMDoc ontology [Lan10]. Support of RDFa relationships make all XML-based formats theoretically equivalent, if they allow fine-grained

[2] The difference between "vocabulary" and "ontology" is not sharply defined. Vocabularies are often developed in a bottom-up community effort and tend to have a low degree of formality, whereas ontologies are often designed by a central group of experts and have a higher degree of formality. Here, we use "vocabulary" in its general sense of a set of terms from a particular domain of interest. This subsumes the term "ontology", which we will reserve for cases that require a more formal domain model.

text structuring with elements like XHTML's `div` or `span` everywhere (so that arbitrary text fragments can be turned into objects). In particular, they become **formats for multi-dimensional markup** as respective other dimensions can always be added via RDFa. We have detailed the necessary extensions for the OMDoc format in [Koh10], so that analogous extensions for any of the XML-based formats used in the MKM community should be rather simple to create.

Note that the pragmatic restriction to XML-based representation formats is not a loss of generality. In the MKM sphere the three classes of non-XML languages used are computational logics, TEX/LATEX, and PostScript/PDF. We see computational logics as compact front-end formats that are optimized for manual input of formal structured representations; it is our experience that these can be transformed into the XML-based OpenMath, MathML, or OMDoc without loss of information (but with a severe loss of notational conciseness). We consider TEX/LATEX as analogous for informal structured representations; they can be transformed to XHTML+MathML by the LATEXML system. The last category of formats are presentation/print-oriented output and archival formats where the situation is more problematic: PostScript (PS) is largely superseded by PDF which allows standard document-level RDF annotations via XMP and the finer-granular annotations we need for structured representations via extensions as in [GMH+07] or [Eri07]. But PS/PDF are usually generated from other formats (mostly office formats or LATEX), so that alternative generation into XML-based formats like XHTML or OMDoc can be used.

Note as well that a dimension typically corresponds to a vocabulary. In the course of the SAMSDocs case study, most vocabularies have initially been implemented from scratch in a project-specific ad hoc way. But they can be elaborated towards ontologies via sTEX and these can be translated to RDF-based formats that automated reasoners understand [KKL10]. An alternative is reusing existing ontologies. This has the advantage that they are more widely used and thus, reusable services may already have been implemented for them. For instance, there already exists a vocabulary that defines basic properties of persons and organizations: FOAF (Friend of a Friend [BM07]). The widely known Dublin Core element set is also available as an ontology [Dub08]. DCMI Terms [DCM08], a modernized and extended version of the Dublin Core element set, offers a basic vocabulary for revision histories – but not for reviewing and certification. DOAP (Description of a Project [Dub10]) describes software projects, albeit focusing on the top-level structure of public open source projects. Lin et al. have developed an ontology for the requirements-related parts of the V-Model (cf. [LFB96]). Happel and Seedorf briefly review further ontologies about Software Engineering [HS06]. As, e. g. the SAMSDocs vocabularies can be integrated with existing ontologies by declaring appropriate subclass or equivalence relationships, services can make use of the best of both worlds.

5 Multi-dimensional Services with MKM Technology

We will now study an avenue towards a concrete implementation of services based on the use cases described in sect. 3 to show how MKM technologies can cope with multi-dimensional information needs demonstrating their feasibility. Concretely, we will study the task of project manager Nora to find a substitute for employee Alice. All required information is contained in the sTeX-SD documents. To abstract from the particulars of sTeX/OMDoc RDFa encoding — e. g. the somewhat arbitrarily chosen direction of the relations or the interaction of metadata relations with the document and the special markup for the mathematical dimension — we extract a uniform RDF representation of the embedded structures, which can then be queried in the SPARQL language [PS08]. Listing 1 shows the necessary query in all detail.

Listing 1. Finding a Substitute for an Employee via the V-Model

```
# declaration of vocabulary (= dimension) namespace URIs
PREFIX vm:    <http://www.sams-projekt.de/ontologies/VersionManagement#>
PREFIX omdoc: <http://omdoc.org/ontology#>                              # OMDoc
PREFIX semVM: <http://www.sams-projekt.de/ontologies/V-model#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>                   # Dublin Core
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>         # XML Schema datatypes

SELECT ?potentialSubstituteName WHERE {
  # for each document Alice is responsible for, get all of its parts
  # i.e. any kind of semantic (sub)object in the document
  ?document vm:responsible <.../employees#Alice> ;
            omdoc:hasPart  ?object .

  # find other objects that are related to each ?object
  # 1. in that ?object refines them via the V-model
  { ?object semVM:refines ?relatedObject }
  UNION
  # 2. or in that they are other mathematical symbols defined in terms
  #      of ?object (only applies if ?object itself is a symbol)
  { ?object omdoc:occursInDefinitionOf ?relatedObject }

  # find the document that contains the related object and the person
  # responsible for that document ...
  ?otherDocument omdoc:hasPart  ?relatedObject ;
                 dc:date        ?date ;
                 vm:responsible ?potentialSubstitute .
  # (only considering documents that are sufficiently up to date)
  FILTER (?otherDocument > "2009-01-01"^^xsd:date)

  # ... and the real name of that person
  ?potentialSubstitute foaf:name ?potentialSubstituteName .
}
```

In this query we assume that Alice's FOAF profile is a part of our collection, having the URI `.../employees#Alice`. Nora retrieves all documents in the collection for which Alice is known to be the responsible person. For any object O in each of these documents (e. g. the detailed specification of the braking distance function for straight-ahead driving s_G from fig. 4), she selects those objects that are refined by O in terms of the V-Model (e. g. the general braking distance s). Additionally, she considers the mathematical dimension and selects all objects that are related to O by mathematical definition, e. g. the braking function that uses s_G. Of any such related object, Nora finds out to what document it belongs. She is only interested in recent documents and therefore filters them by date. Finally, she determines the responsible persons via the version management links, and gets their names from their FOAF descriptions. The assumption behind this query is that, if, for example, Pierre is responsible for the specification that introduces the general braking distance s, which Alice has refined, Pierre can be considered as a substitute for Alice. Note that getting the answer draws on the collection structures of SAMSDocs (V-Model), on the mathematical structures, as well as on the organizational structures (version management). It is easy to imagine how additional formality dimensions can be employed for increasing precision or recall of the query, or for ranking results. Consider, for example, another filter that only accepts as substitutes employees who have never got a document rejected in any previous certification.

The complexity of the query in listing 1 is directly caused by the complexity of the underlying multi-dimensional structures and the non-triviality of answering high-level project management queries from the detailed information in SAMSDocs. As users like Nora would not want to deal with a machine-oriented query language, we have developed a system that integrates versioned storage of semantic document collections with human-oriented presentation with embedded interactive services [DKL+10]. Thus, the rendered documents serve as command centers for executing queries and displaying results[3]. They provide access to queries in two ways: Queries with a fixed structure that have to be answered recurringly will be made available right in the (rendered) documents in appropriate places. This is the case with our employeee substitution query: This month, Alice may be ill, whereas next month, Bob may be on holiday. Access to this query can be given wherever an employee or a reference to an employee occurs in a document. Alternatively, non-prefabricated queries can be composed more intuitively on demand using a visual input form.

These examples show that multi-dimensional queries like the ones naturally coming up in Software Engineering scenarios (sect. 3) can be answered with existing MKM technology. Moreover, it illustrates that multi-dimensional markup affords multi-dimensional services. If we interpret our dimensions as distinct contexts, our services become context-sensitive, as dimensions can be filtered in and out. For instance, the context menu of certification documents could be equipped with menu entries for committing an approval or rejection to the server, which would only be displayed to the certifier. The server could then trigger further actions, such as marking the document that contains a rejected object and all dependencies of that object as rejected, too. In general, the more dimensions are formalized in a document, the more context-sensitive services become available.

6 Conclusion and Further Work

In this paper we have studied the applicability of MKM technologies in Software Engineering beyond “Formal Methods” (based on the concrete SAMSDocs document collection and its formalization). The initial hypothesis here is that

[3] In particular, the rendered XHTML+MathML also preserves the original semantic structure as parallel MathML markup and RDFa annotations, so that a suitable browser plugin can dynamically generate interaction points for semantic services; see [KKL10] for details.

contract documents, design specifications, user manuals, and integration reports can be partially formalized and integrated into a computer-supported software development process. To test this hypothesis we have studied a collection of documents created for the development of a safety zone computation, the formal verification that the braking trajectory always lies in the safety zone, and the SIL3 certification of this fact by a public certification agency. As the project documents contain a wealth of (informal) mathematical content, MKM formats (in this case our OMDoc format) are well-suited for this task. During the formalization of the LaTeX part of the collection, we realized that the documents contain an open-ended, multi-dimensional space of formality that can be used for supporting projects — if made explicit.

We have shown that RDFa-based extensions of MKM formats, employing flexible "metadata" relationships referencing specific vocabularies, can be used to encode this formality space and put it into service. We have pointed out that the "dimensions" of this space can be seen to correspond to different metadata vocabularies. Note that the distinction between data and metadata blurs here as, for example, the OMDoc data model realized by native markup in the OMDoc format can also be seen as OMDoc metadata and could equally be realized by RDFa annotations to some text markup format, where the meaning of the annotations is given by the OMDoc ontology. This "metadata view" is applicable to all MKM formats that mark up informal mathematical texts (e. g. MathDox [CCB06] and MathLang [KWZ08]) as long as they formalize their data model in an ontology. This observation makes decisions about which parts of the formality space to support with native markup a purely pragmatic choice and opens up new possibilities in the design of representation formats. It seems plausible that all MKM formats use native markup for mathematical knowledge structures (we think of them as primary formality structures for MKM) and differ mostly in the secondary ones they internalize. XHTML+MathML+RDFa might even serve as a baseline interchange format for MKM applications[4], since it is minimally committed. Note that if the metadata ontologies are represented in modular formats that admit theory morphisms, then these can be used as crosswalks between secondary metadata for higher levels of interoperability. We leave its development to future work.

The formalized secondary formality structures can be used for enriching interactive document browsing and for enabling multi-dimensional metadata queries over documents and collections. We have shown a set of exemplary multi-dimensional services based on the RDFa-encoded metadata, mostly centered around Linked Data approaches based on RDF-based queries. More services can be obtained by exporting Linked Data to the Semantic Web or a company intranet and thus enabling further reuse. In particular, the multi-dimensionality observed in this paper and its realization via flexible metadata regimes in representation formats allows the knowledge engineers to tailor the level of formality to the intended applications.

[4] Indeed, a similar proposal has been made for Semantic Wikis [VO06], which have related concerns but do not primarily involve mathematics.

In our case study, the metadata vocabularies ranged from project-specific ones that had to be developed (e. g. definition tables) to general ones like the V-Model vocabulary, for which external ontologies could be reused later on. We expect that such a range is generally the case for Software Engineering projects, and that the project-specific vocabularies may stabilize and be standardized in communities and companies, lowering the formalization effort entailed by each individual project. In fact we anticipate that such metadata vocabularies and the software development support services will become part of the strategic knowledge of technical organizations.

In [CF09, 241] Carette and Farmer challenge MKM researchers by assessing some of their technologies: "*A lack of requirements analysis very often leads to interesting solutions to problems which did not need solving*". With this paper we hope to have shown that MKM technologies can be extended to cope with "real world concerns" (in Software Engineering). Indeed, industry is becoming more and more aware of and interested in Linked Data (see e. g. [Ser08] and [LDF, Question 14]), which boosts relevance to the multi-dimensional knowledge management methods presented in this paper.

References

[ABMP08] Adida, B., Birbeck, M., McCarron, S., Pemberton, S.: RDFa in XHTML: Syntax and processing. W3C Recommendation, World Wide Web Consortium (W3C) (October 2008)

[BM07] Brickley, D., Miller, L.: FOAF vocabulary specification 0.91. Technical report, ILRT Bristol (November 2007)

[CCB06] Cohen, A.M., Cuypers, H., Reinaldo Barreiro, E.: Mathdox: Mathematical documents on the web. In: OMDoc – An Open Markup Format for Mathematical Documents (Version 1.2) (Koh 06), ch. 26.7, pp. 278–282

[CDSCW09] Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.): Calculemus 2009. LNCS (LNAI), vol. 5625. Springer, Heidelberg (2009)

[CF09] Carette, J., Farmer, W.: A review of mathematical knowledge management. In: Carette, et al. (eds.) [CDSCW09], pp. 233–246.

[DCM08] DCMI Usage Board. DCMI metadata terms. DCMI recommendation, Dublin Core Metadata Initiative (2008)

[DKL$^+$10] David, C., Kohlhase, M., Lange, C., Rabe, F., Zhiltsov, N., Zholudev, V.: Publishing math lecture notes as linked data. In: Aroyo, L., Antoniou, G., Hyvönen, E. (eds.) ESWC 2010, Part II. LNCS, vol. 6089, pp. 370–375. Springer, Heidelberg (2010)

[Dub08] Dublin Core metadata element set. DCMI recommendation, Dublin Core Metadata Initiative (2008)

[Dub10] Dubmill, E.: DOAP – description of a project (March 2010), http://trac.usefulinc.com/doap

[Eri07] Eriksson, H.: The semantic-document approach to combining documents and ontologies. International Journal of Human-Computer Studies 65(7), 624–639 (2007)

[FHL$^+$08] Frese, U., Hausmann, D., Lüth, C., Täubig, H., Walter, D.: The importance of being formal. In: Hungar, H. (ed.) International Workshop on the Certification of Safety-Critical Software Controlled Systems SafeCert 2008. Electronic Notes in Theoretical Computer Science, vol. 238, pp. 57–70 (September 2008)

[For08] FormalSafe (2008), http://www.dfki.de/sks/formalsafe/ (March 2010)

[GMH$^+$07] Groza, T., Möller, K., Handschuh, S., Trif, D., Decker, S.: SALT: Weaving the claim web. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 197–210. Springer, Heidelberg (2007)

[HS06] Happel, H.-J., Seedorf, S.: Applications of ontologies in software engineering. In: Proc. 2nd International Workshop on Semantic Web Enabled Software Engineering, SWESE 2006 (2006)

[KKL10] Kohlhase, A., Kohlhase, M., Lange, C.: sTeX – a system for flexible formalization of linked data. submitted to I-SEMANTICS 2010 (2010)

[Koh06] Kohlhase, M.: OMDoc – An Open Markup Format for Mathematical Documents [version 1.2]. LNCS (LNAI), vol. 4180. Springer, Heidelberg (2006)

[Koh08] Kohlhase, M.: Using LaTeXas a semantic markup format. Mathematics in Computer Science 2(2), 279–304 (2008)

[Koh10] Kohlhase, M.: An open markup format for mathematical documents OMDoc (version 1.3). Draft Specification (2010)

[KWZ08] Kamareddine, F., Wells, J.B., Zengler, C.: Computerising mathematical text with mathlang. Electron. Notes Theor. Comput. Sci. 205, 5–30 (2008)

[Lan10] Lange, C.: The OMDoc document ontology (2010), http://kwarc.info/projects/docOnto/omdoc.html (March 2010)

[LDF] Linked data FAQ, http://structureddynamics.com/linked_data.html

[LFB96] Lin, J., Fox, M.S., Bilgic, T.: A requirement ontology for engineering design. In: Proceedings of 3rd International Conference on Concurrent Engineering, pp. 343–351. Technomic Publishing Company, Inc. (August 1996)

[MML07] Mossakowski, T., Maeder, C., Lüttich, K.: The heterogeneous tool set. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 519–522. Springer, Heidelberg (2007)

[NPW02] Nipkow, T., Paulson, L.C., Wenzel, M.T.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)

[PS08] Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Recommendation, World Wide Web Consortium (W3C) (January 2008)

[RDF04] Resource description framework (RDF) (2004), http://www.w3.org/RDF/

[SAM09] SAMS. SAMSDocs: The document collection of the SAMS project (2009), http://www.sams-projekt.de

[Ser08] Servant, F.-P.: Linking enterprise data. In: Bizer, C., Heath, T., Idehen, K., Berners-Lee, T. (eds.) Linked Data on the Web (LDOW 2008). CEUR Workshop Proceedings, vol. 369 (April 2008)

[SIM99] Shipman III, F.M., McCall, R.J.: Incremental formalization with the hyper-object substrate. ACM Trans. Inf. Syst. 17(2), 199–227 (1999)

[SKG$^+$10] Stamerjohanns, H., Kohlhase, M., Ginev, D., David, C., Miller, B.: Transforming large collections of scientific publications to XML. Mathematics in Computer Science (in Press 2010)

[Soc09] American Mathematical Society. Mathematics Subject Classification MSC2010 (2009), http://www.ams.org/mathscinet/msc/

[SWJL10] Stankovic, M., Wagner, C., Jovanovic, J., Laublet, P.: Looking for experts? what can linked data do for you? In: Bizer, C., Heath, T., Berners-Lee, T., Hausenblas, M. (eds.) Linked Data on the Web (LDOW 2010). CEUR Workshop Proceedings (April 2010)

[VO06] Völkel, M., Oren, E.: Towards a Wiki Interchange Format (WIF). In: Völkel, M., Schaffert, S., Decker, S. (eds.) Proceedings of the 1st Workshop on Semantic Wikis, European Semantic Web Conference 2006, CEUR Workshop Proceedings, Budva, Montenegro, vol. 206 (June 2006)

Towards MKM in the Large: Modular Representation and Scalable Software Architecture

Michael Kohlhase, Florian Rabe, and Vyacheslav Zholudev

Computer Science, Jacobs University Bremen
{m.kohlhase,f.rabe,v.zholudev}@jacobs-university.de

Abstract. MKM has been defined as the quest for technologies to manage mathematical knowledge. MKM "in the small" is well-studied, so the real problem is to scale up to large, highly interconnected corpora: "MKM in the large". We contend that advances in two areas are needed to reach this goal. We need representation languages that support incremental processing of all primitive MKM operations, and we need software architectures and implementations that implement these operations scalably on large knowledge bases.

We present instances of both in this paper: the MMT framework for modular theory-graphs that integrates meta-logical foundations, which forms the base of the next OMDOC version; and TNTBase, a versioned storage system for XML-based document formats. TNTBase becomes an MMT database by instantiating it with special MKM operations for MMT.

1 Introduction

[12] defines the objective of MKM to be *to develop new and better ways of managing mathematical knowledge using sophisticated software tools* and later states the "Grand Challenge of MKM" as *a universal digital mathematics library (UDML)*, which is indeed a grand challenge, as it envisions that the UDML *would continuously grow and in time would contain essentially all mathematical knowledge*, which is estimated to be well in excess of 10^7 published pages.[1] All current efforts towards comprehensive machine-organizable libraries of mathematics are at least three orders of magnitude smaller than the UDML envisioned by Farmer in 2004: Formal libraries like those of Mizar ([33], Isabelle ([26]) or PVS ([25]) have ca. $10^{4.x}$ statements (definitions and theorems). Even the semi-formal, commercial Wolfram MathWorld which hails itself *the world's most extensive mathematics resource* only has $10^{4.1}$ entries. There is anecdotal evidence that already at this size, management procedures are struggling.

To meet the MKM Grand Challenge will have to develop fundamentally more scalable ways of dealing with mathematical knowledge, especially since [12] goes on to postulate that the UDML *would also be continuously reorganized and consolidated as new connections and discoveries were made*. Clearly this can only be achieved algorithmically; experience with the libraries cited above already show that manual MKM does not scale sufficiently. Most of the work in the MKM community has concentrated

[1] For instance, Zentralblatt Math contains 2.4 million abstracts of articles form mathematical journals in the last 100 years.

on what we could call "MKM in the small", i.e. dealing with aspects of MKM that do not explicitly address issues of very large knowledge collections; these we call "MKM in the large".

In this paper we contribute to the MKM Grand Challenge of doing formal "MKM in the large" by analyzing scalability challenges inherent in MKM and propose steps towards solutions based on our MMT format, which is the basis for the next version of OMDoc. We justify our conclusions and recommendations for scalability techniques with concrete case studies we have undertaken in the last years. Section 2 tackles scalability issues pertaining to the representation languages used in the formalization of mathematical knowledge. Section 3 discusses how the modularity features of MMT can be realized scalably by realizing basic MKM functionality like validation, querying, and presentation incrementally and carefully evaluating the on-the-fly computation (and caching) of induced representations. These considerations, which are mainly concerned with efficient computation "in memory" are complemented with a discussion of mass storage, caching, and indexing in Section 4, which addresses scalability issues in large collections of mathematical knowledge. Section 5 concludes the paper and addresses avenues of further research.

2 A Scalable Representation Language

Our representation language MMT was introduced in [29]. It arises from three central design goals. Firstly, it should provide an expressive but simple **module system** as modularity is a necessary requirement for scalability. As usual in language design, the goals of simplicity and expressivity form a difficult trade-off that must be solved by identifying the right primitive module constructs. Secondly, scalability across semantic domains requires **foundation-independence** in the sense that MMT does not commit to any particular foundation (such as Zermelo-Fraenkel set theory or Church's higher-order logic). Providing a good trade-off between this level of generality and the ability to give a rigorous semantics is a unique feature of MMT. Finally, scalability across implementation domains requires **standards-compliance**, and while using XML and OPENMATH is essentially orthogonal to the language design, the use of URIs as identifiers is not as it imposes subtle constraints that can be very hard to meet a posteriori.

MMT represents logical knowledge on three levels: On the **module level**, MMT builds on modular algebraic specification languages for logical knowledge such as OBJ [14], ASL [32], development graphs [1], and CASL [7]. In particular, MMT uses theories and theory morphism as the primitive modular concepts. Contrary to them, MMT only imposes very lightweight assumptions on the underlying language. This leads to a very simple generic module system that subsumes almost all aspects of the syntax and semantics of specific module systems such as PVS [25], Isabelle [26], or Coq [3].

On the **symbol level**, MMT is a simple declarative language that uses named symbol declarations where symbols may or may not have a type or a definiens. By experimental evidence, this subsumes virtually all declarative languages. In particular, MMT uses the Curry-Howard correspondence [8,17] to represent axioms and theorem as constants, and proofs as terms. Sets of symbol declarations yield theories and correspond to OPENMATH content dictionaries.

On the **object level**, MMT uses the formal grammar of OPENMATH [6] to represent mathematical objects without committing to a specific formal foundation. The semantics of objects is given proof theoretically using judgments for typing and equality between objects. MMT is parametric in these judgments, and the precise choice is relegated to a **foundation**.

2.1 Module System

Sophisticated mathematical reasoning usually involves several related but different mathematical contexts, and it is desirable to exploit these relationships by moving theorems between contexts. It is well-known that modular design can reduce space to an extent that is exponential in the depth of the reuse relation between the modules, and this applies in particular to the large theory hierarchies employed in mathematics and computer science.

The first applications of this technique in mathematics are found in the works by Bourbaki ([4,5]), which tried to prove every theorem in the context with the smallest possible set of axioms. MMT follows the "little theories approach" proposed in [11], in which separate contexts are represented by separate **theories**, and structural relationships between contexts are represented as **theory morphisms**, which serve as conduits for passing information (e.g., definitions and theorems) between theories (see [10]). This yields **theory graphs** where the nodes are theories and the paths are theory morphisms.

Example 1 (Algebra). For example, consider the theory graph in Fig. 1 for a portion of algebra, which was formalized in MMT in [9]. It defines the theory of magmas (A magma has a binary operation without axioms.) and extends it successively to monoids, groups, and commutative groups. Then the theory of rings is formed by importing from both `CGroup` (for the additive operation) and `Monoid` (for the multiplicative operation).

A crucial property here is that the imports are named, e.g., `Monoid` imports from `Magma` via an import named `mag`. While redundant in some cases, it is essential in `Ring` where we have to distinguish two theory morphisms from `Monoid` to `Ring`: The composition `add/grp/mon` for the additive monoid and `mult` for the multiplicative monoid.

The import names are used to form qualified names for the imported symbols. For example, if $*$ is the name of the binary operation in `Magma`, then $\mathtt{add/grp/mon/mag/}*$ denotes addition and $\mathtt{mult/mag/}*$ multiplication in `Ring`. Of course, MMT supports the use of abbreviations instead of qualified names, but it is a crucial prerequisite for scalability to make qualified names the default: Without named imports, every time we add a new name in `Magma` (e.g, for an abbreviation or a theorem), we would have to add corresponding renamings in `Ring` to avoid name clashes.

Another reason to use named imports is that we can use them to instantiate imports with theory morphisms. In our example, distributivity is stated separately as a theory that imports two magmas. Let us assume, the left distributivity axiom is stated as

$$\forall x, y, z. x\ \mathtt{mag1/}*\ (y\ \mathtt{mag2/}*\ z) = (x\ \mathtt{mag1/}*\ y)\ \mathtt{mag2/}*\ (x\ \mathtt{mag1/}*\ z)$$

Then the import *dist* from `Distrib` to `Ring` will carry the instantiations $\mathtt{mag1} \mapsto \mathtt{mult/mag}$ and $\mathtt{mag2} \mapsto \mathtt{add/grp/mon/mag}$.

In other module systems such as SML, such instantiations are called (asymmetric) sharing declarations. In terms of theory morphism, their semantics is a commutative diagram, i.e., an equality between two morphisms such as $\texttt{dist/mag1} = \texttt{mult/mag} : \texttt{Magma} \to \texttt{Ring}$. This provides MMT users and systems with a module level invariant for the efficient structuring of large theory graphs.

Besides imports, which induce theory morphisms into the containing theory, there is a second kind of edge in the theory graph: Views are explicit theory morphisms that represent translations between two theories. For example, the node on the right side of the graph represents a theory for the integers, say declaring the constants $0, +, -, 1$, and $\cdot$. The fact that the integers are a commutative group is represented by the view `v1`: If we assume that `Monoid` declares a constant e for the unit element and `Group` a constant `inv` for the inverse element, then `v1` carries the instantiations $\texttt{grp/mon/mag/}* \mapsto +$, $\texttt{grp/mon/e} \mapsto 1$, and $\texttt{grp/inv} \mapsto -$. Furthermore, every axiom declared or imported in `CGroup` is mapped to a proof of the corresponding property of the integers.

The view `v2` extends `v1` with corresponding instantiations for multiplication. MMT permits modular views as well: When defining `v2`, we can import all instantiations of `v1` using $\texttt{add} \mapsto \texttt{v1}$. As above, the semantics of such an instantiation is a commutative diagram, namely $\texttt{v2} \circ \texttt{add} = \texttt{v1}$ as intended.

The major advantage of modular design is that every declaration — abbreviations, theorems, notations etc. — effects **induced declarations** in the importing theories. A disadvantage is that declarations may not always be located easily, e.g., the addition in a ring is declared in a theory that is four imports away. MMT finds a compromise here: Through qualified names, all induced declarations are addressable and locatable. The process of removing the modularity by adding all these induced declarations to all theories is called **flattening**.

$$\begin{tikzcd}
\texttt{Magma} \arrow[r, "\texttt{mag}"] \arrow[d, bend right, "\texttt{mag1}"'] \arrow[d, bend left, "\texttt{mag2}"] & \texttt{Monoid} \arrow[r, "\texttt{mon}"] \arrow[dr, "\texttt{mult}"'] & \texttt{Group} \arrow[r, "\texttt{grp}"] & \texttt{CGroup} \arrow[dl, "\texttt{add}"'] \arrow[d, "\texttt{v1}"] \\
\texttt{Distrib} \arrow[rr, "\texttt{dist}"'] & & \texttt{Ring} \arrow[r, "\texttt{v2}"'] & \texttt{Integers}
\end{tikzcd}$$

Fig. 1. Algebraic Hierarchy

Case Study 1: The formalization in [9] uses the Twelf module system ([31]), which is a special case of MMT. Twelf automatically computes the flattened theory graph. The modular theory graph including all axioms and proofs can be written in 180 lines of Twelf code. The flattened graph is computed in less than half a second and requires more than 1800 lines.

The same case study defines two theories for lattices, one based on orderings and one based on algebra, and gives mutually inverse views to prove the equivalence of the two theories. Both definitions are modular: Algebraic lattices arise by importing twice from the theory of semi-lattices; order-based lattices arise by importing the infimum operation twice, once for the ordering and once for its dual. Consequently, the views can be given modularly as well, which is particularly important because views must map axioms to expensive-to-find proofs. These additional declarations take 310 lines of Twelf in modular and 3500 lines in flattened form.

These numbers already show the value of modularity in representation already in very small formalizations. If this is lacking, later steps will face severe scalability problems from blow-up in representation. Here, the named imports of MMT were the crucial innovation to strengthen modularity.

2.2 Foundation-Independence

Mathematical knowledge is described using very different foundations, and the most common foundations can be grouped into set theory and type theory. Within each group there are numerous variants, e.g., Zermelo-Fraenkel or Gödel-Bernays set theory, or set theories with or without the axiom of choice. Therefore, a single representation language can only be adequate if it is foundation-independent.

OPENMATH and OMDOC achieve this by concentrating on structural issues and leaving lexical ones to an external definition mechanism like content dictionaries or theories. In particular, this allows us to operate without choosing a particular foundational logical system, as we can just supply content dictionaries for the symbols in the particular logic. Thus, logics and in the same way foundations become theories, and we speak of the **logics-as-theories** approach.

But conceptually, it is helpful to distinguish levels here. To state a property in the theory `CGroup` like commutativity of the operation $\circ := \mathtt{grp/mon/mag/*}$ as $\forall a, b.a \circ b = b \circ a$, we use symbols $\forall$ and $=$ from first-order logic together with $\circ$ from `CGroup`. Even though it is structurally possible to build algebraic theories by simply importing first-order logic, this would fail to describe the meta-relationship between the theories. But this relation is crucial, e.g., when interpreting `CGroup` in the integers, the symbols of the meta-language are not interpreted because a fixed interpretation is given in the context.

To understand this example better, we use the M/T notation for meta-languages. M/T refers to working in the object language T, which is defined within the meta-language M. For example, most of mathematics is carried out in FOL/ZF, i.e., first-order logic is the meta-language, in which set theory is defined. FOL itself might be defined in a logical framework such as LF, and within ZF, we can define the language of natural numbers, which yields $LF/FOL/ZF/Nat$. For algebra, we obtain, e.g., $FOL/\mathtt{Magma}$. MMT makes this meta-relation explicit: Every theory T may point to another theory M as its meta-theory. We can write this as $MMT/(M/T)$.

In Fig. 2, the algebra example is extended by adding meta-theories. The theory `FOL` for first-order logic is the meta-theory for all algebraic theories, and the theory `LF` for the logical framework LF is the meta-theory of `FOL` and of the theory `HOL` for higher-order logic.

$$\begin{tikzcd}
 & & \mathtt{LF} \arrow[r, "m"] \arrow[dl, dash, "\text{meta}"'] \arrow[dr, dash, "\text{meta}"] & \mathtt{Isabelle} \\
 & \mathtt{FOL} \arrow[rr, "m'"] \arrow[dl, dash, "\text{meta}"'] \arrow[dr, dash, "\text{meta}"] & & \mathtt{HOL} \\
\mathtt{Monoid} \arrow[rr, "\mathtt{mult}"] & & \mathtt{Ring} &
\end{tikzcd}$$

Fig. 2. Meta-Theories

Now the crucial advantage of the logics-as-theories approach is that on all three levels the same module system can be used: For example, the views m and m' indicate possible translations on the levels of logical frameworks and logics, respectively. Similarly, logics and foundations can be built modularly. Thus, we can use imports to represent inheritance at the level of logical foundations and views to represent formal translations between them. Just like in the little theories approach, we can prove meta-logical results in the simplest foundation that is expressive enough and then use views to move results between foundations.

Example 2 (Little Logics and Little Foundations). In [15], we formalize the syntax, proof theory, and model theory and prove the soundness of first-order logic in MMT. Using the module system, we can treat all connectives and quantifiers separately. Thus, we can reuse these fragments to define other logics, and in [18] we do that, e.g., for sorted first-order logic and modal logic.

For the definition of the model theory, we need to formalize set theory in MMT, which is a significant investment, and even then doing proofs in set theory — as needed for the soundness proof — is tedious. Therefore, in [16], we develop the set theoretical foundation itself modularly. We define a typed higher-order logic `HOL` first, which is expressive enough for many applications such as the above soundness proof. Then a view from `HOL` to `ZF` proves that `ZF` is a refinement of `HOL` and completes the proof of the soundness of FOL relative to models defined in `ZF`.

Case Study 2: Ex. 2 already showed that it is feasible to represent foundations and relations between foundations in MMT. Being able to this is a qualitative aspect of cross-domain scalability. In another case study, we represented $LF/Isabelle$ and $LF/Isabelle/HOL$ ([26,23]) as well as a translation from them into $LF/FOL/ZFC$ (see [18]). To our knowledge, MMT is the only declarative formalism in which comparable foundation or logic translations have been conducted. In Hets ([21]) a number of logic translations are implemented in Haskell. Twelf and Delphin provide logic and functional programming languages, respectively, on top of LF ([27,28]), which have been used to formalize the HOL-Nuprl translation ([22]).

2.3 Symbol Identifiers "in the Large"

In mathematical languages, we need to be able to refer to (i.e., identify) content objects in order to state the semantic relations. It was a somewhat surprising realization in the design of MMT that to understand the symbol identifiers is almost as difficult as to understand the whole module system. Theories are containers for symbol declarations, and relations between theories define the available symbols in any given theory. Since every available symbol should have a canonical identifier, the syntax of identifiers is inherently connected to the possible relations between theories.

In principle, there are two ways to identify content object: **by location** (relative to a particular document or file) and **by context** (relative to a mathematical theory). The first one essentially makes use of the organizational structure of files and file systems, and the second makes use of mathematical structuring principles supplied by the representation format.

As a general rule, it is preferable to use identification by context as the distribution of knowledge over file systems is usually a secondary consideration. Then the mapping between theory identifiers and physical theory locations can be deferred to an extralinguistic catalog. Resource identification by context should still be compatible with the URI-based approach that mediates most resource transport over the internet. This is common practice in scalable programming languages such as Java where package identifiers are URIs and classes are located using the `classpath`.

For logical and mathematical knowledge, the OPENMATH 2 standard ([6]) and the current OMDOC version 1.2 define URIs for symbols. A symbol is identified by the symbol name and content dictionary, which in turn is identified by the CD name and the CD base, i.e., the URI where the CD is located. From these constituents, symbol URIs are formed using URI fragments (the part after the # delimiter). However, OPENMATH imposes a one-CD-one-file restriction, which is too restrictive in general. While OMDOC1.2 permits multiple theories per file, it requires file-unique identifiers for all symbols. In both cases, the use of URI fragments, which are resolved only on the client, forces clients to retrieve the complete file even if only a single symbol is needed.

Furthermore, many module systems have features that impede or complicate the formation of canonical symbol URIs. Such features include unnamed imports, unnamed axioms, overloading, opening of modules, or shadowing of symbol names. Typically, this leads to a non-trivial correspondence between user-visible and application-internal identifiers. But this impedes or complicates cross-application scalability where all applications (ranging from, e.g., a Javascript GUI to a database backend) must understand the same identifiers.

MMT avoids the above pitfalls and introduces a simple yet expressive web-scalable syntax for symbol identifiers. An MMT-URI is of the form $doc?mod?sym$ where

- doc is a URI without query or fragment, e.g., `http://cds.omdoc.org/math/algebra/algegra1.omdoc` which identifies (but not necessarily locates) an MMT document,
- mod is a /-separated sequence of local names that gives the path to a nested theory in the above document, e.g., `Ring`,
- sym is a /-separated sequence $imp_1/\dots/imp_n/con$ of local names such that imp_i is an import and con a symbol name, e.g., `mult/mon/*`,
- a local name is of the form $pchar^+$ where $pchar$ is defined as in RFC 3986 [2], which — possibly via %-encoding — permits almost all Unicode characters.

In our running example, the canonical URI of multiplication in a ring is `http://cds.omdoc.org/math/algebra/algegra1.omdoc?Ring?mult/mon/*`. Note that the use of two `?` characters in a URI is unusual outside of MMT, but legal w.r.t. RFC 3986. Of course, MMT also defines relative URIs that are resolved against the URI of the containing declaration. The most important case is when doc is empty. Then the resolution proceeds as in RFC 3986, e.g., $?mod'?sym'$ resolves to $doc?mod'?sym'$ relative to $doc?mod?sym$ (Note that this differs from RFC 2396.). MMT defines some additional cases that are needed in mathematical practice and go beyond the expressivity of relative URIs: Relative to $doc?mod?sym$, the resolution of $??sym'$ and $?/mod'?sym'$ yields $doc?mod?sym'$ and $doc?mod/mod'?sym'$, respectively.

Case Study 3: URIs are the main data structure needed for cross-application scalability, and our experience shows that they must be implemented by almost every peripheral system, even those that do not implement MMT itself. Already at this point, we had to implement them in SML ([31]), Javascript ([13]), XQuery ([35]), Haskell (for Hets,

[21]), and Bean Shell (for a jEdit plugin) — in addition to the Scala-based reference API (Sect. 3).

This was only possible because MMT-URIs constitute a well-balanced trade-off between mathematical rigor, feasibility, and URI-compatibility: In particular, due to the use of the two separators / and ? (rather than only one), they can be parsed locally, i.e., without access to or understanding of the surrounding MMT document. And they can be dereferenced using standard URI libraries and URI-URL translations. At the same time, they provide canonical names for all symbols that are in scope, including those that are only available through imports.

3 A Scalable Implementation

As the implementation language for the MMT reference API, we pick Scala, a programming language designed to be *scala*ble ([24]). Being functional, Scala permits elegant code, and based on and fully compatible with Java, it offers cross-application and web-level scalability.

The MMT API implements the syntax and semantics of MMT. It compiles to a 1 MB Java archive file that can be readily integrated into applications. Library and documentation can be obtained from [30]. Two technical aspects are especially notable from the point of view of scalability. Firstly, all MMT functionality is exposed to non-Java applications via a scriptable shell and via an HTTP servlet. Secondly, the API maintains an abstraction layer that separates the backends that physically store MMT documents (URLs) from the document identifiers (URIs). Thus, it is configurable which MMT documents are located, e.g., in a remote database or on the local file system. In the following section we describe some of the advanced features.

3.1 Validation

Validation describes the process of checking MMT theory graphs against the MMT grammar and type system. MMT validation is done in three increasingly strict **stages**.

The first stage is XML validation against a context-free RelaxNG grammar. As this only catches errors in the surface syntax, MMT documents are validated **structurally** in a second stage. Structural validity guarantees that all declarations have unique URIs and that all references to symbols, theories, etc. can be resolved. This is still too lax for mathematics as it lacks type-checking. But it is exactly the right middle ground between the weak validation against a context-free grammar and the expensive and complex validation against a specific type system: On the one hand, it is efficient and foundation-independent, and on the other hand, it provides an invariant that is sufficient for many MKM services such as storage, navigation, or search.

Type-checking is foundation-specific, therefore each foundation must provide an MMT plugin that implements the typing and equality judgments. More precisely, the plugin must provide function that (semi-)decide for two given terms A and B over a theory T, the judgments $\vdash_T A = B$ and $\vdash_T A : B$. Given such a plugin, a third validation stage can refine structural validity by additionally validating well-typedness of all declarations. For scalability, it is important that (i) these plugins are stateless as the theory graph is maintained by MMT, and that the (ii) modular structure is transparent to the plugin. Thus plugin developers only need to provide the core algorithms for the specific type system, and all MKM issues can be relegated to dedicated implementations.

Context-free validation is well-understood. Moreover, MMT is designed such that foundation-specific validation is obtained from structural validation by using the same inference system with some typing and equality constraints added. This leaves structural validation as the central issue for scalability.

Case Study 4: We have implemented structural validation by decomposing an MMT theory graph into a list of atomic declarations. For example, the declaration $T = \{s_1 : \tau_1,\ s_2 : \tau_2\}$ of a theory T with two typed symbols yields the atomic declarations $T = \{\}, T?s_1 : \tau$, and $T?s_2 : \tau_2$. This "unnesting" of declarations is a special property of the MMT type system that is not usually found in other systems. It is possible because every declaration has a canonical URI and can therefore be taken out of its context.

This is important for scalability as it permits **incremental** processing. In particular, large MMT documents can be processed as streams of atomic declarations. Furthermore, the semantics of MMT guarantees that the processing order of these streams never matters if the (easily-inferrable) dependencies between declarations are respected. This would even permit parallel processing, another prerequisite for scalability.

3.2 Querying

Once a theory graph has been read, MMT provides two ways how to access it: MMT-URI dereferencing and querying with respect to a simple ontology.

Firstly, a theory graph always has two forms: the modular form where all nodes are partial theories whose declarations are computed using imports, and the flattened form where all imports are replaced with translated copies of the imported declarations. Many implementations of module systems, e.g., Isabelle's locales, automatically compute the flat form and do not maintain the modular form. This can be a threat to scalability as it can induce combinatorial explosion.

MMT maintains only the modular form. However, as every declaration present in the flat form has a canonical URI, the access to the flat form is possible via MMT-URI dereferencing: Dereferencing computes (and caches) the induced declarations present in the flat form. Thus, applications can ignore the modular structure and interact with a modular theory graph as if it were flattened, but the exponentially expensive flattening is performed transparently and incrementally.

Secondly, the API computes the ABox of a theory graph with respect to the MMT ontology. It has MMT-URIs as individuals and 10 types like `theory` or `symbol` as unary predicates. 11 binary predicates represent relations between individuals such as `HasDomain` relating an import to a theory or `HasOccurrenceOfInType` relating two symbols. These relations are structurally complete: The structure of a theory graph can be recovered from the ABox. The computation time is negligible as it is a byproduct of validation anyway.

The API includes a simple query language for this ontology. It can compute all individuals in the theory graph that are in a certain relation to a given individual. The possible queries include composition, union, transitive closure, and inverse of relations.

The ABox can also be regarded as the result of **compiling** an MMT theory graph. Many operations on theory graphs only require the ABox: for example the computation of the forward or backward dependency cone of a declaration which are needed to generate self-contained documents and in change management, respectively. This is important for cross-application scalability because applications can parse the ABox very easily. Moreover, we obtain a notion of **separate compilation**: ABox-generation only requires structural validity, and the latter can be implemented if only the ABoxes of the referenced files are known.

Case Study 5: Since all MMT knowledge items have a globally unique MMT-URI, being able to dereference them is sufficient to obtain complete information about a theory graph. We have implemented a web servlet for MMT that can act as a local proxy for MMT-URIs and as a URI catalog that maps MMT-URIs into (local or remote) URLs. The former means that all MMT-URIs are resolved locally if possible. The latter means that the MMT-URI of a module can be independent from its physical location. The same servlet can be run remotely, e.g., on the same machine as a mathematical database and configured to retrieve files directly from there or from other servers.

Thus systems can access all their input documents by URI via a local service, which makes all storage issues transparent. (Using presentation, see below, these can even be presented in the system's native syntax.) This solves a central problem in current implementations of formal systems: the restriction to in-memory knowledge. Besides the advantages of distributed storage and caching, a simple example application is that imported theories are automatically included when remote documents are retrieved in order to avoid successive lookups.

3.3 Presentation

MMT comes with a declarative language for notations similar to [19] that can be used to transform MMT theory graphs into arbitrary output formats. Notations are declared by giving parameters such as fixity and input/output precedence, and snippets for separators and brackets. Notations are not only used for mathematical objects but also for all MMT expressions, e.g. theory declarations and theory graphs.

Two aspects are particularly important for scalability. Firstly, sets of notations (called *styles*) behave like theories, which are sets of symbols. In particular, styles and notations have MMT-URIs (and are part of the MMT ontology), and the MMT module system can be used for inheritance between styles.

Secondly, every MMT expression has a URI E, for declarations this is trivial, for most mathematical objects it is the URI of the head symbol. Correspondingly, every notation must give an MMT-URI N, and the notation is applicable to an expression if N is a prefix of E. More specific notations can inherit from more general ones, e.g., the brackets and separators are usually given only once in the most general notation. This simplifies the authoring and maintenance of styles for large theory graphs significantly.

Case Study 6: In order to present MMT content as, e.g., HTML with embedded presentation MATHML, we need a style with only the 20 generic notations given in `http://alpha.tntbase.mathweb.org/repos/cds/omdoc/mathml.omdoc`.

```
<notation for="http://cds.omdoc.org/"
          role="constant">
  <element name="mo">
    <attribute name="xref">
      <text value="#"/><id/>
    </attribute>
    <hole><component index="2"/></hole>
  </element>
</notation>
```

For example, the notation declaration on the right applies to all constants whose `cdbase` starts with `http://cds.omdoc.org/` and renders `OMS` elements as `mo` elements. The latter has an `xref` attribute that links to the parallel markup (which is included by notations at higher levels). The content of the `mo` elements is a "hole" that is by default filled with the second component, for constants that is the name (0 and 1 are `cdbase` and `cd`.).

This scales well because we can give notations for specific theories, e.g., by saying that $?Magma?*$ is associative infix and optionally giving a different operator symbol than $*$. We can also add other output formats easily: Our implementation (see [18]) extends the above notation with a `jobad:href` attribute containing the MMT-URI — this URI is picked up by our JOBAD Javascript ([13]) for hyperlinking.

4 A Scalable Database

The TNTBase system [34] is a versioned XML-database with a client-server architecture. It integrates Berkeley DB XML into a Subversion server. DB XML stores HEAD revisions of XML files; non-XML content like PDF, images or LaTeX source files, differences between revisions, directory entry lists and other repository information are retained in a usual SVN back-end storage (Berkeley DB in our case). Keeping XML documents in DB XML allows accessing files not only via any SVN client but also through the DB XML API that supports efficient querying of XML content via XQuery and (versioned) modification of that content via XQuery Update.

In addition, TNTBase provides a plugin architecture for document format-specific customizations [35]. Using OMDOC as concrete syntax for MMT and the MMT API as a TNTBase plugin, we have made TNTBase MMT-aware so that data-intensive MMT algorithms can be executed within the database.

The TNTBase system and its documentation are available at `http://tntbase.org`. Below we describe some of the features particularly relevant for scalability.

4.1 Generating Content

Large scale collaborative authoring of mathematical documents requires **distributed** and **versioned** storage. On the language end, MMT supports this by making all identifiers URIs so that MMT documents can be distributed among authors and networks and reference each other. On the database end, TNTBase supports this by acting as a versioned MMT database.

In principle, versioning and distribution could also be realized with a plain SVN server. But for mathematics, it is important that the storage backend is aware of at least some aspects of the mathematical semantics. In large scale authoring processes, an important requirement is to guarantee consistency, i.e., it should be possible to reject commits of invalid documents. Therefore, TNTBase supports document format-specific **validation**.

For scalability, it is crucial that validation of interlinked collections of MMT documents is **incremental**, i.e., only those documents added or changed during a commit are validated. This is a significant effect because the committed documents almost always import modules from other documents that are already in the database, and these should not be revalidated — especially not if they contain unnecessary modules that introduce further dependencies.

Therefore, we integrate MMT separate compilation into TNTBase. During a commit TNTBase validates all committed files structurally by calling the MMT API. After successful validation, the ABox is generated and immediately stored in TNTBase. References to previously committed files are not resolved; instead their generated ABox is used for validation. Thus, validation is limited to the committed documents.

Case Study 7: In the LATIN project [18], we create an atlas of logics and logic translations formalized in MMT. At the current early stage of the project 5 people are actively editing so far about 100 files. These contain about 200 theories and 50 views, which form a single highly inter-linked MMT theory graph. We use TNTBase as the validity-guaranteeing backend storage.

The LATIN theory graph is highly modular. For example, the document giving the set-theoretical model theory of first-order logic from [16] depends on about 100 other theories. (We counted them conveniently using an XQuery, see below.) Standalone validation of this document takes about 15 seconds if needed files are retrieved from a local file system. Using separate compilation in TNTBase, it is almost instantaneous. In fact, we can configure TNTBase so that structural validation is preceded by RelaxNG validation. This permits the MMT application to drop inefficient checks for syntax errors. Similarly, structural validation could be preceded by foundation-specific validation, but often we do not have a well-understood notion of separate compilation for specific foundations. But even in that case, we can do better than naive revalidation. MMT is designed so that it is foundation-independent which modules a given document depends on. Thus, we can collect these modules in one document using an efficient XQuery (see below) and then revalidate only this document. Moreover, we can use the presentation algorithm from Sect. 3.3 to transform the generated document into the input syntax of a dedicated implementation.

4.2 Retrieving Content

While the previous section already showed some of the strength of an MMT-aware TNTBase, its true strength lies in retrieving content. As every XML-native database, TNTBase supports XQuery but extends the DB XML syntax by a notion of file system path to address path-based collections of documents. Furthermore, it supports indexing to improve performance of queries and the querying of previous revisions. Finally, custom XQuery modules can be integrated into TNTBase.

MMT-aware retrieval is achieved through two measures. Firstly, **ABox caching** means that for every committed file, the MMT ABox is generated and stored in TNTBase. The ABox contains only two kinds of declarations — instances of unary and binary predicates — and is stored as a simple XML document. The element types in these documents are **indexed**, which yields efficient global queries.

Example 3. An MMT document for the algebra example from Sect. 2.1 is served at http://alpha.tntbase.mathweb.org/repos/cds/math/algebra/algebra1.omdoc. Its ABox is cached at http://alpha.tntbase.mathweb.org:8080/tntbase/cds/restful/integration/validation/mmt/content/math/algebra/algebra1.omdoc.

Secondly, custom **XQuery functions** utilize the cached and indexed ABoxes to provide efficient access to frequently needed aggregated documents. These include in particular the forward and backward dependency cones of a module. The backward dependency cone of a module M is the minimal set of modules needed to make M well-formed. Dually, the forward cone contains all modules that need M. If it were not for the indexed ABoxes, the latter would be highly expensive to compute: linear in the size of the database.

Case Study 8: The MMT presentation algorithm described in Sect. 3.3 takes a set of notations as input. However, additional notations may be given in imported theories, typically format-independent notations such as the one making ?`Magma`?$*$ infix. Therefore, when an MMT expression is rendered, all imported theories must be traversed for the sole reason of obtaining all notations.

Without MMT awareness in TNTBase, this would require multiple successive queries which is particularly harmful when presentation is executed locally while the imported theories are stored remotely. But even when all theories are available on a local disk, these successive calls already take 1.5 seconds for the above algebra document. (Once the notations are retrieved, the presentation itself is instantaneous.)

In MMT-aware TNTBase, we can retrieve all notations in the backward dependency closure of the presented expression with a single XQuery. ABox-indexing made this instantaneous up to network lag.

TNTBase does not only permit the efficient retrieval of such generated documents, but it also permits to commit edited versions of them. We call these **virtual documents** in [35]. These are essentially "XML database views" analogous to views in relational databases. They are editable, and TNTBase transparently patches the differences into the original files in the underlying versioning system.

Case Study 9: While manual refactoring of large theory graphs is as difficult as refactoring large software, there is virtually no tool support for it. For MMT, we obtain a simple renaming tool using a virtual document for the one-step (i.e., non-transitive) forward dependency cone of a theory T (see [35] for an example). That contains all references to T so that T can be renamed and all references modified in one go.

5 Conclusion and Future Work

This paper aims to pave the way for MKM "in the large" by proposing a theoretical and technological basis for a "Universal Digital Mathematics Library" (UDML) which has been touted as the grand challenge for MKM. In a nutshell, we conclude that the problem of scalability has be to addressed on all levels: we need modularity and accessibility of induced declarations in the representation format, incrementality and memoization in the implementation of the fundamental algorithms, and a mass storage solution that supports fragment access and indexing. We have developed prototypical implementations and tested them on a variety of case studies.

The next step will be to integrate the parts and assemble a UDML installation with these. We plan to use the next generation of the OMDOC format, which will integrate the MMT infrastructure described in this paper as an interoperability layer; see [20] for a discussion of the issues involved. In the last years, we have developed OMDOC translation facilities for various fully formal theorem proving systems and their libraries. In the LATIN project [18], we are already developing a graph of concrete "logics-as-theories" to make the underlying logics interoperable.

References

1. Autexier, S., Hutter, D., Mantel, H., Schairer, A.: Towards an Evolutionary Formal Software-Development Using CASL. In: Bert, D., Choppy, C., Mosses, P.D. (eds.) WADT 1999. LNCS, vol. 1827, pp. 73–88. Springer, Heidelberg (2000)
2. Berners-Lee, T., Fielding, R., Masinter, L.: Uniform Resource Identifier (URI): Generic Syntax, RFC 3986, Internet Engineering Task Force (2005)
3. Bertot, Y., Castéran, P.: Coq'Art: The Calculus of Inductive Constructions. Springer, Heidelberg (2004)
4. Bourbaki, N.: Theory of Sets. In: Elements of Mathematics, Springer, Heidelberg (1968)
5. Bourbaki, N.: Algebra I. In: Elements of Mathematics, Springer, Heidelberg (1974)
6. Buswell, S., Caprotti, O., Carlisle, D., Dewar, M., Gaetano, M., Kohlhase, M.: The Open Math Standard, Version 2.0. Technical report, The Open Math Society (2004), `http://www.openmath.org/standard/om20`
7. CoFI, The Common Framework Initiative. In: CASL Reference Manual. LNCS, vol. 2960. Springer, Heidelberg (2004)
8. Curry, H., Feys, R.: Combinatory Logic. North-Holland, Amsterdam (1958)
9. Dumbrava, S., Horozal, F., Sojakova, K.: A Case Study on Formalizing Algebra in a Module System. In: Rabe, F., Schürmann, C. (eds.) Workshop on Modules and Libraries for Proof Assistants. ACM International Conference Proceeding Series, vol. 429, pp. 11–18 (2009)
10. Farmer, W.: An Infrastructure for Intertheory Reasoning. In: McAllester, D. (ed.) Conference on Automated Deduction, pp. 115–131. Springer, Heidelberg (2000)
11. Farmer, W., Guttman, J., Thayer, F.: Little Theories. In: Kapur, D. (ed.) Conference on Automated Deduction, pp. 467–581 (1992)
12. Farmer, W.M.: Mathematical Knowledge Management. In: Schwartz, D.G. (ed.) Mathematical Knowledge Management, pp. 599–604. Idea Group Reference (2005)
13. Giceva, J., Lange, C., Rabe, F.: Integrating Web Services into Active Mathematical Documents. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) Calculemus 2009. LNCS, vol. 5625, pp. 279–293. Springer, Heidelberg (2009)
14. Goguen, J., Winkler, T., Meseguer, J., Futatsugi, K., Jouannaud, J.: Introducing OBJ. In: Goguen, J., Coleman, D., Gallimore, R. (eds.) Applications of Algebraic Specification using OBJ, Cambridge (1993)
15. Horozal, F., Rabe, F.: Representing Model Theory in a Type-Theoretical Logical Framework. In: Fourth Workshop on Logical and Semantic Frameworks, with Applications. Electronic Notes in Theoretical Computer Science, vol. 256, pp. 49–65 (2009)
16. Horozal, F., Rabe, F.: Representing Model Theory in a Type-Theoretical Logical Framework. Under review (2010), `http://kwarc.info/frabe/Research/EArabe_folsound_10.pdf`
17. Howard, W.: The formulas-as-types notion of construction. In: To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism, pp. 479–490. Academic Press, London (1980)

18. Kohlhase, M., Mossakowski, T., Rabe, F.: The LATIN Project (2009), `https://trac.omdoc.org/LATIN/`
19. Kohlhase, M., Müller, C., Rabe, F.: Notations for Living Mathematical Documents. In: Autexier, S., Campbell, J., Rubio, J., Sorge, V., Suzuki, M., Wiedijk, F. (eds.) AISC 2008, Calculemus 2008, and MKM 2008. LNCS (LNAI), vol. 5144, pp. 504–519. Springer, Heidelberg (2008)
20. Kohlhase, M., Rabe, F., Sacerdoti Coen, C.: A Foundational View on Integration Problems (2010) (Submitted to CALCULEMUS)
21. Mossakowski, T., Maeder, C., Lüttich, K.: The Heterogeneous Tool Set. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 519–522. Springer, Heidelberg (2007)
22. Naumov, P., Stehr, M., Meseguer, J.: The HOL/NuPRL proof translator - a practical approach to formal interoperability. In: Boulton, R.J., Jackson, P.B. (eds.) TPHOLs 2001. LNCS, vol. 2152, p. 329. Springer, Heidelberg (2001)
23. Nipkow, T., Paulson, L., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. Springer, Heidelberg (2002)
24. Odersky, M., Spoon, L., Venners, B.: Programming in Scala. artima (2007)
25. Owre, S., Rushby, J., Shankar, N.: PVS: A Prototype Verification System. In: Kapur, D. (ed.) 11th International Conference on Automated Deduction (CADE), pp. 748–752. Springer, Heidelberg (1992)
26. Paulson, L.C.: Isabelle: A Generic Theorem Prover. LNCS, vol. 828. Springer, Heidelberg (1994)
27. Pfenning, F., Schürmann, C.: System description: Twelf - A meta-logical framework for deductive systems. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 202–206. Springer, Heidelberg (1999)
28. Poswolsky, A., Schürmann, C.: System Description: Delphin - A Functional Programming Language for Deductive Systems. In: Abel, A., Urban, C. (eds.) International Workshop on Logical Frameworks and Metalanguages: Theory and Practice. ENTCS, pp. 135–141 (2008)
29. Rabe, F.: Representing Logics and Logic Translations. PhD thesis, Jacobs University Bremen (2008), `http://kwarc.info/frabe/Research/phdthesis.pdf`
30. Rabe, F.: The MMT System (2008), `https://trac.kwarc.info/MMT/`
31. Rabe, F., Schürmann, C.: A Practical Module System for LF. In: Cheney, J., Felty, A. (eds.) Proceedings of the Workshop on Logical Frameworks: Meta-Theory and Practice (LFMTP), pp. 40–48. ACM Press, New York (2009)
32. Sannella, D., Wirsing, M.: A Kernel Language for Algebraic Specification and Implementation. In: Karpinski, M. (ed.) Fundamentals of Computation Theory, pp. 413–427. Springer, Heidelberg (1983)
33. Trybulec, A., Blair, H.: Computer Assisted Reasoning with MIZAR. In: Joshi, A. (ed.) Proceedings of the 9th International Joint Conference on Artificial Intelligence, pp. 26–28 (1985)
34. Zholudev, V., Kohlhase, M.: TNTBase: a Versioned Storage for XML. In: Proceedings of Balisage: The Markup Conference 2009, vol. 3, Mulberry Technologies, Inc. (2009)
35. Zholudev, V., Kohlhase, M., Rabe, F.: A (insert XML Format) Database for (insert cool application). In: Proceedings of XMLPrague, XMPPrague.cz (2010)

The Formulator MathML Editor Project: User-Friendly Authoring of Content Markup Documents

Andriy Kovalchuk, Vyacheslav Levitsky, Igor Samolyuk, and Valentyn Yanchuk

Zhytomyr State Technological University,
Chernyakhivskogo 103, 10005 Zhytomyr, Ukraine
info@mmlsoft.com
http://www.mmlsoft.com

Abstract. Implementation of an editing process for Content MathML formulas in common visual style is a real challenge for a software developer who does not really want the user to have to understand the structure of Content MathML in order to edit an expression, since it is expected that users are often not that technically minded. In this paper, we demonstrate how this aim is achieved in the context of the Formulator project and discuss features of this MathML editor, which provides a user with a WYSIWYG editing style while authoring MathML documents with Content or mixed markup. We also present the approach taken to enhance availability of the MathML editor to end-users, demonstrating an online version of the editor that runs inside a Web browser.

Keywords: Content MathML, mathematical formula, natural editing of algebraic expressions, model-based editor, online MathML equation editor.

1 Introduction

Modern standards for representations of mathematical knowledge and easily accessible software tools greatly benefit education, and scientific and technical publishing. For instance, a good number of software systems supporting the MathML standard have been helping to develop valuable distance learning models, web-based education, and electronic textbooks, which would not be available if one could only use static images, prepared in advance, instead of more natural ways for students and teachers to present and exchange formulas and diagrams. Such communication including mathematical data fosters an understanding between teachers and students when studying disciplines with strong mathematical backgrounds, since it is necessary not only to illustrate ideas, but also to have confirmation that the student has gained insight into the material.

A rendering of mathematical notation is not the only thing needed in such scenarios, because it is the underlying mathematical structure of an expression that must be considered (and maybe examined using additional software), and not any particular rendering of an expression. This means that, in the face of poor software support for mathematical content standards, this communication process takes a turn for the worse. There are good ways to represent and reuse mathematical content data with existing standards (the content markup part of MathML [1], OpenMath [2]), and so it is becoming increasingly important to find ways of expanding our ability to support these standards in software.

This paper presents a short description of one such software system, namely the Formulator MathML Editor Project [3] (`http://www.mmlsoft.com`). We demonstrate how support for a mathematical content standard is achieved in the context of Formulator project and discuss features of this MathML editor that provides a user with a WYSIWYG editing style while authoring MathML documents with content or mixed markup.

2 Formulator MathML Editor

2.1 Challenges of Content MathML Support

A lot of software systems for authoring mathematical documents are available, and many of them have support for MathML, either directly or through some conversion. Many references to well-known software tools and research or prototype projects can be found on the MathML Software list maintained by W3C [4]. However, if we have in mind support of mathematical content standards, it does not matter how many good products help one to type in a mathematical expression if they cannot help produce some mathematical content output.

Filtered by requiring support for Content MathML, the list still includes many interesting projects, e.g., the semantically oriented formula editor WIRIS OM Tools [5], Integre MathML Equation Editor [6], MathDox [10], and new attempts to bring an interactive editing process to the web, like the Connexions MathML Editor [7].

However, an exacting user, seeking to meet the requirements of a better and quicker interaction in the field of expressing mathematical ideas, would even now not be completely satisfied for different reasons in each case [8][16], and this is natural in view of intricacy of the task. One of the important causes of failure to satisfy a user's wishes is that there is more to making mathematical information a useful resource in interactive applications than merely bringing together a common set of components and attributes adopted from the standard. Visual representation and user friendliness of an interface are ultimately also expected by users. The main usability issue of existing software systems is that a user has to understand the structure of Content MathML in order to edit an expression, and this is a considerable disadvantage, since users are often not that technically minded.

An extensive review of common pitfalls and a comparison of behaviors for several editors for mathematical content is given in [8]. Among the issues mentioned there, an important place is occupied by the problem of exposing the internal document model to a user. Presentation-oriented editors can solve the problems of edit point support and understandable navigation rules easily, but such issues can be a serious obstacle for content editors trying to be user-friendly if they consider the underlying standard of mathematical content encoding to be a central source of the editing procedure.

True user friendliness cannot be achieved using the mechanical approach of just wrapping Content MathML constructs in buttons and menu items. In such systems, a developer allows a user to forget Content MathML entity names, but still forces the user to bear in mind the tree-like structures of a text while creating a mathematical expression, not to mention a need for post-editing. Obviously there is a big distance between a usual mathematical editing concentrated on the visual representation of an expression, and the confusing effort to mentally synchronize rendering and semantic aspects of an expression. The last differs quite dramatically both from our experience of using legacy systems with linear mathematical input, and from the exploitation of two-dimensional input forms usual for mathematical equation editors.

These obstacles are natural consequences of attempts to represent the semantics in a different way from purely mental exercises. Any kind of user friendliness in such matters entails compromises between a shape and contents. So, in addition to general problems of human-computer interaction, concerned software developers must address a number of supplementary issues. Demanding a well-formed and finished Content MathML document is not a great help to a developer who needs to build an interactive system. During the editing process, there are ambiguities which either can only be solved at the cost of a deeper analysis of the context, or are doomed to be left unresolved pending user hints. The first approach of context analysis leads to performance problems, the second is not user friendly. But to neglect resolving ambiguities is also not a good way if one is trying to provide lasting correctness of the mathematical content output.

Thus we have a classical triangle of alternatives, where software performance, permanent output correctness, and user friendliness occupy three corners, and existing systems far too often implement only a single edge in this triangle. While having every respect for the efforts and achievements of our colleagues in the field and by no means pretending to cover in full the triangle of alternatives for Content MathML editing, we hopefully have some advances in the Formulator MathML Editor Project which helps to bring a process of authoring MathML documents with content or mixed markup nearer to a user's needs.

2.2 Overview of the Formulator MathML Editor

The Formulator MathML Editor Project was started in 2003 as part of an on-site computer algebra system [11]. Since than the project has grown from a product-oriented mathematical expression editor to a set of desktop and online software tools for editing and rendering MathML documents. Over the years, a number of successful use cases have shown the suitability of the Formulator software for a wide audience of software developers, educators, authors, and students. Featured projects include, for instance, the Standards Unit Mathematics Project at the UK Department for Education and Skills [13], some regional education service centers and educational institutions in the USA and EU, and several commercial applications—National Instruments, Fast Track Systems, Neoptec, Ambow Education, etc.

The main components of the Formulator software are:

– *Formulator MathML Weaver:* a desktop application for WYSIWYG editing of Presentation and Content MathML documents. There is a proprietary version for MS Windows and an open source cross-platform version, available from SourceForge and Google Code sites.

– *Mathematical Templates Builder:* an utility for customizing and amending a dictionary of mathematical symbols and templates. This tool ensures Formulator openness in the sense that a user can change the look and style of a set of existing mathematical constructs without accessing the software source code. This is possible because templates are explicitly defined and can be edited in text form, and a simple built-in language is available to specify the dynamic behavior of graphics and to edit input slots in some complicated situations. During a run-time the built-in language provides a way to calculate coordinates and sizes of graphics objects and edit boxes, margins of a template and its child objects. The simple structure of the language guarantees its fast run-time interpretation and leads to effective rendering of equations.

– *Formulator ActiveX Control and API:* component editions of the desktop editor which are intended for a software developer who needs to build an application aware of the mathematical typesetting and semantic rules.

– *Formulator IE Performer:* a plug-in for Internet Explorer to render MathML fragments inside web pages (similar to MathPlayer [12]).

– *Online MathML Editor:* a MathML editor in the form of a distributed web application that should be run inside an Internet browser.

Considered in the role of a presentation formula editor Formulator is similar to a majority of mathematical expression editors embedded in office products such as OpenOffice.org and MS Office, and to well-known WYSIWYG formula editors, for example, MathType [12]. Among its notable supplements are several dedicated model views for clearer understanding and fine tuning of the MathML document structure, and additional tools for developers and advanced users to enable enhancing the editor's functionality and to build new software applications which use the formula editor.

As a content editing system the Formulator software is far from being completely satisfactory yet, and we definitely lack feedback from existing users worldwide and large-scale user studies. On the other hand, a frame of reference can be achieved using comparison with content editors mentioned earlier, from the methodological and survey papers such as [8][9], from feedback about Formulator's suitability for learners (the Autograph Maths software [15]) and instructors (the Connexions project [14]).

As a mathematical content editor the Formulator project is designed from the start as an intuitive and visually oriented tool. It offers what the paper [9] calls "natural editing of algebraic expressions" with the operations, natural for a presentation formula editor, of insertion, deletion, selection, cut, copy, paste, drag and drop.

The modes of editing and navigation are consistent with the "what-you-see-is-what-you-get" principle, supporting edit-point feedback and accessibility, geometric moves, reversing of moves, deterministic moves, slot navigation and selection flexibility [8]. In addition to simple "natural editing" procedures, the Formulator editor has support for advanced modes, which are described in [9] for the Aplusix software and WIRIS OM Tools: syntactically basic enhancements (recognition of mathematical operators and parentheses, arity representation), enhanced backspace and delete operations, algebraic selections, and automatic bracketing in accordance with changing operator precedence.

The model chosen for the Formulator editor to enhance backspace, delete and insert operations differs from that proposed in [9]. The latter approach has an advantage in the case of unary operators and derives a deeper benefit from insights into user intentions, for instance, in manipulations with selections and operand movements. The Formulator MathML editor employs semantic information about a document in a different way, for example, by evaluating simple Content MathML formulas and formula chains with user variables. We expect to enhance the editing process in the Formulator editor further by additional utilization of semantic information.

2.3 General Approach to the Document Structure

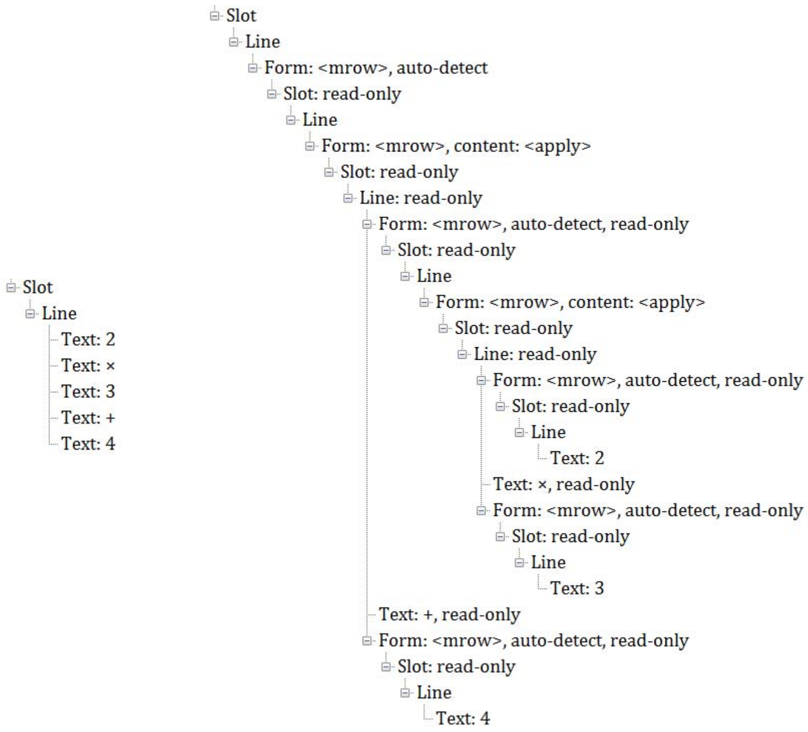
The nature of Content MathML differs radically from Presentation MathML, as they are used to define different aspects of mathematics. Presentation elements describe the visually important two-dimensional hierarchical forms and thus give more or less precise instructions on how to render and how to edit mathematical constructs. Content markup follows closely the semantic structure of mathematical objects. Consolidation of these two quite different formats into one document structure seems overly complicated from a software implementation point of view, and it appears a good idea to have one dominant format.

Since initially the Formulator MathML editor was as a tool for Presentation MathML authoring, our aim was to bring semantics into existing visual editing and to retain user friendliness. In this way, the document structure in Formulator has support for Presentation MathML, both encouraging development of the editing process of Content MathML formulas in a common visual style without exposing implementation details of the Content MathML format, and keeping good software performance while rendering and editing the document. The internal document model is a tree of four basic node types, holding additional attributes as reference information: (1) an input slot, (2) a line of horizontally neighboring text and formula nodes, (3) a formula, composed of input slots and graphics, and (4) plain text.

This document structure allows carrying out the initial task of creating Content MathML expressions by relating each semantic construct with corresponding visual elements and by adding supplementary nodes, which are invisible either from rendering or from the semantic point of view. Further free-style operations also require more labeling throughout a document to account for intermediate editing states, which inevitably break the proper Content MathML format. Thus a compromise must be found between maintaining semantic correctness of the document model and user operations which are incompatible with expression semantics (e.g., brackets which are not balanced).

For instance, in the context of Presentation MathML the formula “$2 \times 3 + 4$” is presented by the trivial hierarchy of fig. 1a, but an essentially more verbose document structure is needed to implement the free style of editing in the context of Content MathML, fig. 1b.

In addition to more complicated document structure, there are attributes and hidden reference nodes, which guarantee correctness of the formula during



(a) **(b)**

Fig. 1. Document structure for a formula “$2 \times 3 + 4$” in the context of Presentation MathML (a) and Content MathML (b). After each formula node, there is an indication of a Presentation MathML tag that is used by a rendering procedure and either a type of the node in the Content MathML context, or an “auto-detect” node attribute. Read-only nodes cannot be either deleted or moved by a user.

editing actions and are used to produce proper Content MathML output from this structure. E.g., “do not edit” and “do not move” attributes are used to preserve the mathematical template and to provide a user with comfortable navigation through nodes (no “fake moves” [8], each user action corresponds to an understandable movement of the caret marking the insertion point). Some formula nodes are marked with “auto-detect self entity by contents” attributes and do not recognize their type until a user finishes editing and wants to save or to export a formula to MathML. The combination of these attributes allows one to remove parts of a formula and to insert quite different kinds of elements, for instance, a division operator instead of a token element, and still to have a proper output. The “auto-detect” attribute is one of the reasons for the verbosity in a document structure, but at the same time it is a way to avoid the rigidity of a template-based editing procedure.

2.4 Improving Usability for a Template-Based Approach

Earlier versions of the Formulator MathML editor implemented a template-based approach to Content MathML creation, where each new construct was chosen from a palette of atomic formulas and had to be placed into some input slot of previously used constructs.

Users in experiments we conducted with creating and post-editing of content formulas gave us feedback on usability issues, concentrating on the uncomfortable rigidity of templates with a read-only structure and several edit spots, and on the confusing view of the editing process as building a tree. On the other hand, just considering this approach to be faulty seems to be going too far. Amongst the good ideas of a template-based approach are supporting a user with run-time information about proper formula structures, quick construction of new formulas from basic building blocks, and an editing style that facilitates general correctness of a document from a semantic point of view.

In moving towards improved usability of Content MathML support, we prefer evolutionary changes rather than revolutionary ones, and so combine the existing positive features of the template-based approach with a newly implemented free style of editing. The change is to replace rigid templates with a more fluid structure that can be changed by a user, instead of the former frustrating experience of having to delete a template as a whole.

An important aspect of this proposed free-style editing is that changing the structure of a Content MathML formula is closer to the behavior of a state machine than, for instance, to plain text editing. This means that we preserve run-time hints and explicit correctness of the template-based approach by transforming one proper structure of an expression to another one as a response to user insertions and deletions.

Such a process seems more reliable and predictable in comparison with encouraging a user to go ahead and to break down an expression, with the hope that later the user will manage to reassemble the expression in a semantically proper way. The latter would be more like a computer programming, and maybe that is too much to expect from not so technically minded users.

2.5 Examples of Free-Style Editing

In order to realize our conception of free-style editing, we paid attention to such important aspects of the input system as random post-editing, insertion, deletion of mathematical operators and whole expressions, automatic completion of an expression, and a model for bracket editing.

For instance, once an operator has been entered it was impossible in the earlier versions to change or delete it without deleting the entire sub-tree containing it. If a user entered "3+2" and then wanted to change it to "3−2", it was impossible to position the cursor at the '+' operator to change it to a '−'. Instead the whole template of "3+2" had to be deleted and re-entered. Now a user can edit initially read-only mathematical operators by deleting them (a black box appears instead, standing for "no operation", see fig. 2b), and later typing in a new mathematical operator. This black box preserves the correct expression structure, hints that the editing process should be continued to recover expression correctness, and allows a user to change it into a needed mathematical operator from a palette, while still being protected from unconditional deletion from the expression.

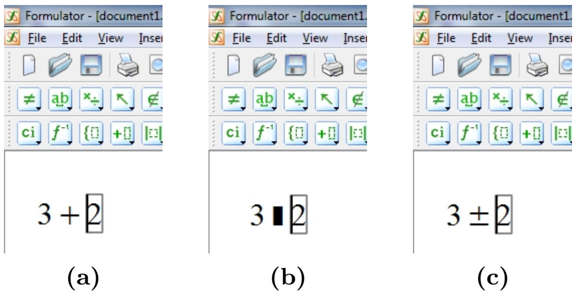


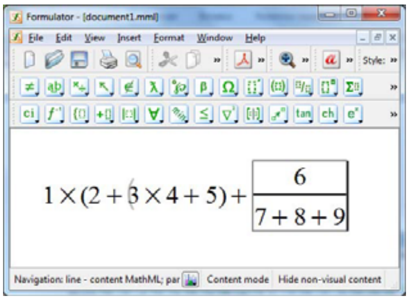
Fig. 2. These snapshots demonstrate how a formula can be edited without breaking the underlying template. Transition from case (a) to case (b) occurs after Backspace is pressed, and after typing '+' and '−' in sequence the case (c) appears. The latter transformation also demonstrates a feature of automatic replacement.

This example shows also that a sequence of characters can be converted (where it is possible) to a mathematical operator that has no direct equivalent on a keyboard. For instance, by typing the '<' operator and then immediately typing the '=' operator, we will get a new '≤' operator. The feature of automatic replacement does not exclude any subset of proper expressions, since it can be undone if not wanted, thus leading to more complicated structures with nested mathematical operations.

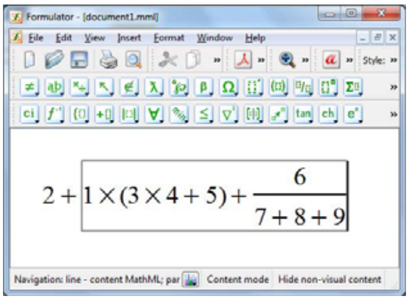
Another example of free-style editing is changing the structure of a formula by explicit insertion or deletion of brackets (in contrast to the implicit brackets used where required by a combination of the existing structure of an expression and mathematical precedence information). What we are looking for is an editing procedure that is as similar to linear text entry as possible, both for the initial input of an expression and the subsequent editing of it. Again, since we do not want the user to have to understand the structure of Content MathML, the main challenges are:

– to save valid MathML even when a user opens brackets and does not close them, and
– to find a proper transition from an old formula structure to a new one after a selection is made that should be inserted into brackets.

The second problem (i.e., genuine changing of a formula structure) seems to have no universal solution because of possible ambiguities. Thus, a user can break an

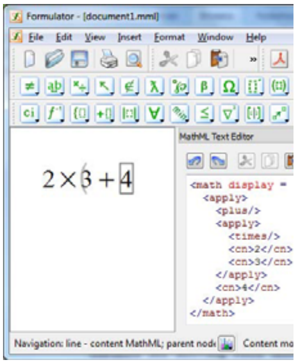

Formulator - [document1.mml]
File Edit View Insert Format Window Help
Style:
$1\times(2+3\times4+5)+\frac{6}{7+8+9}$
Navigation: line - content MathML; par
Content mode
Hide non-visual content

(a)


Formulator - [document1.mml]
File Edit View Insert Format Window Help
Style:
$2+1\times(3\times4+5)+\frac{6}{7+8+9}$
Navigation: line - content MathML; par
Content mode
Hide non-visual content

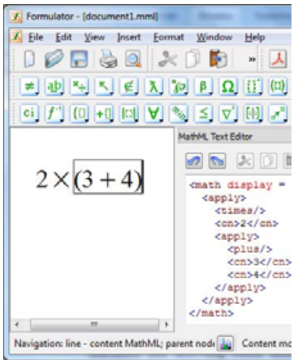
(b)

Fig. 3. These snapshots demonstrate semi-transparent Content MathML rendering of brackets and highlights some possible ambiguities in arbitrary bracketing. Case (a) has an unbalanced bracket to the left of number 3, and by identifying the left edge of a newly created bracketed expression to be inside existing brackets and the right edge to be on the rightmost end of the whole expression, a user splits the former expression with a result, case (b), that at first is not evident, but in a sense is logical.


Formulator - [document1.mml]
File Edit View Insert Format Window Help
MathML Text Editor
$2\times3+4$
Navigation: line - content MathML; parent node
Content mo

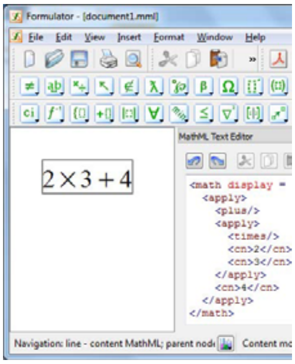
```
<math display =
  <apply>
    <plus/>
    <apply>
      <times/>
      <cn>2</cn>
      <cn>3</cn>
    </apply>
    <cn>4</cn>
  </apply>
</math>
```

(a)


Formulator - [document1.mml]
File Edit View Insert Format Window Help
MathML Text Editor
$2\times(3+4)$
Navigation: line - content MathML; parent node
Content mo

```
<math display =
  <apply>
    <times/>
    <cn>2</cn>
    <apply>
      <plus/>
      <cn>3</cn>
      <cn>4</cn>
    </apply>
  </apply>
</math>
```

(b)


Formulator - [document1.mml]
File Edit View Insert Format Window Help
MathML Text Editor
$2\times3+4$
Navigation: line - content MathML; parent node
Content mo

```
<math display =
  <apply>
    <plus/>
    <apply>
      <times/>
      <cn>2</cn>
      <cn>3</cn>
    </apply>
    <cn>4</cn>
  </apply>
</math>
```

(c)

Fig. 4. These snapshots demonstrate how structural changes in bracketed formula can be reverted using deletion. Transition from the case (a) to (b) occurs after ')' is pressed at the right of the formula, and if a user presses Backspace when a caret is in this position, the case (c) appears. The latter snapshot corresponds to the structure of case (a) as we can see from plain text of Content MathML output.

existing bracketed expression by enclosing only a part of this expression in a new pair of brackets along with the rest of the formula that initially lies outside the previously existing brackets (fig. 3b). Nevertheless, Formulator MathML editor implements a smart method for solving such ambiguities and where it is possible (in most cases, as our practice shows) a formula structure is changed predictably and clearly for a user.

The transitions of a formula's structure discussed above can be reverted by using further editing operations. Although a bracketed expression is treated in accordance with the underlying template as having read-only brackets and several input slots inside them, this structure is not rigid and can be altered if a user presses the Delete button and a caret is standing before the left bracket, or if a user presses the Backspace button and a caret is standing after the right bracket (fig. 4).

2.6 Supporting Users Accustomed to Legacy Input

There is one more enhancement of the Content MathML editing process in the Formulator MathML Editor project that can be valuable for users accustomed to legacy systems with a plain text mathematical input. This means a procedure similar to linear input that tries to express mathematical equations using just keyboard characters, for instance, the circumflex '^' sign for an exponentiation and the '/' sign for a fraction or division, as in "$y = 1/x\hat{}2$". Support for this input mode allows an easy transition to the Formulator entry system for users who are used to the legacy style, and also allows faster and easier input in some special cases using only a keyboard without a mouse.

Improved usability of this sort means that users are able to enter expressions intuitively with a sequence of key presses. For instance, the equation $y = \sin x$ should be entered by pressing 'y', '=', 's', 'i', 'n', 'x', and an equation $y = 2a^2b$ should be entered by pressing: 'y', '=', '2', 'a', 'a', 'b'. No mathematical operator buttons or screen forms are used in either example, but as a trade-off for using such shortcuts we assume that a user has only variables of one-letter length. In addition to the legacy input style, there are also a few special cases, where a certain sequence of key presses can generate slightly different but more intuitive input strings (for instance, several sequential presses of the letter 'a' generate exponentiation instead of multiplication).

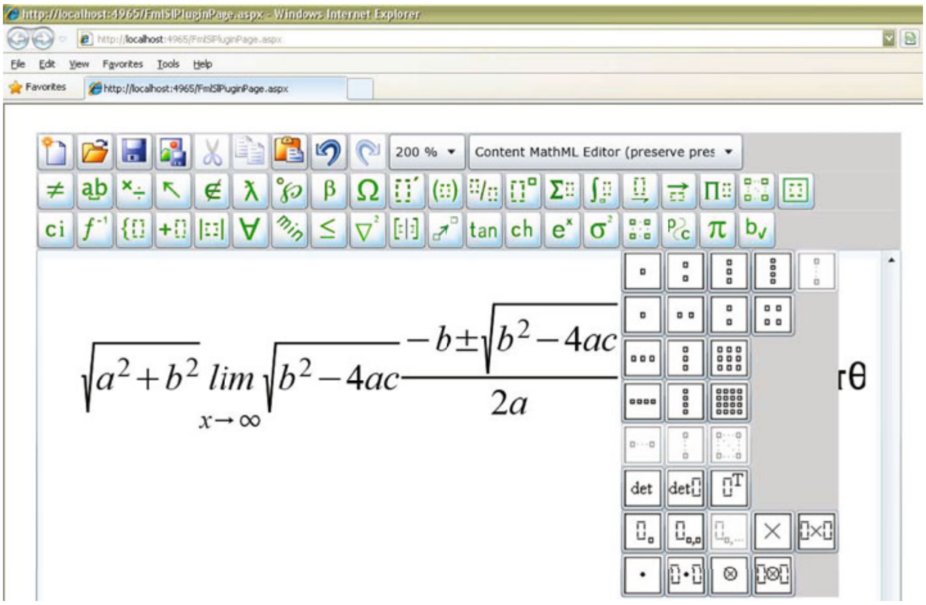
More is expected, and thus user friendly behavior is also provided by using information about operator precedence to automatically help a user to navigate through the text; namely, the editor automatically shifts a caret forward in situations where a user of a legacy software would expect this. For instance, when a user types 'y', '=', '1', '/', 'x', '+, '1', in basic editing mode the formula would be considered as "$y = 1/(x + 1)$", since a caret is near the 'x' position and the '+' operator is normally associated with a caret focus. In order to meet user expectations this rule is adjusted during legacy style input (by considering precedences of division and addition operators), so that we get exactly the formula "$y = 1/x + 1$", as a user would expect who is used to textual mathematical input.

3 Availability and Future Work

Guided by a vision of public accessibility of technologies for learning activities creating and authoring documents containing mathematical data, the Formulator MathML Editor project is evolving to enhance its availability to different kinds of end-users, from students to software developers.

The main steps toward this goal have been:

1. Porting the Formulator MathML Editor project to Mac OS X and Linux operating systems (in addition to MS Windows that was supported initially) and issuing this new version of the Formulator MathML Editor project with an open source license.
2. Developing an online version of the editor that runs inside a Web browser in a form of Rich Internet Application (fig. 5).


http://localhost:4965/FmlSIPluginPage.aspx - Windows Internet Explorer
File Edit View Favorites Tools Help
Favorites
200 %
Content MathML Editor (preserve pres

$$\sqrt{a^2+b^2}\ \lim_{x\to\infty}\sqrt{b^2-4ac}\,\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

Fig. 5. The test page of the online version of the Formulator MathML Editor. A link to the page is available from `http://www.mmlsoft.com`

The second step is the more challenging, because basically the editor uses native code to get more performance and a better appearance. It turns out that a good way to solve the problem lies in using a browser plug-in (in contrast to a scripting language approach) as a client of the editor's algorithmic core that runs remotely as a native application. The general picture of this solution includes a server part and a thin Silverlight client working inside a browser; in an ideal case this should not depend on the operating system at all. Actually, this scheme is too good to be true, and in a real world scenario the user is limited to a combination of Internet browser and operating system that already has support for the Silverlight plug-in. Furthermore, there are some delicate issues where the choice of a browser does matter (for example, when working with the Clipboard of an operating system).

It is important that generally the online Formulator editor not be restricted to the Silverlight plug-in. It should work with any other similar technology as well. The choice of the browser plug-in approach (in contrast to a scripting language method) is a more fundamental point than the choice of a Silverlight, Flash or some other plug-in.

Further improvements in the efficiency and functionality of this online version of the Formulator MathML Editor are matters for future work, although tests which have been conducted show acceptable performance for the core editing functions already implemented. Moreover, it seems that by installing a server part within a Local Area Network of an organization where Formulator is used even more efficient behavior can be achieved. Currently, the online version of the Formulator MathML editor is available on our test server, and the set of features implemented is already sufficient to perform most popular operations, such as: to create/open/save formulas in Presentation, Content and mixed MathML markup; to export formulas to MathML text or to an image; to cut/copy/paste formulas.

Future plans include making a thicker client, and evolving the server part to a more usable environment, for instance, closer to online word processor functionality. One more interesting opportunity that is opened after creating the Silverlight version of the Formulator MathML software is to build more complicated scenarios for education above a layer of online mathematical editing, for instance, to support interactive online learning activity.

References

1. Mathematical Markup Language (MathML) Version 2.0 (Second Edition) W3C Recommendation (October 21, 2003), `http://www.w3.org/TR/MathML2`
2. OpenMath, `http://www.openmath.org`
3. Kovalchuk, A., Levitsky, V., Samolyuk, I., Yanchuk, V.: Establishing Comprehensive Interface for Authoring Mathematical Expressions with the Formulator, MathML Equation Editor. In: Wspolczesne problemy informatyki: Problemy analizy i projektowania sieci komputerowych, Legnica, pp. 47–58 (2005)
4. MathML Software List, `http://www.w3.org/Math/implementations.html`
5. Marquès, D., Eixarch, R., Casanellas, G., Martinez, B.: WIRIS OM Tools: A Semantic Formula Editor. In: Proceedings of the 2006 Mathematical User-Interfaces Workshop, St Anne's Manor, Wokingham (2006)
6. Integre MathML Equation Editor, `http://www.integretechpub.com/zed/`
7. Connexions MathML Editor, `http://cnx.org/help/matheditor`
8. Padovani, L., Solmi, R.: An Investigation on the Dynamics of Direct-Manipulation Editors for Mathematics. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 302–316. Springer, Heidelberg (2004)

9. Nicaud, J.-F.: Natural Editing of Algebraic Expressions. In: Proceedings of the Mathematical User-Interfaces Workshop (MathUI 2007) at the 6th Mathematical Knowledge Management Conference, Schloss Hagenberg, Linz (2007), http://www.activemath.org/workshops/MathUI/07/proceedings
10. Cohen, A.M., Cuypers, H., Jibetean, D., Spanbroek, M.: Interactive learning and mathematical calculus. In: Kohlhase, M. (ed.) MKM 2005. LNCS (LNAI), vol. 3863, pp. 330–345. Springer, Heidelberg (2006)
11. Kolodnytsky, M., Kovalchuk, A., Kuryata, S., Levitsky, V.: The Mathematical Software Implementation for Computational Algebra and Number Theory. In: Proceedings of the 4th Asian Symposium on Computer Mathematics 2000, Chiang Mai, Thailand, pp. 291–294 (2000)
12. Design Science. MathType, MathPlayer software, http://www.dessci.com
13. Swan, M.B.: Improving learning in mathematics: challenges and strategies. Department for Education and Skills, UK (2005)
14. The Connexions project, http://cnx.org
15. The Autograph Maths software, http://www.autograph-maths.com
16. Dragunov, A.N., Herlocker, J.L.: Designing intelligent and dynamic interfaces for communicating mathematics. In: Proceedings of the 2003 International Conference on Intelligent User Interfaces, Miami, USA, pp. 236–238 (2003)

Notations Around the World: Census and Exploitation

Paul Libbrecht

DFKI GmbH and University of Saarland, Saarbrücken, Germany
paul@{activemath.org,dfki.de}

Abstract. Mathematical notations around the world are diverse. Not as much as requiring computing machines' makers to adapt to each culture, but as much as to disorient a person landing on a web-page with a text in mathematics.

In order to understand better this diversity, we are building a census of notations: it should allow any content creator or mathematician to grasp which mathematical notation is used in which language and culture. The census is built collaboratively, collected in pages with a given semantic and presenting observations of the widespread notations being used in existing materials by a graphical extract. We contend that our approach should dissipate the fallacies found here and there about the notations in "other cultures" so that a better understanding of the cultures can be realized.

The exploitation of the census in the math-bridge project is also presented: this project aims at taking learners "where they are in their math-knowledge" and bring them to a level ready to start engineering studies. The census serves as definitive reference for the transformation elements that generate the rendering of formulæ in web-browsers.

1 Introduction

This paper reminds the great variation in the notations used in mathematical texts in many cultures: in order to obtain knowledge of this diversity, we propose to realize a census of the mathematical notations made of concrete observations from multiple texts. This census allows us to reach verifiable statements about the notations in wide use.

Misconceptions about widely-used notations are quite easy to reach: indeed, it is almost impossible to obtain persons that master the mathematical notations of several languages. Thus, one can find in several texts about mathematical notations the fact that the binomial coefficient in Russian is written as C_k^n whereas the binomial coefficient in French is written as C_n^k, i.e. they are opposite of each other. This is the case of [CIMP01], [Koh06], and [LAG09]. As the authors could observe and proof in many russian sources, this turned out to be false: the binomial coefficient in Russian is written the same as in French. Tracking down the source of this confusion turned out to be impossible with the inability to verify the origin of such a claim and with, even, easy assertions about the pre-industrial era where such a notation was carried between France and Russia.

The census we propose should avoid such misconceptions by the maintenance of web-pages that link to textbooks that are commonly used hence should allow to dissipate any doubt about notations in wide usage in different cultures even if it is not well known.

We also aim at exploiting the census for the presentation system of ActiveMath, a web-based environment that renders mathematical formulæ from a semantic source into a presentation suitable for the learner. The Math-Bridge project's objective is to start where the learners start: in their own mathematical culture (the background can be quite different as described in [MGLU09] and [LCME$^+$08], and bridge the gap to reach university-entry-level mathematics. Therefore, a census of college-level text-books' notations for the languages of Math-Bridge is useful: English, German, French, Spanish, Finnish and Hungarian.

In this paper we explain how notations are exploited to create the necessary information that instructs the ways of rendering of the presentation system of ActiveMath: it attaches an OpenMath *prototype*, within a given *user-context*, to a MathML-presentation expression. One of the conclusions appearing here is that the user-contexts can be quite multiple. We explain the architectural choices that allow these user-contexts to be fine-grained but still perform high-speed rendering of the pages.

1.1 Outline

The paper first introduces related works, then proposes a broad classification of the notational diversity found around the world. The research litterature that speaks about the notational diversity is covered. The paper then presents the ingredients and principles of the notation census and indicates some typical examples. The exploitation within the ActiveMath environment is described. An outlook concludes the paper.

2 Related Works

One of the most notable and comprehensive census of mathematical notations was done by Florian Cajori in [Caj28] in 1928 who concentrated on the English language. Encyclopediae of mathematics could also claim to a very broad coverage but all of them are monolingual with the exception of the WikiPedia initiative, created by a great amount of contributors; not surprisingly, even the notations of the English WikiPedia is not fully consistant: one finds for example the binomial coefficient written one way or another depending on the page.

The adaptation needed so that a software can interact with its users dependending on the cultures of the world have been investigated by such works as [Mar09] often based on the influential work of Hofstede [Hof01]. However, such studies rarely went as detailed as to indicate the variability of mathematical formulæ.

Within the mathematical knowledge communities several papers quote the need to respect mathematical diversity, e.g. [MLUM06, KMM07, SW06, DL08] but the examples used there are too restricted to be representative. The confusion indicated above about the notation for binomial coefficient has crept into [CIMP01] and been cited by several such papers as example.

We have, in [MGLU09] presented some of the diversity we have met and some approaches to a solution. However this paper aimed at presenting the issues rather than aim at comprehensiveness. Therefore, we describe here the approach towards reaching the coverage needed for an informed usage of mathematical notations.

3 Notational Diversity Accross Cultures

Mathematics is widely viewed as a universal "methodology of reasoning" and a "language" common to all humans, truly "objective", and thus independent of subjective beliefs and cultural preformation. In particular, statements written in the mathematical symbol language, i.e. in terms of "formulæ", should thus be understandable in principle by everyone. In a general sense this is indeed true, and it is exactly this fact that makes mathematics a widely applicable tool for (almost) all sorts of problems: The mathematical language provides the possibility to express abstract concepts and logic reasoning in an unambigous way that can be understood independent of the reader's language and culture. If we say that a mathematical theorem (for example Fermat's last theorem) has been *proven*, we mean that a certain mathematical truth has been acertained at an objective level: It has been shown to hold *as such*, without reference to a particular language or culture.

However, when *practicing* mathematics (no matter whether the application of pre-established methods or genuine mathematical creativity are concerned, whether easy or difficult), language and culture dependent issues do play a role at several levels. Mathematical thought has evolved in history in different cultural contexts (beginning with several hundred years B.C.), each developing its own methods by which mathematical ideas can be expressed and operated with. Although the development of modern science and a world-wide scientific communication within the last few centuries lead to an extensive unification and "internationalization" of mathematical conventions, a lot of variations remained, so that we may talk about "mathematical cultures", which sometimes (but not always) are associated with "cultures" in the sense of countries or languages. In particular, differences exist even between regions as close to each other as the European countries.

To begin with, mathematical statements are usually embedded in phrases of "ordinary language", which may be viewed as "everyday language" used in a more or less rigorous way. For example, the statement

$$\text{there exists a natural number } n \text{ such that } n^2 = 4$$

contains the verbal elements "there exists", "natural number" and "such that". It is *in principle* possible to completely omit verbal elements of this type. For example, statement above could be re-expressed as $\exists n \in \mathbb{N} : n^2 = 4$.

However, in most cases a rigorous elimination of all "words between the symbols" would be tremendously impractical, in particular in texts addressing mathematics students (who are just about to *learn* mathematics) and non-specialists (interested to apply mathematical techniques for various purposes). As a consequence, mathematical texts in practice rely on written *words* at least as much as on written *mathematical symbols*, thus suffering from all sorts of variations in meaning and difficulties of translation.

One might expect now that at least the mathematical *symbols* provide something like a well-defined and culturally independent language (or at least vocabulary). However, mathematical concepts are not only used in the contemporary international mathematical research communication – which indeed uses a largely standardized symbolic language and may thus be considered as a "mathematical culture" in its own – but also in local contexts (e.g. technical or economic) or just in every-day life. All these fields of usage of mathematical concepts constitute "cultures" whose customs differ in many respects.

3.1 Differences in Decimal Numbers

The most basic example is how *numbers* are written. Whereas in many countries (as well as in the international communication) the number "twelve and a half" is written in decimal representation as 12.5 some mathematical cultures would use $12,5$ instead. In the first form *decimal separator* is indicated by a period or point (we thus say in English language "twelve point five"), whereas in the second a comma is used instead (consequently, in German language one calls this number "zwölf Komma fünf" and in the French language, one calls this number "douze virgule cinq"). Having evolved historically, notational differences of this type have even been standardized in local contexts. For example, the DIN (Deutsch Industrie-Norm – German industrial norm) specifies the comma as used in the second form as the "official" decimal separator. Students in German schools are thus acquainted with the notation $12,5$ rather than 12.5. Note however that the *meaning* of the period in the first and the comma in the second are precisely the same in both cases! In technical terms, these two differing notations constitute *semantically equivalent concepts.*

Consequently, a general rule such as the period means the same as the comma spelled out once and for all, could be admissible in a mathematical text that uses the period notation but is otherwise written in German. However, such rules are most often simplistic and fail because the other character (the comma or period) is commonly used as thousands separator. Thus, on the web, a german user may well encounter and be confused by 1.001 which could mean $1 + \frac{1}{1000}$ or $10^3 + 1$. So as to attempt clarification, the notation census thus contains a page about decimal numbers: `http://wiki.math-bridge.org/display/ntns/Decimal+Numbers`. A comprehensive set of number patterns is assembled at `http://unicode.org/repos/cldr-tmp/trunk/diff/by_type/number.pattern.html` albeit quite technical.

3.2 Difference Because of Names Differences

Mathematical concepts and operations are often *named* by words or phrases and these words are sometimes seen in the notations. For example the "sine" function

(German: “Sinus”, Spanish: “seno”). Just by their use, verbal expressions of this type have become part of the (scientific) language of the speaker or reader. Which word or phrase is used to denote a concept or an operation is to some extent a matter of (official or informal) standardization, but it might as well be a matter of *cultural tradition.* As the above examples show, the particular word or phrase used to denote *one* concept will be different when expressed in different languages. Most mathematicians in the world would use the symbol “sin” as an abbreviation of “sine” (“Sinus”), so that we may find a formula such as $\sin(\pi) = 0$ in a mathematical textbook. However, in Spanish language (or “culture”) of mathematics the abbreviation for this concept is “sen”. A textbook written in Spanish language will thus rather contain the formula $\mathrm{sen}(\pi) = 0$. Of course Spanish mathematicians are aware of this difference, and when addressing an international audience, they would use “sin” instead of “sen”, but it is a fact that Spanish pupils and students are familiar with “sen” rather than “sin”.

Similarly, in Arithmetics, even at an elementary level such as in secondary school, one knows the concept of the “greatest common divisor” of two integer numbers. In many languages the symbol used to denote this concept just consist of the initial letters of the corresponding verbal phrase. In English it is thus written as “gcd”. In the German tradition the same concept is called “größter gemeinsamer Teiler” and abbreviated as “ggT”. In Dutch, the corresponding phase “grootste gemene deler” is abbreviated as “ggd”. In French the corresponding phrase is generally called plus grand commun diviseur and is written“pgcd” though modernists tend to change it to pgdc because the formulation *commun diviseur* is archaic. See the notation census page about it: `http://wiki.math-bridge.org/display/ntns/gcd`

3.3 Differences to Avoid Confusion

Sometimes variations in the mathematical notation are dictated by the wish of clarity in the given written work: for example authors prefer to write the “binomial coefficients” in the French way in order to avoid possible confusion with the 2D-vectors: what is denoted by $\binom{5}{3}$ in most mathematical cultures tends to appear in French and Russian textbooks in the form C_5^3 instead (see the census page about the binomial coefficient: `http://wiki.math-bridge.org/display/ntns/binomial-coefficient`).

3.4 Same Notations, Different Concepts

We have stated before that in general *one* mathematical concept may be represented by different cultures in different ways. However, in some cases, a particular name or phrase – that may easily be translated between the languages – denotes *different* concepts. The most prominent example is the notion of “natural numbers”. Originally, it meant set of integer numbers greater or equal to 1, i.e. $1, 2, 3, 4, 5 \ldots$ and was denoted by the symbol $\mathbb{N}$. However, for practical reasons, it would be better to include the zero, hence to define the set of natural numbers by $0, 1, 2, 3, 4, 5 \ldots$.

Meanwhile, several mathematical cultures have followed this idea, while sticking to the original symbol $\mathbb{N}$. In order to denote the set of numbers without zero, one then usually writes something like $\mathbb{N}^+$ or $\mathbb{N}^*$. In the tradition in which the zero is not included, one may write $\mathbb{N}^0$ or $\mathbb{N}_0$ in order to denote the set with zero. As a consequence, when the symbol $\mathbb{N}$ (or the name "natural numbers") appears in a mathematical text, it could mean the integers with or without zero, depending on the mathematical culture it originates from.[1]

The problem is also pointed out in Eric Weissteins *MathWorld* pages: "Regrettably, there seems to be no general agreement about whether to include 0 in the set of natural numbers" [Weib]. *Wikipedia* also reflects this ambiguity in most pages on the subject of natural numbers.

It should be added that the web in general pulls the user out of a single textbook into easy jumps between many content sources, with varying degree of reliability. When *automated formula rendering* is concerned, ambiguities of this type must be taken into account in order to avoid confusion and let the user recognize easily which variant it is.

4 The Notation Census

Because mathematical notations are very diverse and their context of occurrence is just as diverse, we propose to establish a **census of mathematical notations**. That census should list all available mathematical notations that are widely spread around the world in a way that enables mathematics readers to see the mathematical notations used in the multiple cultural contexts even though they do not understand the language of the documents.

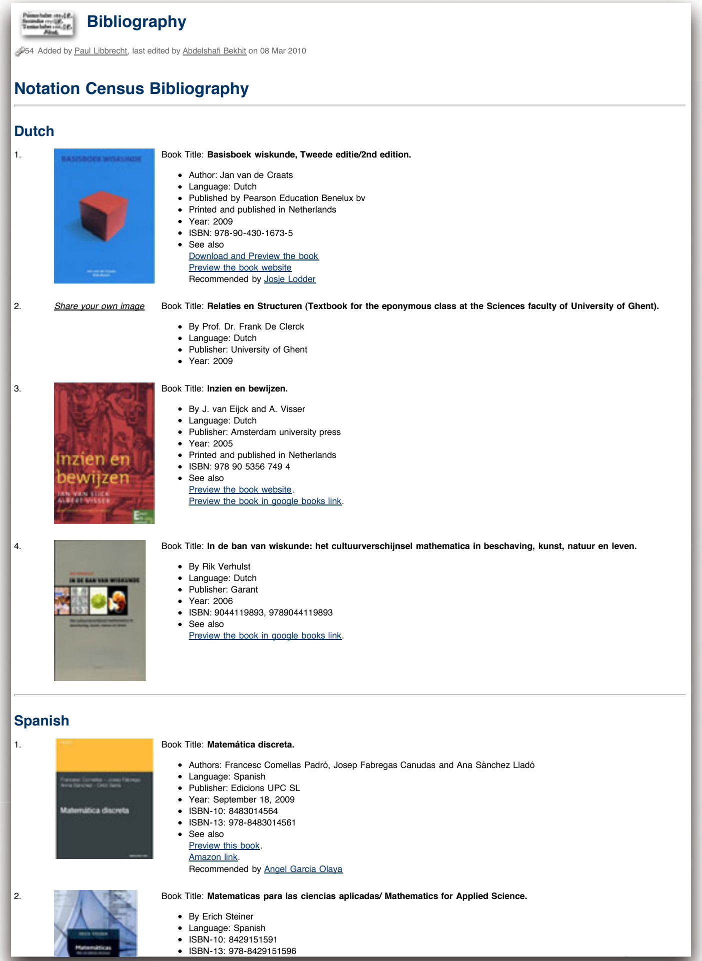
The census should be **visual** because mathematical notations are a graphical artifact and their rendering in web-browsers should succeed in all situations: this requirement prevents the usage of elaborate display technologies of mathematical formulæ so as to ensure the certainty of rendering the notation of that cultural context.

The census should be **traceable** so that one can recognize who has written each part and **commentable** so that its quality can be steadily improved; normal web-authoring practices similarly to those of Wikipedia or others apply here, together with the public visibility of any editing action.

The census should be displaying **widely used notations** by relying on extracts of mathematical texts that are widely used themselves; this is fundamental so as to achieve usefulness of the census.

Finally, the census should be **verifiable** because the misunderstanding among the cultures can be critically high: any interested reader should be able to find in just a few clicks who are the authors making a claim that a notation is widely used and in which cultural context it is used with the trustability of the source of notation serving as discussable reference point.

[1] In a *good* textbook this point is of course explicitly clarified.



Bibliography

54 Added by Paul Libbrecht, last edited by Abdelshafi Bekhit on 08 Mar 2010

Notation Census Bibliography

Dutch

1. Book Title: **Basisboek wiskunde, Tweede editie/2nd edition.**
   - Author: Jan van de Craats
   - Language: Dutch
   - Published by Pearson Education Benelux bv
   - Printed and published in Netherlands
   - Year: 2009
   - ISBN: 978-90-430-1673-5
   - See also
     Download and Preview the book
     Preview the book website
     Recommended by Josje Lodder

2. Share your own image Book Title: **Relaties en Structuren (Textbook for the eponymous class at the Sciences faculty of University of Ghent).**
   - By Prof. Dr. Frank De Clerck
   - Language: Dutch
   - Publisher: University of Ghent
   - Year: 2009

3. Book Title: **Inzien en bewijzen.**
   - By J. van Eijck and A. Visser
   - Language: Dutch
   - Publisher: Amsterdam university press
   - Year: 2005
   - Printed and published in Netherlands
   - ISBN: 978 90 5356 749 4
   - See also
     Preview the book website.
     Preview the book in google books link.

4. Book Title: **In de ban van wiskunde: het cultuurverschijnsel mathematica in beschaving, kunst, natuur en leven.**
   - By Rik Verhulst
   - Language: Dutch
   - Publisher: Garant
   - Year: 2006
   - ISBN: 9044119893, 9789044119893
   - See also
     Preview the book in google books link.

Spanish

1. Book Title: **Matemática discreta.**
   - Authors: Francesc Comellas Padró, Josep Fabregas Canudas and Ana Sànchez Lladó
   - Language: Spanish
   - Publisher: Edicions UPC SL
   - Year: September 18, 2009
   - ISBN-10: 8483014564
   - ISBN-13: 978-8483014561
   - See also
     Preview this book.
     Amazon link.
     Recommended by Angel Garcia Olaya

2. Book Title: **Matematicas para las ciencias aplicadas/ Mathematics for Applied Science.**
   - By Erich Steiner
   - Language: Spanish
   - ISBN-10: 8429151591
   - ISBN-13: 978-8429151596

Fig. 1. An extract of the bibliography

4.1 Ingredients

We realize the notation census in a **public wiki** which is made of the following core ingredients:

– **sources** are listed in a bibliography-like approach: on a single page containing all sources and a name of the cultural context they are used in; a link to a publisher page and link to possible web-download of the (partial) content. A source reference would be the entry point to judge the *relevance* of the notations with any given cultural context, something a person living in that context can do.

– notations are grouped per page, **one page per semantic**, each grouped in content-dictionaries following the semantic classification of, at least, the official OpenMath content dictionaries. Each semantic is linked to the OpenMath dictionary, the Wikipedia entry, and the entry in the Wolfram Encyclopedia [Weia] if possible.
– notations are listed there with a small informal description, links to the content-dictionary entries, and a series of **observations** made of a text containing the observed notation, the name of the symbol in that context, a pointer to the element of the bibliography containing it, and a **graphics copy** of the relevant bit indicated with a page number or other internal**reference** allowing a reader to find the used notation fast. This allows a predictable display, relying on graphical copies (scans or extracts of electronic versions of the book rendered in the browser as PNG or JPEG images). The name of the symbol in this culture is requested and, if possible, a character-based reproduction of the symbols' elements based on the Unicode character set.

The observations and sources both are given for a **cultural context** whose definition is unprecise. It is generally accepted to be at least made of a human language but it often goes beyond it including traditions, domain of study, and the mere practical conventions in communities of practice. For example the French notation for binomial coefficient, with the big C, is recognized to be widespread at school levels, as can be seen in `http://wiki.math-bridge.org/display/ntns/binomial-coefficient` but several combinatorics researchers agree that the vector notation is preferable even in French [Fra09]. A more widespread example is that of the square root of -1 which is written mostly with the letter i except in electrical engineering where the letter i is too close to that of electrical current hence the square root of unity is written j; the list of different observed notations is presented on `http://wiki.math-bridge.org/display/ntns/nums1_i`. For this reason, the census speaks about the *cultural context* which can, typically, be also defined by major texts.

Conformance is not, yet, strictly enforced for each contribution to the notation census: a check-list of ingredients before contributing an observation is provided in the *manifest*. We believe the requirements above, except maybe for a web-access to the text's content which is mentioned optional, are minimal.

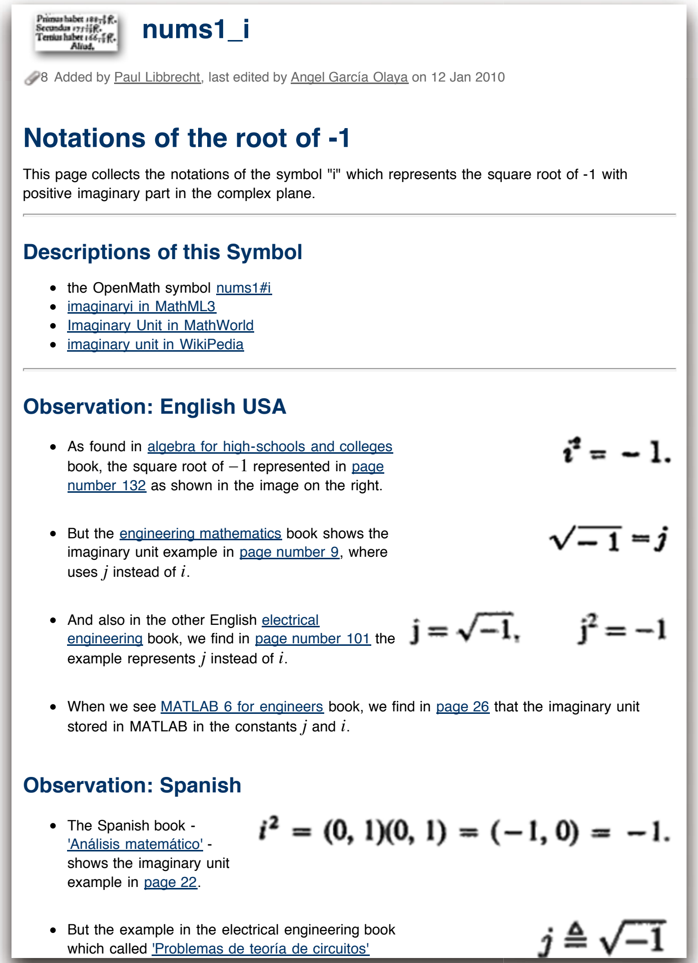
4.2 Evolution

The notation census can be reached at:

`http://wiki.math-bridge.org/display/ntns`

It has started in October 2009.

The explanations are currently provided in the *notation census manifest* at `http://wiki.math-bridge.org/display/ntns/Notation-Census-Manifest`. This text intends to be the one stop information source about the census guiding ideas and principles.



nums1_i

8 Added by Paul Libbrecht, last edited by Angel García Olaya on 12 Jan 2010

Notations of the root of -1

This page collects the notations of the symbol "i" which represents the square root of -1 with positive imaginary part in the complex plane.

Descriptions of this Symbol

- the OpenMath symbol nums1#i
- imaginaryi in MathML3
- Imaginary Unit in MathWorld
- imaginary unit in WikiPedia

Observation: English USA

- As found in algebra for high-schools and colleges book, the square root of -1 represented in page number 132 as shown in the image on the right. $i^2 = -1.$
- But the engineering mathematics book shows the imaginary unit example in page number 9, where uses j instead of i. $\sqrt{-1} = j$
- And also in the other English electrical engineering book, we find in page number 101 the example represents j instead of i. $j = \sqrt{-1}, \quad j^2 = -1$
- When we see MATLAB 6 for engineers book, we find in page 26 that the imaginary unit stored in MATLAB in the constants j and i.

Observation: Spanish

- The Spanish book - 'Análisis matemático' - shows the imaginary unit example in page 22. $i^2 = (0, 1)(0, 1) = (-1, 0) = -1.$
- But the example in the electrical engineering book which called 'Problemas de teoría de circuitos' $j \triangleq \sqrt{-1}$

Fig. 2. A page of observation: about the complex i

We expect to populate the notation census gradually with the following contributions: first the MathML CD-group, then all ActiveMath available content, then the full Math-Bridge content.

The first objective of the notation census is to serve the math-bridge project which we describe below: the wiki is a good communication means between practicing mathematics educators and the technical team *encoding* the notations.

Beyond this project, it should support the ActiveMath project's development which is ongoing since more than 10 years. It should also support other initiatives of collecting notations for the rendering of semantic-mathematical-terms such as [KMM07].

Finally, thanks to the connections to the sources, this census should also be one of the reliable places of information for a curious user wishing to be informed of the various ways to note mathematical terms in different cultures.

For each of these usages completeness is desirable but is not a requirement: the more coverage the better, but even a partial coverage can find its usages.

5 Notations for ActiveMath

In this section, we turn to one of the exploitations: how the notation census will be exploited for the purposes of the Math-Bridge project and how the ActiveMath platform can be endowed to honour the cultural contexts by presenting mathematical formulæ in a customizable way.

Math-Bridge intends to offer bridging-courses allowing learners to bridge mathematics knowledge gaps, taking them with their pre-existing knowledge, hence by the notations they already know, to a level ready to start University. Hence it aims to use the right notations in the cultural contexts the learner probably has learned in, that is late-school-level mathematics for Dutch, English, French, Finnish, German, Hungarian, and Spanish classes.

Math-Bridge intends to use the rich formulæ rendering of the ActiveMath platform, coupled with other personalization features of this web-platform. This platform renders mathematical formuæ from their OpenMath semantic source enabling a rich set of services attached to the rendering. These include the transfer to computer algebra systems, exercise inputs, plotter or the searchability...

ActiveMath presents mathematical formulæ using a few widely supported display technologies on the web: HTML completed with CSS, XHTML with formulæ in MathML, or PDF created by `pdflatex`. The transformation from the OpenMath encoding to the rendering formats is done through several stages that are explained in [ULWM04]. Two aspects are worth mentioning here:

Authorable Notations. ActiveMath rendering to HTML, XHTML, or TeX is done through XSLT and Velocity; most of the mathematical formulæ conversion is done through `symbolpresentation` elements which collect `notation` elements each of which is a pair of an OpenMath expression and its associated rendering in MathML, see [MLUM06]. Each `notation` element is annotated with a context of use: a notation is used when the prototype matches as well as the context.

Wealth of Configurable Contexts. As we have described above, the *cultures* in which mathematical notations are defined, are somewhat fuzzily defined. Therefore ActiveMath allows the notations' cultures to be written in the following *contexts*:

– most importantly, a notation can be associated to a **language**. At XSLT time, the notation of the right language is chosen.
– a notation can be associated to an **output format** (this is most important to care for imperfections of individual formats. At XSLT time, the notation for the right format is chosen.

- a notation can be associated to an **educational level**, indeed, we have often been requested that students at University see some operators one way while others in another way. This is realized by the notation generating the velocity code which is evaluated when this user-information is ready, at each delivery.
- finally, a notation can be associated to a **collection,** that is, to the collection of the book the item is in. This allows an author re-using a collection of someone else within his book, to ensure that formulæ in this book are used consistently, for his items or the one of others. This also evaluated at each delivery.

Within the Math-Bridge project, where a broad sharing of learning items is expected, these contexts are of fundamental importance to allow consistent mathematical notations within the context of each learning experience. Indeed, the most common review note we have received is about the notations whereas these notations were fully checked for the LeActiveMath project by math educators that use it in their learning in German, Spanish, and English.

Based on the notation census, the content enrichment and translation processes will *encode* the necessary `symbolpresentation` and `notation` elements. This will be done as part of the encoding process and will be integrated in the quality proofing process.

6 Conclusion

This paper has sketched the issues encountered to meet proper knowledge about mathematical notations in the multiple cultures and the great diversity of notations that are widely used in, at least, the secondary education mathematics texts. To address this lack of knowledge, we have proposed a notation census, in the form of a collaborative wiki-based effort collecting referenced and hyperlinked observations of the usage of widely used notations.

The census population has started in October 2009 and now contains about 130 observations. The duration to write a page collecting observations in the available sources is about 30 minutes with variations we have observed due to: the digital availability of the observations, the understanding by the encoder of the mathematical concept behind it, the searchability of the sources (a quality often missing in PDF books in Arabic we have been handling), the availability of glyphs to search for (e.g. search for *gcd* is a lot easier than search for a matrix-transpose).

The wiki technology used here is quite basic and generic. Together with the limited requirement for structure, the potential for public contributions, without large technical competencies, is large. This lack, however, means a limited set of services: for example, it is not possible, yet, to list all the observations coming from one source or in a given language. Some of these can be recovered by an enriched tool set we intend to work on (sections in a wiki page are quite commonly automatically detected). Similarly the searchability is quite limited: the name of the symbol is the best key, provided it is known, but no *graphical search* can be done (except browsing through all); we believe the latter is a research challenge.

We intend to build on the census to help into creating links to web pages that speak about the mathematical symbols around the OpenMath content-dictionaries' web-pages in http://www.openmath.org/. The visual nature of the census is probably one of the efficient hooks for a mathematician's eyes which can enable him to identify that a given semantic is matching what he expects.

Finally, the collaborative nature of the census web system opens the long-term possibility of contributions of a broad body of contributors, similarly to that obtained by the Unicode consortium's Common Locale Data Repository as can be seen at http://cldr.unicode.org/index/charts. Indeed, shortly following the public announce of this repository, in December 2009, spontaneous contributors appeared.

Acknowledgements

The author wishes to thank Franz Embacher for fruitful input on how to present this work and Abdelshafi Bekhit for his numerous contributions to the notation census.

This work is partially funded by the European Commission under the eContentPlus programme in the Math-Bridge project (ECP-2008-EDU-428046).

References

[Caj28] Cajori, F.: A History of Mathematical Notations. 2 vols. Open Court (1928), Browsable from http://www.archive.org/stream/historyofmathema031756mbp

[CIMP01] Carlisle, D., Ion, P., Miner, R., Poppelier, N.: Mathematical markup language, version 2.0 (2001), http://www.w3.org/TR/MathML2/

[DL08] Davenport, J.H., Libbrecht, P.: The freedom to extend openmath and its utility. Journal of Computer Science and Mathematics 59, 1–19 (2008)

[Fra09] Bergeron, F.: Re: Fascicule de combinatoire, Private email communication on August 31st at 16:09 GMT+2 (2009)

[Hof01] Hofstede, G.: Culture's consequences. Comparing values, behaviors, institutions, and organizations across nations. Sage Publications, Thousand Oaks (2001)

[KMM07] Kohlhase, M., Müller, C., Müller, N.: Documents with flexible notation contexts as interfaces to mathematical knowledge. In: Proceedings of MathUI 2007, Linz (2007), http://www.activemath.org/workshops/MathUI/07/proceedings/Kolhase-et-al-DocumentNotations.html

[Koh06] Kohlhase, M.: OMDoc – An Open Markup Format for Mathematical Documents (version 1.2). LNCS (LNAI), vol. 4180. Springer, Heidelberg (2006), http://www.mathweb.org/omdoc

[LAG09] Libbrecht, P., Andrès, É., Gu, Y.: Smart Pasting for ActiveMath Authoring. In: Proceedings of MathUI 2009 (July 2009), http://www.activemath.org/workshops/MathUI/09/

[LCME$^+$08] Laborde, C., Creus-Mir, A., Egido, S., Dietrich, M., Libbrecht, P.: Curricula categorisation into ontology. Deliverable D2.5, The Intergeo consortium (September 2008)

[Mar09] Marcus, A.: Global/intercultural user interface design. In: Sears, A., Jacko, J. (eds.) Human-Computer Interaction: Design Issues, Solutions, and Applications, ch. 18, pp. 355–381. CRC Press, Inc., Boca Raton (2009)

[MGLU09] Melis, E., Goguadze, G., Libbrecht, P., Ullrich, C.: Culturally adapted mathematics education with activemath. Artificial Intelligence and Society, Special Issue on Enculturating HCI 24, 251–265 (2009)

[MLUM06] Manzoor, S., Libbrecht, P., Ullrich, C., Melis, E.: Authoring presentation for OpenMath. In: Kohlhase, M. (ed.) MKM 2005. LNCS (LNAI), vol. 3863, pp. 33–48. Springer, Heidelberg (2006)

[SW06] Smirnova, E., Watt, S.: Generating tex from mathematical content with respect to notational settings. In: Proc. International Conference on Digital Typography and Electronic Publishing: Localization and Internationalization (TUG 2006), Marrakech, Morocco, November, pp. 96–105 (2006), http://www.tug.org/TUGboat/Contents/contents27-2.html

[ULWM04] Ullrich, C., Libbrecht, P., Winterstein, S., Mühlenbrock, M.: A flexible and efficient presentation-architecture for adaptive hypermedia: Description and technical evaluation. In: Kinshuk, C., Looi, E., Sutinen, D., Sampson, I., Aedo, L.U., Kähkönen, E. (eds.) Proceedings of the 4th IEEE International Conference on Advanced Learning Technologies (ICALT 2004), Joensuu, Finland, pp. 21–25 (2004)

[Weia] Weisstein, E.: Mathworld, the web's most extensive mathematics resource, http://mathworld.wolfram.com/

[Weib] Weisstein, E.W.: Natural number, http://mathworld.wolfram.com/NaturalNumber.html

Evidence Algorithm and System for Automated Deduction: A Retrospective View

(In Honor of 40 Years of the EA Announcement)

Alexander Lyaletski[1] and Konstantin Verchinine[2]

[1] Faculty of Cybernetics, Kiev National Taras Shevchenko University
2, Glushkov avenue, 03680 Kiev, Ukraine
lav@unicyb.kiev.ua

[2] Math-Info Department, Paris 12 University
61, avenue du General De Gaulle, 94010 Creteil, France
verko@logique.jussieu.fr

Abstract. A research project aimed at the development of an automated theorem proving system was started in Kiev (Ukraine) in early 1960s. The mastermind of the project, Academician V.Glushkov, baptized it “Evidence Algorithm”, EA[1]. The work on the project lasted, off and on, more than 40 years. In the framework of the project, the Russian and English versions of the System for Automated Deduction, SAD, were constructed. They may be already seen as powerful theorem-proving assistants. The paper gives a retrospective view to the whole history of the development of the EA and SAD. Theoretical and practical results obtained on the long way are systematized. No comparison with similar projects is made.

1 Introduction

The research project entitled “Evidence Algorithm” was initiated by V.Glushkov in the early 60-s in Kiev. At that time, some fundamental facts concerning formal proof search and opportunities (potential in most cases) to use computers to find a proof, were already known. The domain that was called “automated theorem proving” (ATP or “machine reasoning” in the AI community) became a challenging one for logicians as well as for computer scientists (see e.g. [92] for short history). There were hopes! Recall the title of an early Hao Wang’s paper: “Towards mechanical mathematics” [104]!

V.Glushkov, as he personally told us, was motivated by two main reasons:

(1) To get an aid while verifying long and routine algebraic transformation (as a working mathematician he obtained valuable results concerning Hilbert’s 5th problem).

(2) To try the strength of the existent computers pushing them to run on the limits of their abilities.

V.Glushkov formulated the main question in a slightly unusual way.

Let us consider some relatively well formalized mathematical theory, e.g. Lie algebras. There are a small number of basic facts (axioms) which are considered to be

[1] Below, we explain why this title was chosen; it was used first in [9].

evident even for beginners. Let's apply simple purely logical tools to obtain several consequences. They are also evident. Then one can apply the same logical tools to the conclusions and so on. Are the results still evident? If the conclusions were obtained by a programmed inference engine, the answer is "yes, they are". From the viewpoint of this engine. But probably not from the human point of view. Thus, provided the above mentioned engine, we would be able to prove/verify something that is not evident for humans. Further to that, this "evidence maintaining engine" may be reinforced with heuristics, proof methods, lemma application, definition expansion, and so on. In this way, we could enlarge the notion of "being evident" to the extent that might include nontrivial facts/theorems. Well, "now do it, guys!"

That is why the algorithmic part of the project (and afterwards the project as the whole) has got the name "Evidence Algorithm", EA, or $\exists\forall$ for fun.

It was also already clear at that time that nobody would like to formalize the mathematical knowledge/reasoning in the usual first order language. Hence a formal but human-friendly language had to be developed to provide a possibility for the construction of a mathematical assistant system convenient for wide range of scientists.

So, three major components of such a system should be:

(1) a powerful input language that must be close to the natural mathematical language and easy to use;

(2) an inference engine that implements the basic level of evidence (sometimes, we call it a "prover" below);

(3) an extensible collection of tools that reinforce the basic engine (sometimes, we call it a "reasoner" below).

In what follows, we give a short description in chronological order of what has been done in each of the above-mentioned directions.

Please note that our main goal is to trace the long path of the project development and to recall the results obtained. That is why the reference list is so long. For the same reason, we could neither make any comparison with similar existing systems, nor give an illustrative set of examples. Sorry for that. We frequently got an impression that the automated reasoning community is not sufficiently acquainted with EA project (for instance, SAD was not mentioned in F.Wiedijk's book [106]) though we think that some ideas and results might be useful to know. We hope that the text given below will partially meet the lack of such information.

Note on the bibliography. Almost all papers published before 1992 were written in Russian and therefore are hardly available now. We translated the titles and put them onto the list just to indicate what was done in the old time. All the papers are listed in chronological order.

The rest of the paper contains three parts according to the three periods in the history of the EA project. They are as follows.

The first one: 1962 - 1970. We call it "Pre-EA Stage" below.

The second one: 1970 - 1992. It is called "EA and Russian SAD" below.

The third one: 1998 - nowadays. Below it is called "Post-EA Stage and English SAD".

Several final remarks conclude the paper.

2 Pre-EA Stage (1962–1970)

Few people remember now that the Soviet computer history began in Kiev. The first von Neuman computer was assembled and tested at the turn of 1950 in a small laboratory headed by the academician S.Lebedev. In 1955 S.Lebedev left for Moscow and the director of Kiev Institute for Mathematics, prominent mathematician B.Gnedenko invited V.Glushkov to take the supervision of the laboratory (which was transformed into the Institute of Cybernetics 5 years later).

On the other hand, at that time there was a powerful logic, linguistic and algebraic team at the mathematical department of the Kiev University. Professor L.Kaluzhnin who was the head and the heart of the team, invited V.Glushkov to join their efforts.

So the Kiev school in the ATP domain appeared really at the borderline of computer science and mathematical logic.

In 1962, V.Glushkov published a paper [1] where he analyzed several rather simple proofs in Group Theory and suggested that the proofs might be built automatically with the help of a not too complex procedure. The idea attracted three people who began their research on the subject: A.Letichevsky, one of the first Glushkov's disciples, (in 1962), F.Anufriev (in 1962) and V.Fedyurko (in 1963). A bit later, V.Kostyrko and Z.Aselderov had joined the team. The first-time approach to the problem was purely empirical – they analyzed a lot of proofs taken from textbooks, monographs and articles for trying to formalize all them and to find (almost by feeling) methods, heuristics and representation details that might help to construct a proof of a theorem under consideration automatically. As a result an algorithm of proof search in Group Theory was constructed and even implemented (the corresponding program run on the monstrous Ural-1 computer). The first communication about it was done at the First All-Union Symposium on the Machine Methods of Logical Inference Search, that took place in Lithuania in 1964 [97] (see also [93]). Later a paper on the subject was published [4] (and translated afterwards into English).

The algorithm, though being comparatively simple, contained nevertheless:

- a method of inference search for some class of first-order formulae;
- a reduction technique for simplifying search space;
- a collection of heuristics (e.g. the inclusion relation was exploited);
- special methods of equation solving.

So we can say that it was the first problem-oriented prover for Group Theory. Here is an example of proved theorem: "The centralizer Z of any subgroup P is a normal subgroup of the normalizer N of P".

The above-mentioned proof-search method resembled, in some sense, well known backward chaining, but some features were added to make it applicable to non-Horn formulae. Later on, the method was generalized by F.Anufriev and extended to the whole first-order classical logic without equality [5,8]. It can be interpreted as a goal-oriented sequent calculus not requiring skolemization and using an analog of Kanger's notion of substitution admissibility. Later, the method was transformed into a correct and complete sequent calculus [22] with skolemization. It had got the name "Auxiliary-Goals Search calculus" ("AGS calculus" below) and served as a prototype for various sequent-type inference engines of the EA project.

Equation solving became the subject of Z.Aselderov PhD thesis [7], which was successfully defended in 1968.

Now let's cast a glance at the list of required components of the conceived EA system. No convenient input language was yet proposed at the time being. On the other hand, it was difficult to continue the project without it. To see why, try to convert the theorem above to the first-order language. On this subject, there were only two Kaluzhnin's papers: [2] and [3]. Some time later, V.Kostyrko made an attempt to solve the problem and after some period, a paper was published [10] where a contour of such a language was outlined. The main idea was as follows.

Let's consider an atomic first-order formula. It is always of the form $R(t_1,t_2,\ldots,t_n)$ where R is an n-ary relation symbol, whereas a "natural" atomic statement is of the form $< subject\ group > < predicate\ group >$. Well, one can:

- select an argument among $t_1,t_2,\ldots,t_n$, say t_1,
- consider it as the subject,
- "reduce" R to something $(n-1)$-ary $N(t_2,\ldots,t_n)$,
- add a new connector to "attach" t_1 to $N(t_2,\ldots,t_n)$ (ε was chosen in the original version).

Now $R(t_1,t_2,\ldots,t_n)$ can be written as $t_1 \varepsilon N(t_2,\ldots,t_n)$ and read as t_1 *is a* $N(t_2,\ldots,t_n)$. For example, $Subgroup(H,G)$ gives H ε $Subgroup\_of\ G$ and so on. Was it not more than syntactic "sugar" or one could gain something interesting with it? Below we demonstrate what was made in this direction later.

For the completeness of the description of that time, it may be needed to remind the last implementation of the propositional part of Anufriev's procedure, which was made by A. Malashonok on the BESM-2 computer at the beginning of 1970 [13].

3 EA and Russian SAD (1971 - 1992)

In 1970, V.Glushkov published one more paper on the subject [9]. At that time, he associated the progress in the domain of ATP with the general tendency to make computers more intellectual (see also [14]). As to the project in question (except for the fact that it has got its final name "Evidence Algorithm"), V.Glushkov emphasized in the paper the importance of a convenient but formal language for mathematical texts. We also have to note that for the first time the term "automated theorem proving" was used instead of "automatic theorem proving" and the problem of how to build something like an "interactive environment" was explicitly formulated. In fact, a proof assistant was conceived.

It seems that somewhen in the middle of 1970 V.Glushkov decided to add "young forces" to the existing EA team, and he charged one of his former pupil, V.Bodnarchuk to gather them. At that time, V.Bodnarchuk was the head of a computer department; its members had just finished their work on a specialized mini-computer for engineering computation with the language "Analitik" [11].

The input language "Analitik", being convenient for engineers and having its hardware implementation, was one of the distinguishing features of the computer, and V.Bodnarchuk was its main creator. Besides, V.Bodnarchuk was congenial soul for L.Kaluzhnin. All these exerted great influence on the development of the EA.

At the same time, four young people became the postgraduate students at the Institute of Cybernetics, both authors were among them.

Two of them, A.Degtyarev and K.Verchinin, were graduated from the Mechanical and Mathematical Faculty of the Moscow State University. Other two, A.Lyaletski and N.Malevanyi – from the Cybernetics Faculty of the Kiev State University. So, we had joined the EA team and V.Bodnarchuk became our "local supervisor" (the global one was V.Glushkov). We were young, full of energy and illusions...

At the very beginning, V.Bodnarchuk has formulated the following tasks:

- careful revision of everything that was done previously by the "old team";
- detailed analysis of mathematical texts in various domains;
- preparation of two surveys: (1) of combinatorial proof-search methods (published, see [16]) and (2) of using heuristics in proof search (published, see [18])

The revision of the existing version of the AGS method demonstrated that, first, the use of the Kanger's notion of substitution admissibility instead of skolemization complicates drastically an eventual implementation and, second, the method requires a special technique for equality handling. So, to advance the whole project, one needed:

- either to improve the AGS method paying special attention to redundancy avoidance and equality handling, or to adapt one of existing combinatorial methods of proof search for the role of inference engine in the EA project;
- to develop a practically usable version of the "mathematical language" along with the whole syntactical service around it;
- to find a convenient formalization of what is frequently used in mathematical texts to make them available for human reader – "proof method", "proof scheme", "lemma application", "definition dependency", etc.
- to find methods of what is called "knowledge management" now, e.g. to try to understand what the "relevancy relation" on mathematical facts might be;
- to develop an implementation base (it became clear at the very beginning that experimental work was strongly needed and it could not be done in paper-and-pencil mode).

We began in quite favorable setting. Two circumstances should be especially noted. At that time, it was easy to establish scientific contacts in the ex-USSR and we have done that: with famous Leningrad logic school, with excellent Novosibirsk logic school (founded by A.Maltsev), with strong Moscow logic school, with linguists, psychologists, etc. The second point is that last-year students of the new Cybernetics department of the Kiev University used to pass their six month professional training at the Institute of Cybernetics. In this way the second EA team had got two very capable young researchers: A.Zhezherun (in 1973) and M.Morokhovets (in 1978).

3.1 Theoretical Work

Here is a brief description of research interests and results obtained by members of the second EA team.

At the beginning, A.Degtyarev studied the role of heuristics in formal proofs. He restricted himself with linear algebra and showed that for large class of theorems, the proof search (by resolution with paramodulation) may be controlled in a way and reduced to the problem of finding solution to a set of linear equations [17,21]. It was quite interesting result but A.Degtyarev did not continue that direction and devoted himself to the problem of equality handling in resolution-like methods. As a "side effect" he obtained an efficient unification algorithm (published later in [23,42]) that was based on the same principles that the well-known Martelli and Montanary algorithm [95] formulated later.

His main results concern various paramodulation strategies and the problem of compatibility the paramodulation rule with term orderings. The most known is so called monotonic paramodulation [30,31,50] subsequently used in many other researchs on the subject.

A.Lyaletski occupied himself with the careful analysis of combinatorial proof search methods trying to put them in a common setting and find (or build) the best candidate for a resolution-type inference engine. He suggested a modification of the resolution rule which operated with more general objects than clauses – conjunctive clauses or c-clauses. (Later, V.Lifschitz [94] independently proposed something similar and called them "super-clauses"). Two different c-clause calculi were build [24,25] which permitted to reformulate well-known Maslov's Inverse Method [96] in a resolution-like manner.

Another problem was the skolemization. Is it bad or not? Anufriev's method did not use skolemization, but it adds new entities as in the case of Kanger's method. On the other hand, skolemization simplifies the algorithmic part of proof search methods. A.Lyaletski found an original notion of admissible substitution that allowed him to get in some sense a compromise. He built a series of sequent calculi with resolution style inference rules, that, on one hand, don't require skolemization and, on the other hand, are not less efficient than the usual resolution calculus ([34,35,53]).

K.Verchinine was strongly involved in the language problem. We had to formalize mathematical texts, not only isolated statements. A text may be considered as a structured collection of sections: chapters, paragraphs, definitions, theorems, proofs, etc. So a part of the language was designed to represent this structure, its "semantics" was given by the "trip rules". Another part served to formalize a statement. New units were added to the standard first-order syntax which permitted to use nouns, adjectives, special quantifiers, etc. The language was developed and has got the name TL – Theory Language [15,20]. Here is a formal TL phrase: "there is no remedy against all deseases but there is a desease against all remedies". (That time the vocabulary as well as the syntax was certainly Russian.)

Two kind of semantics were defined for that part: a transformational one (an algorithm to convert a TL statement into its first-order image) and another one – in the traditional set-theoretical style where ε was interpreted as the membership relation [19]. The last semantics permitted to define the "extension" of every notion (e.g. the extension of "subgroup_of G" is the class of all subgroups of G) and to introduce a structure on the set of notions which restrict quantifiers in the given sentence. That structure was called "situation" and was used in attempts to formalize a relevancy relation.

At the beginning, A.Zhezherun took active part in the TL language development. He designed and implemented the whole syntactic service for the linguistic part of the future system. As usual, there were funny side effects of the work. For instance, computer linguists have always searched for some invariant (called profound semantic structure) that could be used in machine translation algorithms. A.Zhezherun and K.Verchinine showed that the first-order image of a TL statement can play the role of such invariant. So just changing the superficial decorations in some regular way, one can translate mathematical statements from Russian into English and vice versa (provided the dictionary). A.Zhezherun wrote a program to play with, and it worked surprisingly well! Besides, he studied the opportunity to formalize mathematical reasoning in a higher-order logic and proved in particular the decidability of the second-order monadic unification [39].

M.Morokhovets occupied herself with the problem of "reasoner" (see above). As the reasoner must have a prover to cooperate with, the last was badly needed. The AGS based prover didn't fit well to that purpose, so we decided to develop and implement a resolution-and-paramodulation based prover with a flexible architecture that could be adapted to various strategies and auxiliary inference rules. M.Morokhovets has done it. The first observation showed that some particular premises are strongly responsible for the search space explosion. The transitivity axiom clearly is among them. M.Morokhovets proved that for some large class of transitive relations, this axiom may be eliminated and replaced by a special inference rule which can be controlled to shorten the search space [57].

Another idea was to use the fact that all quantifiers in the TL statement are restricted (bounded). Is it possible to "forget" the restrictions, find an inference and then just verify that all substitutions are correct w.r.t. these restrictions (bounds). M.Morokhovets has found several classes of statements for which the answer is "yes", and has implemented corresponding procedure [56]. One more question was as follows. Let's suppose that a conjecture is proved and the resolution-style inference is constructed. How to present it in a human readable form? The set of conversion rules that permit to do it (based on an early result of K.Verchinine), was designed and implemented by M.Morokhovets, too.

3.2 Experimental Work

Certainly, some computer experiments have been done from the very beginning of EA project development (it was one of Glushkov's ideas – to be permanently accompanied with computers while doing theoretical research). Still in 1971 K.Verchinine used the syntactic tools taken from another system (developed in the same department) to implement a part of TL grammar. A.Malashonok have programmed AGS prover to make local experiments with. Also local experiments with paramodulation strategies were maid by A.Degtyarev. N.Malevanyi began to prepare something like a specialized library for future experiments on the BESM-6 machine – another Lebedev's creation – one of the most powerful computer in the ex-USSR.

Systematic programming was initiated after A.Zhezherun appeared. He became the main designer and programmer of the system for mathematical text processing. But no doubt, we all were involved in programming. At that time, the IBM System 360/370 (cached under the name "ES Line Computer") was admitted in the ex-USSR as the main platform. With the native operating system and the PL/1 as the main programming language – what a hell!!!

Nevertheless the work advanced and the first experiments with the whole system were done in 1976/1977. The main task was formulated as mathematical text verification and may be presented as follows.

Let a TL text be given. The system can:

- parse the text informing the user about syntactic errors (if any);
- convert the text to some tree-like internal form;
- run the main loop: choose a goal sentence to verify and find its logical predecessors;
- construct an initial proof environment for one of available provers[2];
- start the prover and wait;
- if the prover fails then ask to help;
- if the prover succeeds then output the proof, choose the next goal and repeat the main loop until the end of the text be reached.

The first public presentation of the system in question was made at the All-Union symposium "Artificial intelligence and automated research in Mathematics" (Kiev, Ukraine, 28-30 November 1978). It worked!

In 1980, V.M. Glushkov gave the name "System for Automated Deduction" (SAD) to the implemented system and it has this name now.

The further work consisted in improving the system and adding new features to it. We extended the mathematical texts library and developed a conception of further extension of TL language with "imperative" (algorithmic) constructions. A method of using auxiliary statements in proof search (based on the notion of situation) was implemented by V.Atayan [47]. Efficient paramodulation strategies were added and tested by A.Degtyarev. A resolution-based prover was implemented by M.Morokhovets.

In the meantime four PhD thesis were defended at the Institut for Cybernetics: A.Zhezherun has got his PhD in 1980 [46], A.Lyaletski [53], A.Degtyarev [51] and K.Verchinine [54] – in 1982. M.Morokhovets' thesis was in preparation.

We understood that to advance the project we need to try the SAD system in some more or less practical applications. One possible application was the automated program synthesis and we established a contact with professor Enn Tyugu (Tallinn, Estonia) and his team. Another interesting application was the deduction tool for expert systems. The problem is that classical logic is rarely used in this domain. So, the question appeared: is it possible to adapt SAD for the inference search problem in non-classical logics?[3].

But everithing comes to its end. Sooner or later.

3.3 Team Evolution (or the Sad Part of the SAD History)

Already in the end of 1972, V.Bodnarchuk falled seriously ill and, actually, he abandoned the research activity for a long time. From 1973 to 1975 F. Anufriev,

[2] At that time, the SAD system prover was constructed and implemented on the base of an original sequent-type calculus [48]. It had the following features: it was goal-oriented, skolemization was not obligatory, and equality handling was separated from deduction. Now, the native prover of the current (English) SAD possesses the same features.

[3] Later, a theoretical answer on this question was obtained in a number of papers of A.Lyaletski (see, for example, [77,86]); from this point of view, some researches on Herbrand theorems ([78,79,82,90]) also may seem to be interesting.

Z. Aselderov, V. Kostyrko, and A. Malashonok left the team because of various reasons, they never came back to the subject area afterwards. In 1982, V.Glushkov was dead. The administration style in the Institut for Cybernetics changed and we were not the favorit director's team any more. In the middle of 1983, A. Lyaletski and A. Zhezherun left for the Kiev University. In 1984, K. Verchinine moved to another department and changed his research area. Finally, in 1987, A. Degtyarev left for the Kiev University, too. M.Morochovets stayed at our former department of the Institute of Cybernetics. The EA team did no more exist...

4 Post-EA Stage and English SAD (1998-Nowadays)

In 1998, the Evidence Algorithm project moved into a new stage. That year the INTAS project 96-0760 "Rewriting techniques and efficient theorem proving" started and brought financial support for resumption of work on SAD. The new working group included Alexander Lyaletski at Kiev National University (KNU), Marina Morokhovets at the Institute of Cybernetics in Kiev, Konstantin Verchinine at Paris 12 University in France, and Andrei Paskevich, fourth-year undergraduate student of KNU.

The work started in 1999, with re-implementation of the TL language on IBM PC. The programs were written in C on the Linux platform. In a year, towards March 2000, parsing and translation of TL sentences into a first-order language was implemented. The English-based version of TL had been given the name ForTheL, an acronym for "FORmal THEory Language" (also a Russian word meaning "trick" or "stunt"). The language was presented firstly at the Fifth International Conference "Information Theories and Applications" in September 2000 in Varna, Bulgaria [67].

The same summer the work started on re-implementation of the deductive tools of SAD. By January 2001, A.Paskevich created the first prototype of the prover (the prover had gotten the name "Moses"). A bit later the technique of admissible substitutions by A. Lyaletski which permitted to dispense with skolemization and preserve the initial signature of a proof task, was also implemented. Later, the equality elimination procedure by Brand [91] was added to handle the problems with equality. By June 2001, the complete "workflow" of the initial SAD: from ForTheL text to first-order representation to proof task to proof tree, was reestablished. Of course, a lot of functionality of the previous implementation has not been transferred into the new system.

In September 2001, A. Paskevich started his doctoral study under the joint supervision of Konstantin Verchinine and Alexander Lyaletski. His work aimed at the development of a new, two-level architecture of a mathematical assistant.

In the first prototype of the English SAD system, the reasoner was virtually nonexistent. The theoretical development of the reasoner started with the work on "local validity", which allowed to perform sound logical inferences inside a formula, possibly under quantifiers. This technique could provide a basis for in-place transformations (such as definition expansions) as well as for handling of partial functions and predicates [71].

By the end of 2003, tools for supporting proofs by case analysis and by general induction (with respect to some well-found ordering) were implemented in the SAD. In 2004, an experimental support for binding constructions, such as summation and limit, was also added [81].

An algorithm for generation of atomic local lemmas was constructed and implemented: these lemmas help to prove a lot of simple statements without using a prover at all.

An interesting feature of the SAD is that the prover does not depend on the rest of the system. It means that various provers can be used as the system inference engine (provided the interface be written). The following ones were used in our experiments: SPASS [105], Otter [98], E Prover [103], Vampire [100] and Prover9 [99].

In July, 2007, the "enriched" SAD system was presented at the 21st Conference on Automated Deduction in Bremen, Germany [85]. A. Paskevich has made several improvements since then. The current version of the system is freely available at http://www.nevidal.org . Here is a short list of texts (proofs) that were successfully verified by the SAD: Tarski's Fixed Point theorem, Newman's lemma, Chinese Remainder theorem, Infinite Ramsey theorem, "The square root of a prime number is not rational", Cauchy-Bouniakowsky-Schwartz inequality for real vectors, Fuerstenberg's proof of the infinitude of primes.

Finally note that the EA project leaded to the carrying out of new investigations in automated reasoning (see the last publications in the reference list).

5 Conclusion

Let's imagine an ideal Mathematical Assistant. What its architecture might be from the EA position?

A user communicates with the system with the help of texts written in a high-level formal input language close to the natural one. She or he submits a problem like "verify whether the given text is correct" or "how to prove the following statement", or "what is the given text about" and so on. The text, provided being syntactically correct, is treated by the part of the system that we call "reasoner". The reasoner analyzes the problem and formulates a series of tasks that it submits to the inference engine, a prover. If the prover succeeds, the resulting conclusion (e.g. human-readable proof) is given to the user and the game is over. If it fails then a kind of "morbid anatomist" makes a diagnosis and supplies it to the reasoner who tries to repair the situation. In particular, the reasoner can decide that an auxiliary statement (lemma) might be useful and start the search for those in the mathematical archives. To do that it submits a request to the archive service, we call it "librarian". After getting an answer, the reasoner begins a new proof search cycle with the modified problem and the process goes on.

The user can interact with the system by playing for the reasoner, librarian, for the morbid anatomist (provided that she or he understands the internal prover's life) or for the prover itself, deciding whether a given conjecture should be considered as valid.

Where we are with respect to the ideal? Optimistic answers are welcome.

Acknowledgements

We are grateful to our teachers. We are grateful to everybody who worked side by side with us during all that time. Special thank to the referees for their patience.

References

1962 – 1969

1. Glushkov, V.M.: Intelligent machines and the mental activity of a human. Soviet School 2, 87–91 (1962) (in Russian)
2. Kaluzhnin, L.A.: On an information language for mathematics. In: Applied Linguistic and Machine Translation, pp. 21–29. Kiev University, Kiev (1962) (in Russian)
3. Kaluzhnin, L.A., Koroliuk, V.S.: Algorithms and Mathematical Machines. Soviet School, 283 p. (1964) (in Ukrainian)
4. Anufriev, F.V., Fedyurko, V.V., Letichevskii, A.A., Asel'derov, Z.M., Didukh, I.I.: An algorithm for proving theorems in Group theory. Cybernetics and System Analysis 2(1), 20–25 (1966)
5. Anufriev, F.V.: An algorithm for proving theorems in Set theory. In: Theory of Automata, GIC, AS of UkrSSR, Kiev, vol. 4, pp. 3–24 (1967) (in Russian)
6. Glushkov, V.M., Pogrebinskii, S.B., Rabinovich, Z.L., Stognii, A.A.: Aspects of the development of digital computer architectures in the dependence of computer software systems. Cybernetics and System Analysis 3(5), 13–24 (1967)
7. Asel'derov, Z.M.: Solving equations in free groups. In: Abstract of candidate dissertation in Physics and Mathematics, GIC, AS of UkrSSR, Kiev, 12 p. (1968) (in Russian)
8. Anufriyev, F.V.: An algorithm of theorem proving in logical calculi. In: Theory of Automata, GIC, AS of UkrSSR, Kiev, pp. 3–26 (1969) (in Russian)

1970 – 1992

9. Glushkov, V.M.: Some problems in the theory of automata and artificial intelligence. Cybernetics and System Analysis 6(2), 17–27 (1970)
10. Glushkov, V.M., Kostyrko, V.F., Letichevskii, A.A., Anufriyev, F.V., Asel'derov, Z.M.: On a language for the writing of formal theories. In: Theoretical Cybernetics, GIC, AS of UkrSSR, Kiev, vol. 3, pp. 4–31 (1970) (in Russian)
11. Glushkov, V.M., Bodnarchuk, V.G., Grinchenko, T.A., Dorodnitsyna, A.A., Klimenko, V.P., Letichevskii, A.A., Pogrebinskii, S.B., Stognii, A.A., Fishman, Y.S.: ANALITIK algorithmic language for the description of computing processes using analytical transformations. Cybernetics and Systems Analysis 7(3), 513–552 (1971)
12. Anufriev, F.V., Asel'derov, Z.M.: The obviousness algorithm. Cybernetics and System Analysis 8(5), 740–768 (1972)
13. Anufriyev, F.V., Kostiakov, V.M., Malashonok, A.I.: Algorithm and computer experiment for seeking proofs of theorems in the predicate calculus. Cybernetics and System Analysis 8(5), 777–783 (1972)
14. Glushkov, V.M., Kapitonova, Y.V.: Automatic search for proofs of mathematical theorems and intelligent computers. Cybernetics and System Analysis 8(5), 709–713 (1972)
15. Glushkov, V.M., Kapitonova, Y.V., Letichevskii, A.A., Vershinin, K.P., Malevanyi, N.P.: Construction of a practical formal language for mathematical theories. Cybernetics and System Analysis 8(5), 730–739 (1972)
16. Kapitonova, Y.V., Kostyrko, V.F., Lyaletski, A.V., Degtyarev, A.I., Malashonok, A.I., Anufriev, F.V., Asel'derov, Z.M.: A brief review and bibliography of investigations into automation of search of theorem proofs in formal theories. Cybernetics and System Analysis 8(5), 714–729 (1972)
17. Degtyarev, A.I.: Question of constructing a problem-oriented procedure for the proof of theorems. Cybernetics and System Analysis 9(4), 628–629 (1973)
18. Kapitonova, Y.V., Degtyarev, A.I., Lyaletski, A.V.: Use of heuristic procedures in search programs for proofs of theorems (survey). Cybernetics and System Analysis 9(4), 630–641 (1973)

19. Vershinin, K.P.: Relationship between formal language of mathematical theories and axiomatic systems of the theory of sets. Cybernetics and System Analysis 9(4), 621–627 (1973)
20. Glushkov, V.M., Vershinin, K.P., Kapitonova, Y.V., Letichevsky, A.A., Malevanyi, N.P., Kostyrko, V.F.: On a formal language for representation of mathematical texts. search of theorem proofs in formal theories. Cybernetics and System Analysis 14(5), 755–757 (1974)
21. Degtyarev, A.I.: On heuristic procedure for proving theorems from vector spaces. In: Computer-aided Theorem Proving in Mathematics, GIC, AS of UkrSSR, Kiev, pp. 95–99 (1974) (in Russian)
22. Malashonok, A.I.: The soundness and completeness of the obviousness algorithm. In: Computer-aided Theorem Proving in Mathematics, GIC, AS of UkrSSR, Kiev, pp. 75–95 (1974) (in Russian)
23. Degtyarev, A.I.: On the use of axioms of functional reflexivity in P&R refutation procedure. Preprint 75-28, GIC, AS of UkrSSR, Kiev (1975) (in Russian)
24. Lyaletski, A.V.: On a calculus of c-clauses. In: Mathematical Issues of Intelligent Machines Theory, GIC, Kiev, pp. 34–48 (1975) (in Russian)
25. Lyaletski, A.V., Malashonok, A.I.: A calculus of c-clauses based on the clash-resolution rule. In: Mathematical Issues of Intelligent Machines Theory, GIC, AS of UkrSSR, Kiev, pp. 3–33 (1975) (in Russian)
26. Vershinin, K.P.: On the notion of the correctness of a TL-text. In: Mathematical Issues of Intelligent Machines Theory, GIC, AS of UkrSSR, Kiev, pp. 61–70 (1975) (in Russian)
27. Lyaletski, A.V.: On minimal inferences in calculi of c-clauses. Issues of Robot Theory and of Artificial Intelligence, GIC, AS of UkrSSR, Kiev, pp. 34–48 (1975) (in Russian)
28. Atayan, V.V., Vershinin, K.P., Zhezherun, A.P.: Structural processing of mathematical texts. Pattern Recognition, GIC, AS of UkrSSR, Kiev, pp. 43–54 (1978) (in Russian)
29. Zhezherun, A.P.: Implementation of dynamic syntax tools for mathematical text processing systems. In: Abstracts of the All-union Symposium on Artificial Intelligence and Automatization of Mathematical Research, GIC, AS of UkrSSR, Kiev (1978) (in Russian)
30. Degtyarev, A.I.: A monotonic paramodulation strategy. In: Abstracts of the Vth All-union Conference on Mathematical Logic, Novosibirsk, USSR, p. 39 (1979) (in Russian)
31. Degtyarev, A.I.: A strict paramodulation strategy. In: Semiotics and Informatics, vol. 12, pp. 20–22. All-union Institute of Scientific and Technical Information, Moscow (1979) (in Russian)
32. Kapitonova, Y.V., Vershinin, K.P., Degtyarev, A.I., Zhezherun, A.P., Lyaletski, A.V.: System for processing mathematical texts. Cybernetics and System Analysis 15(2), 209–210 (1979)
33. Lyaletski, A.: On a variant of Herbrand theorem. In: Abstracts of the Vth All-union Conference on Mathematical Logic, Novosibirsk, USSR, p. 87 (1979) (in Russian)
34. Lyaletski, A.: On a procedure of refutation search. In: Semiotics and Informatics, vol. 12, pp. 29–32. All-union Institute of Scientific and Technical Information, Moscow (1979) (in Russian)
35. Lyaletski, A.: On the issue of the construction of refutation procedures for which preliminary skolemization is not obligatory. In: Computer-aided Mathematical Texts Processing and Issues of Robot Construction, GIC, AS of UkrSSR, Kiev, pp. 28–35 (1979) (in Russian)
36. Morokhovets, M.K.: On the implementation of logical inference procedures in the framework of a mathematical texts processing system. In: Computer-aided Mathematical Texts Processing and Issues of Robot Construction, GIC, AS of UkrSSR, Kiev, pp. 36–41 (1979) (in Russian)
37. Vershinin, K.P.: Application of auxiliary propositions in inference search. In: Semiotics and Informatics, vol. 12, pp. 3–7. All-union Institute of Scientific and Technical Information, Moscow (1979) (in Russian)

38. Vershinin, K.P.: Refutation search and “natural” proofs. In: Computer-aided Mathematical Texts Processing and Issues of Robot Construction, GIC, AS of UkrSSR, Kiev, pp. 12–28 (1979) (in Russian)
39. Zhezherun, A.P.: Decidability of the unification problem for second-order languages with unary functional symbols. Cybernetics and System Analysis 15(5), 120–125 (1979)
40. Vershinin, K.P., Zhezherun, A.P.: Data representation in a system for processing of mathematical texts. In: Issues of the Construction and Application of Mathematical Methods and Computers, GIC, AS of UkrSSR, Kiev, pp. 7–15 (1979) (in Russian)
41. Atayan, V.V., Vershinin, K.P.: Formalization of some deduction techniques. In: Computer-aided Processing of Mathematical Texts, GIC, AS of UkrSSR, Kiev, pp. 36–52 (1980) (in Russian)
42. Degtyarev, A.I.: Some special tools of Evidence Algorithm for equality handling. In: Computer-aided Processing of Mathematical Texts, GIC, AS of UkrSSR, Kiev, pp. 30–36 (1980) (in Russian)
43. Glushkov, V.M.: The system for automatic theorem proving (SAD) (a brief informal description). In: Computer-aided Processing of Mathematical Texts, GIC, AS of UkrSSR, Kiev, pp. 3–30 (1980) (in Russian)
44. Morokhovets, M.K.: On editing proofs. In: Computer-aided Processing of Mathematical Texts, GIC, AS of UkrSSR, Kiev, pp. 53–61 (1980) (in Russian)
45. Zhezherun, A.P.: Basic tools for data processing in an automatic theorem-proving system. In: Abstracts of the All-union Conference “Methods of Mathematical Logic in Artificial Intelligence and Systematic Programming”, Palanga, Lithuania, vol. I, p. 105 (1980) (in Russian)
46. Zhezherun, A.P.: Tools for computer-aided processing of mathematical texts. In: Abstract of candidate dissertation in Physics and Mathematics, GIC, AS of UkrSSR, Kiev (1980) (in Russian)
47. Atayan, V.V.: On some tools for the construction of an information environment in a computer-aided proving system. In: Mathematical Foundations of Artificial Intelligence Systems, GIC, AS of UkrSSR, Kiev, pp. 11–17 (1981) (in Russian)
48. Degtyarev, A.I., Lyaletski, A.V.: Logical inference in the system for automatic theorem proving, SAD. In: Mathematical Foundations of Artificial Intelligence Systems, GIC, AS of UkrSSR, Kiev, pp. 3–11 (1981) (in Russian)
49. Lyaletski, A.V.: A variant of Herbrand’s theorem for formulas in prenex form. Cybernetics and System Analysis 17(1), 125–129 (1981)
50. Degtyarev, A.I.: Equality handling in proof search in theories with the complete set of reductions. In: Abstracts of the VIth All-union Conference on Mathematical Logic, Tbilisi, USSR, p. 55 (1982) (in Russian)
51. Degtyarev, A.I.: Methods and tools for equality handling in machine theorem proving. In: Abstract of candidate dissertation in Physics and Mathematics, GIC, AS of UkrSSR, Kiev, 24 p. (1982) (in Russian)
52. Lyaletski, A.V.: On a modification of Kanger’s method. In: Abstracts of the VIth All-union Conference on Mathematical Logic, pp. 98–99. Tbilisi University, Tbilisi (1982) (in Russian)
53. Lyaletski, A.V.: Methods of machine proof search in the first-order predicate calculus. In: Abstract of candidate dissertation in Physics and Mathematics, GIC, AS of UkrSSR, Kiev, 23 p. (1982) (in Russian)
54. Vershinin, K.P.: On the correctness of mathematical texts and its computer-aided verification. In: Abstract of candidate dissertation in Physics and Mathematics, GIC, AS of UkrSSR, Kiev, 20 p. (1982) (in Russian)

55. Lyaletski, A.V.: Generating sufficient statements in the SAD system. In: Abstracts of the II-Ith All-union Conference "Application of Methods of Mathematical Logic", Tallinn, USSR, pp. 65–66 (1983) (in Russian)
56. Vershinin, K.P., Morokhovets, M.K.: Strategies of the search for derivation of statements with restricted quantifiers. Cybernetics and System Analysis 19(3), 298–308 (1983)
57. Morokhovets, M.K.: Deduction-seeking procedures and transitive relations. Cybernetics and System Analysis 21(5), 702–708 (1985)
58. Degtyarev, A.I.: Equality handling methods for Horn sets. In: Methods for Algorithmization and Realization of Processes of Finding Solutions of Intelligent Problems, GIC, Kiev, pp. 19–26 (1986) (in Russian)
59. Degtyarev, A.I., Voronkov, A.A.: Equality control methods in machine theorem proving. Cybernetics and System Analysis 22(3), 298–307 (1986)
60. Morokhovets, M.K.: Special strategies for theorem proof search in mathematics and tools for their implementation. In: Abstract of candidate dissertation in Physics and Mathematics, GIC, AS of UkrSSR, Kiev, 14 p. (1986) (in Russian)
61. Voronkov, A.A., Degtyarev, A.I.: Automatic theorem proving. I. Cybernetics and System Analysis 22(3), 290–297 (1986)
62. Voronkov, A.A., Degtyarev, A.I.: Automatic theorem proving. II. Cybernetics and System Analysis 23(4), 547–556 (1987)
63. Lyaletski, A.: Gentzen calculi and admissible substitutions. In: Actes Preliminaires du Symposium Franco-Sovietique "Informatika-91", Grenoble, France, pp. 99–111 (October 1991)

1998 – Now

64. Degtyarev, A.I., Kapitonova, J.V., Letichevski, A.A., Lyaletski, A.V., Morokhovets, M.K.: A brief historical sketch on Kiev school of automated theorem proving. In: Proceedings of the 2nd International THEOREMA Workshop, Linz, Austria, pp. 151–156 (1998)
65. Kapitonova, Y., Letichevsky, A., Lyaletski, A., Morokhovets, M.: The Evidence Algorithm 2000 (a project). In: Proceedings of the 1st International Conference UkrPROG 1998, Kiev, Ukraine, pp. 68–70 (1998)
66. Degtyarev, A., Lyaletski, A., Morokhovets, M.: Evidence Algorithm and sequent logical inference search. In: Ganzinger, H., McAllester, D., Voronkov, A. (eds.) LPAR 1999. LNCS, vol. 1705, pp. 44–61. Springer, Heidelberg (1999)
67. Vershinin, K., Paskevich, A.: ForTheL — the language of formal theories. International Journal of Information Theories and Applications 7(3), 120–126 (2000)
68. Lyaletski, A., Paskevich, A.: Goal-driven inference search in classical propositional logic. In: Proceedings of the International Workshop STRATEGIES 2001, Siena, Italy, pp. 65–74 (June 2001)
69. Aselderov, Z., Verchinine, K., Degtyarev, A., Lyaletski, A., Paskevich, A., Pavlov, A.: Linguistic tools and deductive technique of the System for Automated Deduction. In: Implementation of Logics, the 3rd International Workshop "WIL 2002", Tbilisi, Georgia, pp. 21–24 (October 2002)
70. Lyaletski, A., Verchinine, K., Degtyarev, A., Paskevich, A.: System for Automated Deduction (SAD): Linguistic and deductive peculiarities. In: Klopotek, M.A., Wierzchon, S.T., Michalewicz, M. (eds.) Intelligent Information Systems, the 11th International Symposium, IIS 2002. Advances in Soft Computing, pp. 413–422. Sopot, Poland. Physica-Verlag, Heidelberg (June 2002)
71. Paskevich, A.: Reasoning inside a formula. SEKI Report SR-02-06, University of Kaiserslautern (2002)
72. Verchinine, K., Degtyarev, A., Lyaletski, A., Paskevich, A.: SAD, a System for Automated Deduction: a current state. In: Kamareddine, F. (ed.) Proceedings of the Workshop on 35 Years of Automath, Edinburgh, Scotland (April 2002)

73. Lyaletski, A.: Admissible substitutions in sequent calculi. International Journal on Information Theories and Applications 10(4), 388–393 (2003)
74. Lyaletski, A.: Evidential paradigm: the logical aspect. Cybernetics and System Analysis 39(5), 659–667 (2003)
75. Lyaletski, A., Verchinine, K., Paskevich, A.: On verification tools implemented in the System for Automated Deduction. In: Implementation Technology for Computational Logic Systems, the 2nd CoLogNet Workshop, ITCLS 2003, Pisa, Italy, pp. 3–14 (September 2003)
76. Lyaletski, A., Verchinine, K., Paskevich, A.: Theorem proving and proof verification in the system SAD. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 236–250. Springer, Heidelberg (2004)
77. Konev, B., Lyaletski, A.: Tableau method with free variables for intuitionistic logic. In: Klopotek, M., Wierzchon, S., Trojanowski, K. (eds.) Proceedings of the International IIS:IIPWM 2006 Conference. Intelligent Information Processing and Web Mining, Ustron, Poland, pp. 293–305. Springer, Heidelberg (June 2006)
78. Lyaletski, A.: Sequent forms of Herbrand theorem and their applications. Annals of Mathematics and Artificial Intelligence 46(1-2), 191–230 (2006)
79. Lyaletski, A., Konev, B.: On Herbrand's theorem for intuitionistic logic. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 293–305. Springer, Heidelberg (2006)
80. Lyaletski, A., Paskevich, A., Vershinin, K.: Deductive assistance in an intelligent linguistic environment. In: Proceedings of the 3th EIII Conference on Intelligent Systems (IEEE IS 2006), London, UK (September 2006)
81. Lyaletski, A., Paskevich, A., Verchinine, K.: SAD as a mathematical assistant — how should we go from here to there? Journal of Applied Logic 4(4), 560–591 (2006)
82. Lyaletski (Sr.), A., Lyaletsky Jr., A.: Admissible substitutions and Herbrand's theorems for classical and intuitionistic logics. In: Baaz, M., Preining, N. (eds.) Geodel Centenary 2006: Collegium Logicum, Posters, Vienna, Austria, vol. IX, pp. 41–45. Springer, Heidelberg (April 2006)
83. Paskevych, A.: Methodes de formalisation des connaissances et des raisonnements mathematiques: aspects appliques et theoriques. PhD thesis, Universite Paris 12 (2007) (in French)
84. Paskevich, A., Verchinine, K., Lyaletski, A., Anisimov, A.: Reasoning inside a formula and ontological correctness of a formal mathematical text. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) Calculemus/MKM 2007 — Work in Progress, RISC-Linz Report Series, Hagenberg, Austria, vol. 07-06, pp. 77–91. University of Linz, Austria (2007)
85. Verchinine, K., Lyaletski, A., Paskevich, A.: System for Automated Deduction (SAD): A tool for proof verification. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 398–403. Springer, Heidelberg (2007)
86. Lyaletski, A.: On some problems of efficient inference search in first-order cut-free modal sequent calculi. In: Proceedings of the 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, pp. 39–46. IEEE Inc., Los Alamitos (2008)
87. Lyaletski, A.: Herbrand theorems: the classical and intuitionistic cases. In: Philosophical Logic, Poland. Studies in Logic, Grammar and Rhetoric, vol. 14(27), pp. 101–122 (2008)
88. Paskevich, A.: Connection tableaux with lazy paramodulation. Journal of Automated Reasoning 40(2-3), 179–194 (2008)
89. Verchinine, K., Lyaletski, A., Paskevich, A., Anisimov, A.: On correctness of mathematical texts from a logical and practical point of view. In: Autexier, S., Campbell, J., Rubio, J., Sorge, V., Suzuki, M., Wiedijk, F. (eds.) AISC 2008, Calculemus 2008, and MKM 2008. LNCS (LNAI), vol. 5144, pp. 583–598. Springer, Heidelberg (2008)

90. Lyaletski, A.: On Herbrand-like theorems for cut-free modal sequent logics. In: Proceedings of the 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania. IEEE Inc., Los Alamitos (2009)

References to Necessary Papers of Other Authors

91. Brand, D.: Proving theorems with the modification method. SIAM Journal of Computing 4, 412–430 (1975)
92. Davis, M.: The early history of automated deduction. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 3–15. MIT Press, Cambridge (2001)
93. Lifschitz, V.: Mechanical theorem proving in the USSR: The Leningrad school. Delphic Associates, Inc. (1986)
94. Lifschitz, V.: What is the inverse method? Journal of Automated Reasoning 5(1), 1–23 (1989)
95. Martelli, A., Montanari, U.: An efficient unification algorithm. ACM Trans. on Prog. Languages and Systems 4(2), 258–282 (1982)
96. Maslov, S.Y.: The inverse method for establishing the deducibility in the classical predicate calculus. DAN SSSR 159(1), 17–20 (1964) (in Russian)
97. Matulis, V.A.: The first All-union symposium on the problem of machine searching the logical deduction. UMN 19(6), 239–241 (1964) (in Russian)
98. McCune, W.: Otter 3.0 reference manual and guide. Tech. Report ANL-94/6, Argonne National Laboratory, Argonne, USA
99. Prover9. Home page, http://www.cs.unm.edu/~mccune/prover9/
100. Riazanov, A., Voronkov, A.: The design and implementation of VAMPIRE. AI Communications 15(2-3), 91–110 (2002)
101. Robinson, J.A.: A machine-oriented logic based on the resolution principle. ACM 12(1), 23–41 (1965)
102. Robinson, J.A., Voronkov, A. (eds.): Handbook of Automated Reasoning (in 2 volumes). Elsevier and MIT Press (2001)
103. Schulz, S.: System Description: E 0.81. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004. LNCS (LNAI), vol. 3097, pp. 223–228. Springer, Heidelberg (2004)
104. Wang, H.: Towards mechanical mathematics. IBM J. of Research and Development 4, 2–22 (1960)
105. Weidenbach, C., Schmidt, R., Hillenbrand, T., Rusev, R., Topic, D.: SPASS version 3.0. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 514–520. Springer, Heidelberg (2007)
106. Wiedijk, F. (ed.): The Seventeen Provers of the World. LNCS (LNAI), vol. 3600, 184 p. Springer, Heidelberg (2006)

On Building a Knowledge Base for Stability Theory

Agnieszka Rowinska-Schwarzweller[1] and Christoph Schwarzweller[2]

[1] Chair of Display Technology, University of Stuttgart
Allmandring 3b, 70569 Stuttgart, Germany
`schwarzweller@neostrada.pl`

[2] Department of Computer Science, University of Gdańsk
ul. Wita Stwosza 57, 80-952 Gdańsk, Poland
`schwarzw@inf.ug.edu.pl`

Abstract. A lot of mathematical knowledge has been formalized and stored in repositories by now: Different mathematical theorems and theories have been taken into consideration and included in mathematical repositories. Applications more distant from pure mathematics, however — though based on these theories — often need more detailed knowledge about the underlying theories. In this paper we present an example Mizar formalization from the area of electrical engineering focusing on stability theory which is based on complex analysis. We discuss what kind of special knowledge is necessary and which amount of this knowledge is included in existing repositories.

1 Introduction

The aim of mathematical knowledge management is to provide both tools and infrastructure supporting the organization, development, and teaching of mathematics with the help of effective up-to-date computer technologies. To achieve this ambitious goal it should be taken into account that the predominant part of potential users will not be professional mathematicians themselves, but rather scientists or teachers that apply mathematics in their special domain. To attract these people it is essential that our repositories provide a sufficient knowledge base for those domains. We are interested in how far existing mathematical repositories are from meeting this precondition, or in other words: How large is the gap between the knowledge already included in repositories and the knowledge necessary for particular applications?

This problem, however, concerns not only the simple question how much knowledge of a domain is available in a repository. We believe, that in order to measure this gap, it is equally important to consider the basic conditions for a successful formalization of applications on top of existing knowledge, that is on top of a mathematical repository: The more easy such a formalization is, the more attractive is a mathematical repository. To describe attractiveness of a repository for an application one can identify three major points:

1. Amount of knowledge
   This is the obvious question of how much knowledge of a particular domain already has been formalized and included in the repository. Basically, the more knowledge of a domain is included the more attractive a repository is for applications.
2. Representation of knowledge
   This concerns the question of how the knowledge has been defined and formalized: Often mathematicians use more abstract constructions than necessary — and attractive — for applications. An example is the construction of rational functions from polynomials.
3. Applicability of knowledge
   This point deals with both how the knowledge of a domain is organized in a repository and the question of how easy it is to adapt available knowledge to one's own purposes.

In this paper we focus on electrical engineering, in particular on network stability [Unb93]. Network theory deals with the mathematical description, analysis and synthesis of electrical (e.g. continuous, time-discrete or digital) networks. For a reliable application such systems have to be (input/output-) stable, that is for an arbitrary bounded input the output have to be bounded again. In practice it is impossible to verify responses for all input signals. In this situation there is, however, a number of theorems permitting easier methods to decide whether a network is stable [Unb93]. We shall introduce the mathematical fundamentals and prequisites of one example theorem and present a Mizar formalization of this theorem. After that we discus our formalization in the spirit of the three points from above: How far are we from providing a suitable mathematical repository for applications in stability theory?

2 Stability of Networks

As mentioned in the introduction the (input/output-) stability of networks is one of the main issues when dealing with the analysis and design of electrical circuits and systems. In the following we briefly review definitions and properties of electrical systems necessary to understand the rest of the paper. In electrical engineering stability applies to the input/output behaviour of networks (see figure 1). For (time-) continuous systems one finds the following definition. For discrete systems an analogous definition is used.

Definition 1. ([Unb93])
A continuous system is (BIBO-)[1] stable, if and only if each bounded input signal $x(t)$ results in a bounded output signal $y(t)$.

Physically realizable, linear time-invariant systems (LTI systems) can be described by a set of differential equations [Unb93]. The behaviour of a LTI system

[1] BIBO stands for Bounded Input Bounded Output.

then is completely characterized by its impulse response $h(t)$.[2] If the impulse response of such a system is known, the relation between the input $x(t)$ and the output $y(t)$ is given by the convolution integral

$$y(t) \;=\; \int\limits_{-\infty}^{\infty} x(\tau)h(t-\tau)\mathrm{d}\tau. \tag{1}$$

Furthermore, a LTI system is stable, if and only if its impulse response $h(t)$ is absolute integrable, that is there exists a constant K such that

$$\int\limits_{-\infty}^{\infty} |h(\tau)|\,\mathrm{d}\tau \;\le\; K \;<\; \infty. \tag{2}$$

In network and filter analysis and design, however, one commonly employs the frequency domain rather than the time domain. To this end the system is described based on its transfer function $H(s)$. In case the Laplace transformation is used we have[3]

$$H(s) \;=\; \int\limits_{-\infty}^{\infty} h(t)e^{-st}\mathrm{d}t.$$

where $s = \sigma + j\omega$ is a complex variable with $\Re\{s\} = \sigma$ and $\Im\{s\} = \omega$.

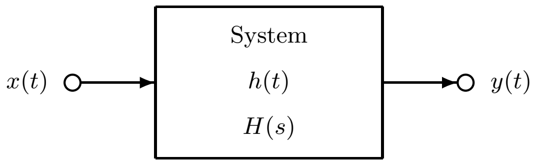


Fig. 1. LTI system with one input $x(t)$ and one output $y(t)$

The evaluation of $H(s)$ for $s = j\omega$ — in case of convergence[4] — enables the qualitative understanding of how the system handles and selects various frequencies ω, so for example whether the system describes a high-pass filter, low-pass filter, etc. Now the necessary condition to demonstrate the stability of LTI systems in the frequency domain reduces to show, that the $j\omega$-axis lies in the Laplace transformation's region of convergence (ROC).

For physically realizable LTI systems, such as the class of networks with constant and concentrated parameters (so-called lumped parameters circuits), $H(s)$ is given in form of a rational function with real coefficients, that is

$$H(s) \;=\; \frac{a_n s^n + \ldots + a_0}{b_m s^m + \ldots + b_0}, \quad a_i, b_i \in \mathbb{R}. \tag{3}$$

[2] $h(t)$ is the output of the system, when the input is the Dirac delta function $\delta(t)$.

[3] Note that this is a generalization of the continuous-time Fourier transformation.

[4] In this case $H(j\omega)$ equals the Fourier transform.

In this case the region of convergence can be described by the roots of the denominator polynomial: If $s_i = \sigma_i + j\omega_i$ for $i = 1, \dots m$ are the roots of $b_m s^m + \dots + b_0$, the region of convergence is given by

$$\Re\{s\} \ > \max\{\sigma_i, \ i = 1, \dots m\}.$$

To check stability it is therefore sufficient, to show that the real part $\Re\{s\}$ of all poles of $H(s)$ is smaller then 0. The denominator of $H(s)$ is thus a so-called Hurwitz polynomial.

Note that the stability problem for discrete-time signals and systems can be analized with the same approach. For a given discrete-time transfer function $H(z)$ in the Z- domain, it has to be checked whether the unit circle is contained in the region of convergence. Hence for all poles z_i of $H(z)$ we must have $|z_i| < 1$. Using bilinear transformations [OS98]

$$z \ := \ \frac{1+s}{1-s}. \tag{4}$$

it is thus sufficient to check whether the denominator of

$$H(z)|_{z:=\frac{1+s}{1-s}} \tag{5}$$

is a Hurwitz polynomial.

The practical proof of stability of high-precision filters, however, turns out to be very hard. In practical applications the poles of concern are usually close to the axis $s = j\omega$ or the unit circle $|z| = e^{j\omega}$ respectively. Thus numerical determination of the poles is highly error-prone due to its rounding effects. In digital signal processing in addition degrees of transfer functions tend to be very high, for example 128 and higher in communication networks.

An interesting and in practice often used method to check the stability of a given network is based on the following theorem.

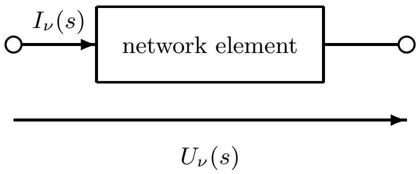
Theorem 1. ([Unb93])

Let $f(x)$ be a real polynomial with degree $n \geq 1$. Furthermore let all coefficients of $f(x)$ be greater than 0. Let $f_e(x)$ and $f_o(x)$ denote the even part resp. the odd part of $f(x)$. Assume further that

$$Z(x) \ = \ \frac{f_e(x)}{f_o(x)}$$

or the reciprocal of $Z(x)$ is a reactance one-port function of degree n. Then $f(x)$ is a Hurwitz polynomial.

The concept of reactance one-port function stems from electrical network theory: In arbitrary passive (that is RLC-) networks we find the following relations between the complex voltage $U_\nu(s)$ and the complex current $I_\nu(s)$:



$$\begin{aligned} U_r(s) &= R_r \cdot I_r(s) && \text{for a resistor } R_r \\ U_l(s) &= s \cdot L_l \cdot I_l(s) && \text{for an inductance } L_l \\ U_k(s) &= \tfrac{1}{s \cdot C_k} \cdot I_k(s) && \text{for a capacity } C_k \end{aligned}$$

An impedance (complex resistor) or admittance (complex conductance) composed of network elements R, L and C only is called a (RLC-) one-port function, an impedance or admittance composed of network elements L and C only is called a reactance one-port function. Conversely, for every one-port function $Z(s)$ there exists at least one one-port, whose impedance or admittance is equal to $Z(s)$:

Hence, theorem 1 reduces stability checking to the considerable easier task to synthesize a one-port solely using inductors (L) and capacitors (C), that is to synthesize a reactance one-port. To this end there exist easy procedures like for example Routh's method to construct a chain one-port [Unb93].

It turns out that one-port functions $Z(s)$ are exactly the real positive rational functions. For a real function we have that for real s also $Z(s)$ is real[5], and a positive function means that $\Re\{s\} > 0$ implies $\Re\{Z(s)\} > 0$. A reactance one-port function is a one-port function, that is in addition odd. The property of being positive is closely connected to Hurwitz polynomials:

Theorem 2. ([Unb93])
Let $f(x)$ be a real polynomial with degree $n \geq 1$. Furthermore let all coefficients of $f(x)$ be greater than 0. Let $f_e(x)$ and $f_o(x)$ denote the even part resp. the odd part of $f(x)$. Assume that$f_e(x)$ and $f_o(x)$ have no common roots and that $Z(x) = f_e(x)/f_o(x)$ is positive. Then

(i) $\Re\{Z(x)\} \geq 0$ for all x with $\Re\{x\} = 0$
(ii) $f_e(x) + f_o(x)$ is a Hurwitz polynomial.

In section 4.2 we will see that this theorem is also the key to prove that stability checking can be reduced to synthesizing LC-one-ports, in other words to prove theorem 1 from above.

3 The Mizar System

The logical basis of Mizar [RT01, Miz10] is classical first order logic extended, however, with so-called schemes. Schemes introduce free second order variables,

[5] This condition implies that the coefficients of $Z(s)$ are real. In network theory, however, this definition is used.

in this way enabling among others the definition of induction schemes. In addition Mizar objects are typed, the types forming a hierarchy with the fundamental type `set`. The user can introduce new (sub)types describing mathematical objects such as groups, fields, vector spaces or polynomials over rings or fields. To this end the Mizar language provides a powerful typing mechanism based on adjective subtypes [Ban03].

The current development of Mizar relies on Tarski-Grothendieck set theory — a variant of Zermelo Fraenkel set theory using Tarski's axiom on arbitrarily large, strongly inaccessible cardinals [Tar39] which can be used to prove the axiom of choice —, though in principle the Mizar language can be used with other axiom systems also. Mizar proofs are written in natural deduction style as presented in the calculus of [Jaś34]. The rules of the calculus are connected with corresponding (English) natural language phrases so that the Mizar language is close to the one used in mathematical textbooks. The Mizar proof checker verifies the individual proof steps using the notion of obvious inferences [Dav81] to shorten the rather long proofs of pure natural deduction.

4 Mizar Formalization of the Theorem

4.1 Preliminaries about Rational Functions

Although the theory of polynomials in Mizar is rather well developed, rational functions have not been defined yet. In the following we briefly review how we have introduced rational functions in Mizar. Rational functions can — analogously to polynomials — be defined over arbitrary fields: Rational functions are simply pairs of polynomials whose second component is not the zero polynomial.[6] These can be easily introduced as a Mizar type `Rational_function of L`, where `L` is the underlying coefficient domain.

```
definition
let L be non trivial multLoopStr_0;
mode rational_function of L means
  ex p1 being Polynomial of L st
  ex p2 being non zero Polynomial of L st it = [p1,p2];
end;
```

Note that in Mizar the result type of the pair constructor `[ , ]` and the projections `‘1` and `‘2`, in the original definition is simply `set`. These, however, can be modified into `Rational_function` and `(non zero) Polynomial` respectively by employing `redefinitions`. In addition one can introduce the usual functions `num` and `denom` as synonyms for the corresponding projections.

[6] Of course rational functions can be introduced "more algebraically" as the quotient field of a polynomial ring. Here we decided to use pairs to concentrate on application issues; see the discussion in section 5.

```
definition
let L be non trivial multLoopStr_0;
let p1 be Polynomial of L;
let p2 be non zero Polynomial of L;
redefine func [p1,p2] -> rational_function of L;
end;

definition
let L be non trivial multLoopStr_0;
let z be rational_function of L;
redefine func z`1 -> Polynomial of L;
redefine func z`2 -> non zero Polynomial of L;
end;

notation
let L be non trivial multLoopStr_0;
let z be rational_function of L;
synonym num(z) for z`1;
synonym denom(z) for z`2;
end;
```

Now that `num` and `denom` — applied to rational functions — have result type `Polynomial`, operations for rational functions can straightforwardly be defined by employing the corresponding functions for polynomials. So, for example, the evaluation of rational functions can be defined using evaluation of polynomials and the division operator `/` defined for arbitrary fields `L`.

```
definition
let L be Field;
let z be rational_function of L;
let x be Element of L;
func eval(z,x) -> Element of L equals
  eval(num(z),x) / eval(denom(z),x);
end;
```

Note that according to the definition of `eval` for polynomials the type of the first argument — that is of `num(z)` and `denom(z)` — has to be `Polynomial`. This is ensured by the redefinitions from above, which in this sense allow for reusing the operations defined for polynomials in the case of rational functions. Other necessary operations for rational functions such as the degree or arithmetic operations can be defined the same way.

4.2 The Theorem

To apply the general Mizar theory of polynomials and rational functions with complex numbers we just instantiate the parameter `L` describing the coefficient domain with the field of complex numbers `F_Complex` from [Mil01a]. So an object of type

```
rational_function of F_Complex
```

combines the theory of rational functions with the one of complex numbers.

Further properties necessary to state the main theorem are introduced by defining appropriate attributes for polynomials and rational functions resp. Note that these definitions apply to polynomials and rational functions over the complex numbers only.

```
definition
let p be Polynomial of F_Complex;
attr p is real means
  for i being Element of NAT holds p.i is real number;
end;

definition
let p be rational_function of F_Complex;
attr Z is positive means
  for x being Element of F_Complex
  holds Re(x) > 0 implies Re(eval(Z,x)) > 0;
end;
```

Using these attributes — and the attribute `odd` describing odd functions — we can then introduce one-ports and reactance one-ports in Mizar by the following mode definitions.

```
definition
mode one_port_function is real positive rational_function of F_Complex;

mode reactance_one_port_function is
                      odd real positive rational_function of F_Complex;
end;
```

We also needed to define the odd and the even part of a polynomial `f`. This is accomplished by two Mizar functors `even_part(f)` and `odd_part(f)`, which however can be defined straightforwardly. Finally we formalize the condition from the theorem, that all the coefficients of the given polynomial f should be greater than 0 as usual as a Mizar attribute:

```
definition
let f be real Polynomial of F_Complex;
attr f is with_all_coefficients_positive means
  for i being Element of NAT st i <= deg p holds p.i > 0;
end;
```

Note that for a real polynomial f where all coeffiecients of f are greater than 0 and $\deg(f) \geq 1$ both the even and the odd part of f are not 0, hence both can appear as the denominator of a rational function. Thus prepared we can state theorem 1 from section 2 in Mizar. Note again that due to the redefinitions of section 4.1 the functor `[ , ]` returns a rational function.

```
theorem
for p be non constant with_positive_coefficients
        (real Polynomial of F_Complex)
st [even_part(p),odd_part(p)] is reactance_one_port_function &
   degree([even_part(p),odd_part(p)]) = degree p
holds p is Hurwitz;
```

The proof of the theorem, as already indicated, basically relies on theorem 2 from section 2, which connects rational functions with the property of being a Hurwitz polynomial. Based on our development from above one can formulate this theorem as follows.

```
theorem
for p be non constant with_positive_coefficients
        (real Polynomial of F_Complex)
st [even_part(p),odd_part(p)] is positive &
   even_part(p),odd_part(p) have_no_common_roots
holds (for x being Element of F_Complex
        st Re(x) = 0 & eval(odd_part(p),x) <> 0
        holds Re(eval([even_part(p),odd_part(p)],x)) >= 0) &
      even_part(p) + odd_part(p) is Hurwitz;
```

The corresponding Mizar proof is rather technical. The basic idea consists of considering in addition to

$$Z(x) \;=\; \frac{p_e(x)}{p_o(x)} \tag{6}$$

the rational function

$$W(x) \;=\; \frac{Z(x)-1}{Z(x)+1} = \frac{p_e(x)-p_o(x)}{p_e(x)+p_o(x)}. \tag{7}$$

and to analyze the absolute values $|W(x)|$. If $Z(x)$ is positive, then $|W(x)| \leq 1$ for all x with $\Re\{x\} \geq 0$, which implies that $W(x)$ has no poles for $\Re\{x\} \geq 0$. Thus the denominator polynomial of $W(x)$ can have roots only for $\Re(x) < 0$, so $p_e(x) + p_o(x) = p(x)$ is a Hurwitz polynomial.

The main theorem now easily follows from theorem 2, because the degree condition implies that $p_e(x)$ and $p_o(x)$ have no common roots.

5 Discussion — Lessons Learned

In the following we discuss our formalization from the last section with respect to the three criteria presented in the introduction. Though carried out for Mizar and the Mizar Mathematical Libray (MML) we believe that the situation in other repositories is similar, so that most of our results should hold in a more general context also.

In [RS07] we already presented a Mizar formalization of Schur's theorem, another helpful criterion for stability checking. Based on polynomials only its formalization was rather harmless. The only missing point that caused some work was division of polynomials. However, as we will see, stability checking in general needs definitely more extension than in this case.

5.1 Amount of Knowledge

Complex numbers and polynomials (over arbitrary rings) are included in MML. A lot of theorems have been proved here, so that almost all we needed could be found in the repository. Interestingly rational functions — a rather basic structure — had not been defined, yet. The reason might be that rational functions are mathematically rather simple and this is the first time that a theorem relying on rational functions has been formalized.

Though even and odd functions were already included in MML, the even and odd part of a polynomial was not. This, however, comes with no surprise, just because these polynomial operations are rather seldom used. We hence had to prove a number of theorems dealing with these polynomials, most of them however being elementary like for example

```
theorem
for p being real Polynomial of F_Complex
for x being Element of F_Complex st Re(x) = 0
holds Re(eval(odd_part(p),x)) = 0;
```

Summarizing, except for the lack of rational functions, MML provides the amount of knowledge for stability checking one could expected.

5.2 Representation of Knowledge

The construction of rational functions can be performed in different ways. Of course, one can define rational functions as pairs of polynomials. On the other hand there is the possibility to construct (the field of) rational functions as the completion of polynomial rings. Though the second version is mathematically more challenging we decided to use pairs. We wanted to emphasize the contribution to applications by concentrating on prior knowledge of potential users: Electrical engineers are probably not interested in (working with!) abstract algebra, their interests and needs are different.

In the same context there is another representational problem: In MML we find both the complex numbers and the field of complex numbers. Not a problem in itself, this may cause some confusion when searching for notions and theorems: The functors `Re` and `Im` giving the real and imaginary part, for exmaple, are defined for complex numbers only, thus — theoretically — not applicable to elements of a field. In Mizar, however, this is not necessarily the case: Using a special registration — `identify` [Kor09] — the user can identify terms and operations from different structures, here complex numbers with elements of the field of complex numbers:

```
registration
  let a,b be complex number;
  let x,y be Element of F_Complex;
  identify x+y with a+b when x=a, y=b;
  identify x*y with a*b when x=a, y=b;
end;
```

In effect, after this registration functors `Re` and `Im` are applicable to elements of the field of complex numbers.

In general, different views on mathematical objects — here, complex numbers as numbers or elements of a field — have to be handled carefully in mathematical repositories in order to not confuse possible users. Even the rudimentary difference between a polnomial and its polynomial function can lead to surprises and incomprehension for people not familiar with the formal treatment of mathematics in repositories.

5.3 Applicability of Knowledge

As we have already seen, the adaption of general knowledge in MML to special cases is straightforward: One just instantiates parameters describing the general domain with the special one, so for example `Polynomial of F_Complex` for polynomials over the (field of) complex numbers.

This, on the other hands, means that to work with such instantiations the user has to apply theorems about the general structure. Though highly desirable from the mathematical point of view, it is not clear whether this is really convenient for application users: To work in the special field of complex numbers, for example, then means to search for helpful theorems in the theory of fields, rings or even groups and semigroups. Maybe here a search tool that generates and collects theorems for special instances of theories would be a reasonable help.

The organization of MML is mainly by articles in which authors prove not only their main theorems, but also whatever is necessary and not found in MML. As a consequence theorems of the same topic, e.g. polynomials, can be spread over the repository. A step to overcome this shortcoming is an ongoing project called Mizar encyclopedia building articles with monographic character whose contents is semi-automatically extracted from contributed Mizar articles. Unfortunately polynomials have not been considered in this project, yet.

Summarizing the Mizar system though flexible in order to support special applications lacks an organization of its corresponding repository to support application users in their efforts.

6 Conclusions

We have presented a Mizar formalization of a theorem for stability checking and have discussed how the knowledge contained in MML supported the process from an application user's point of view. Here we want to emphasize two points.

First, when building a knowledge base for an application area, it is difficult to foresee what knowledge is necessary. We have seen that the formalization of Schur's theorem went through without major problems, while the present theorem caused definitely more work and preparation. Furthermore, there are theorems on stability checking using even involved mathematical techniques such as e.g. analytic functions and the maximum principle.

Second, attractiveness of mathematical repositories does not only depend on the amount of knowledge included. Equally important are a clear representation and organization of knowledge so that it in particular stays familiar for users outside the mathematical community.

Consequently is it essential to communicate with experts from the application area. If we want our repositories to be widely used, we have both to provide a reasonable knowledge base and to take care of the fact that application users might represent mathematical knowledge in a different way we are used to. To this end, we need much more formalizations in application domains. Analysis of such formalizations may then lead to methodologies for building applications on top of mathematical repositories and hence may close the gap between knowledge already included in repositories and knowledge necessary for particular applications.

References

[Byl90] Byliński, C.: The Complex Numbers. Formalized Mathematics 1(3), 507–513 (1990)

[Ban03] Bancerek, G.: On the Structure of Mizar Types. In: Geuvers, H., Kamareddine, F. (eds.) Proc. of MLC 2003. ENTCS, vol. 85(7) (2003)

[Dav81] Davies, M.: Obvious Logical Inferences. In: Proceedings of the 7th International Joint Conference on Artificial Intelligence, pp. 530–531 (1981)

[DeB87] de Bruijn, N.G.: The Mathematical Vernacular, a language for mathematics with typed sets. In: Dybjer, P., et al. (eds.) Proc. of the Workshop on Programming Languages, Marstrand, Sweden (1987)

[Jaś34] Jaśkowski, S.: On the Rules of Suppositon in Formal Logic. In: Studia Logica, vol. 1 (1934)

[Kor09] Korniłowicz, A.: How to Define Terms in Mizar Effectively. Studies in Logic, Grammar and Rhetoric 18(31), 67–77 (2009)

[Mil01a] Milewska, A.J.: The Field of Complex Numbers. Formalized Mathematics 9(2), 265–269 (2001)

[Mil01b] Milewski, R.: The Ring of Polynomials. Formalized Mathematics 9(2), 339–346 (2001)

[Miz10] The Mizar Home Page, http://mizar.org

[NB04] Naumowicz, A., Byliński, C.: Improving Mizar texts with properties and requirements. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 290–301. Springer, Heidelberg (2004)

[OS98] Oppenheim, A.V., Schafer, R.W.: Discrete-Time Signal Processing, 2nd edn. Prenctice-Hall, New Jersey (1998)

[RS07] Rowinska-Schwarzweller, A., Schwarzweller, C.: Towards Mathematical Knowledge Management for Electrical Engineering. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, pp. 371–380. Springer, Heidelberg (2007)

[RT01] Rudnicki, P., Trybulec, A.: Mathematical Knowledge Management in Mizar. In: Buchberger, B., Caprotti, O. (eds.) Proc. of MKM 2001, Linz, Austria (2001)

[Sch10] Schwarzweller, C.: Rational Functions. Journal of Formalized Mathematics (to appear)

[SR10] Schwarzweller, C., Rowinska-Schwarzweller, A.: A Theorem for Checking Stability of Networks. Journal of Formalized Mathematics (to appear)

[Tar39] Tarski, A.: On Well-Ordered Subsets of Any Set. Fundamenta Mathematicae 32, 176–183 (1939)

[Unb93] Unbehauen, R.: Netzwerk-und Filtersynthese: Grundlagen und Anwendungen (4. Auflage). Oldenbourg-Verlag, München (1993)

Proviola: A Tool for Proof Re-animation

Carst Tankink[1], Herman Geuvers[1,2], James McKinna[1], and Freek Wiedijk[1]

[1] Radboud University Nijmegen
Institute for Computing and Information Sciences
Heyendaalseweg 135, 6525 AJ Nijmegen
[2] Technical University Eindhoven
The Netherlands

Abstract. To improve on existing models of interaction with a proof assistant (PA), in particular for storage and replay of proofs, we introduce three related concepts, those of: a **proof movie**, consisting of frames which record both user input and the corresponding PA response; a **camera**, which films a user's interactive session with a PA as a movie; and a **proviola**, which replays a movie frame-by-frame to a third party.

In this paper we describe the movie data structure and we discuss a prototype implementation of the camera and proviola based on the ProofWeb system [7]. ProofWeb uncouples the interaction with a PA via a web-interface (the client) from the actual PA that resides on the server. Our camera films a movie by "listening" to the ProofWeb communication.

The first reason for developing movies is to uncouple the reviewing of a formal proof from the PA used to develop it: the movie concept enables users to discuss small code fragments without the need to install the PA or to load a whole library into it.

Other advantages include the possibility to develop a separate **commentary track** to discuss or explain the PA interaction. We assert that a combined camera+proviola provides a generic layer between a client (user) and a server (PA). Finally we claim that movies are the right type of **data** to be stored in an encyclopedia of formalized mathematics, based on our experience in filming the Coq standard library.

1 Introduction

Interaction with modern theorem-proving tools, Proof Assistants (PAs), still imposes heavy (temporal, spatial, computational, cognitive) resource demands on users. It is hard to write a proof, but typically even harder to write a formalized version with a PA. This additional overhead arises from at least the following:

- It is necessary to become (at least somewhat) familiar with the technical details of the PA, since the user needs to install and configure it before use;
- The user needs to understand the tools the PA gives her, in terms of libraries and commands, and how to use them to achieve a formalization of the proof;
- The user needs to know the proof and its implicit assumptions in far greater detail than required to communicate the main ideas to another person.

Much of the effort in interaction design for PAs has focused on the second and third of these issues from the point of view of defining a language of suitable basic proof steps, augmented with automation layers which are either fully programmable, or else encapsulate well-defined larger-scale proof steps, together with an editing model of how to soundly maintain a partially completed proof.

That is to say, the basic PA use case of "writing a proof" has received most attention, while those of "reading a proof" (written by someone else) or "browsing a library" rather less so: the narrative or explanatory possibilities afforded by a formal proof text have been largely overlooked in favour of (variously prettily rendered) static digests of named definitions and theorem statements. These are necessary prerequisites for these use cases, but hardly sufficient for gaining insight into how such proofs "work" (or even: how partial proof attempts may *fail*, as when trying to discuss such examples on a mailing list).

In each use case, however, there still remain the computational and cognitive bottlenecks arising from the first and second problems. For example, as illustrated by Kaliszyk [8], before being able to formalize a theorem in Isabelle, a user needs to install the system, comprising the program itself, an HOL heap and a version of PolyML. This does not include a user interface, provided by the Emacs-based ProofGeneral. Having installed the PA, the user is then left with understanding it: digging through a tutorial or manual, and finding out what contributions and libraries are necessary for the formalization of the proof.

Our work contributes to addressing the first and second issues, reducing the overheads of installing and configuring a PA and simplifying access to, and communication about, existing proofs. The third issue we do not address here.

Previous work at Nijmegen has partially addressed the first issue. By providing a generic web-interface for PAs, the ProofWeb system [8] removes the computational load on users by uncoupling interaction with a PA via a dedicated webserver. This relieves the user from the installation problem: she can just visit a website and use a web-interface to access the PA. However, when trying to understand a (part of a) formal proof script, it is often necessary to see how the proof state changes through execution of a specific tactic. This requires first to bring the PA into that specific state — finding and loading the required libraries and files — a significant overhead, not addressed by ProofWeb.

Similarly, when explaining a tactic or a part of a proof script, with the current technology, a proof author has to publish the proof script, sometimes with an explanation of the output the PA returns to her. This is not very satisfactory and often not informative enough, especially when the publication of the script is intended to show the intricacies of the proof: if the reviewer is uninitiated in the specifics of the PA, she might not understand the proof script.

This paper discusses the design of a further uncoupling layer, providing a client-side Proxy to the server-side PA output. What we do is send a script piecewise to the PA and record the response: that is, we "film" the interaction with the PA. So by analogy with film-editing, we have dubbed our system a "proviola": a playback and editing suite for "proof movies" created by submitting formal proof texts, "scripts", to the PA.

In the present paper we describe the basic ideas behind the movie-camera-proviola concept by discussing two use cases and a prototype implementation based on ProofWeb, which can be inspected at `http://mws.cs.ru.nl/proviola`.

Then we further examine the versatility of the concept and observe that we can further use our movies as a drop-in replacement for the existing ProofWeb interaction model for editing proofs: the proof movie then becomes the "file in the middle" that receives interactive updates from the user and the PA.

1.1 Contributions

In this paper, we describe the design of an architecture for capturing the interaction between a PA and its users. Specifically, we:

– articulate two roles and associated use cases: *creating* and *reading* a proof;
– define a **proof movie** datastructure that encapsulates interaction between a proof author and a PA; such wrapping affords a reviewer fast access to details of this interaction;
– have built tools for creating and viewing movies: **camera** and **proviola**;
– identify the set of actions an author needs to create and modify a movie on-the-fly, and the gestures that give access to these actions;
– extend the model to allow arbitrary tools to operate on movie content; and
– design a concrete architecture for implementing such a system.

2 Background and Use Cases

We identify two roles involved in communicating a proof script: the **proof author** and a **proof reviewer** (the reader). The proof author is a user of a PA that creates a (formal) proof. The interaction with the PA is encoded in a **proof script**, which contains the commands used to build the proof. The author can communicate the proof to any reviewer by publishing this script, and the reviewer can look at the script by loading it in his local version of the PA.

These two roles both have their own well-defined activities, but an actor (an actual user of a system, instead of just an abstract role) can play both the author-role and the reviewer-role: when writing a proof, an author might want to review what he has done before. To make the activities of both roles explicit, we identify a single use case for each of them. These use cases are *creating a proof* for the author (Figure 1), and *reading a proof* for the reviewer (Figure 2).

A third use case is *browsing a library*, in which a user takes on a role similar to that of the reviewer, but also searches for useful lemmas in the library, and tries to understand how they are used. While we do not treat this use case here (we are grateful to one of our referees for drawing attention to it), we nevertheless claim that such "advanced" elaborations of the reviewer's use case would benefit from the proviola technology presented here. Current technology does not give users fast access to other developments, which is a prerequisite for this use case.

Legend The diagrams below are almost, but not quite, standard UML:

- A stick figure represent a user role (proof author and proof reviewer).
- A "package" represents a tool/program instance (here: the PAs).
- The cloud represents the Internet.
- A folded page is a file (here: the proof script).
- Arrows represent data flow; a double arrow, interaction between two parts.

First Use Case: Creating a proof To create a proof, an author writes commands to be interpreted by the PA. In response, the state of a proof changes by decomposing the theorem to be proved, generating new proof obligations or discharging goals as proven. A proof script stores a a transcript of the commands issued.

In Figure 1, we display the traditional implementation of this use case, in which the PA is locally installed, and creates a local copy of the proof script.

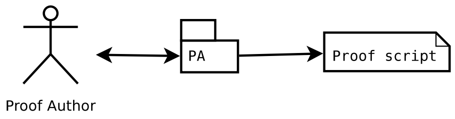


Fig. 1. Creating a proof script

Second Use Case: Reading a proof To read a proof, the reviewer obtains a (copy of a) proof script, possibly via the Internet. Subsequently, he can load it in his copy of the PA, and "replay" the proof: many PA interfaces have a notion of stepping through a script, by sending commands one-by-one to the PA or by undoing the last command sent. Because the PA does not know that the commands it receives are extracted from a script, it responds to commands in the same way as in the creation use case: by sending the new proof state.

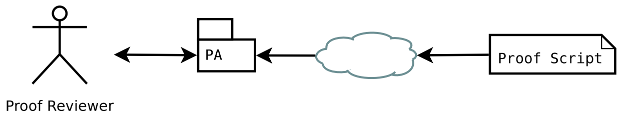


Fig. 2. Reading a proof script

Discussion. This form of communication has a number of problems:

1. If only a small part of the script is relevant, it might still be necessary to send the entire script to the reviewer: the parts of interest might require definitions defined previously in a script, or might use lemmas proved earlier.
2. Before a proof can be reviewed, the reviewer needs to install (ideally the same version of) the PA the author used, or be so familiar with its technology to interpret the script mentally[1]. Especially when the script is used to

[1] For a PA that uses a procedural proof style (tactics), it is hopeless to try to interpret a proof script purely mentally, without seeing it executed.

communicate a proof to a reviewer who is not part of the PA community, this can be a large handicap.

A possible solution to the first problem, frequently exercised on the Coq-club mailing list [2], is to simplify a proof script to a minimal example, which focuses on the problem in the script, and the definitions directly necessary to obtain this problem. But such simplification might be too drastic, abstracting away crucial details. Furthermore, abstraction is not an option when the main purpose of the communication is not to point out a problem, but to display a (partial) formalization of a proof to an outsider, because it is necessary to stay close to the vocabulary and methods of the target audience.

The ProofWeb system developed by Kaliszyk [7] places the PA on a central server that is accessible through an AJAX-based web application. This means that to review a proof, a reviewer needs only a web browser and the proof script, which could be hosted on the same server as the PA.

ProofWeb is an Internet-mediated realisation of the "creation" use case, which mitigates the second problem of having to install and configure a PA, but does not yet allow partial communication of the proof to a reviewer. It does not provide fast access to an arbitrary proof state: to obtain the state after a given command, all preceding ones need to be resent to the PA for reprocessing.

3 The Proof Movie

In the previous section, we identified two problems with the current method of distributing a proof script from a proof author to a proof reviewer. In summary, these problems are that reviewing requires the proof reviewer to resubmit the proof script to the PA, and that the proof script cannot be reviewed in fragments.

To solve these problems, we propose to enrich the proof script data structure. In the new data structure, which we call the **proof movie**, we record the commands sent to the PA coupled together with the response of the PA to each command. Such a pair of a command and a response is a **frame**. The exact Document Type Definition for the movie data structure can be found at `http://mws.cs.ru.nl/proviola/movies/film.dtd`.

The proof movie is designed to be *self-contained* and *generic*:

Self-contained. Making the movie self-contained means that a proof reviewer only needs a movie and a tool capable of displaying it to replay the proof: no other tools are necessary for this. Aside from this, the frames themselves contain the exact state of the proof at the point represented, meaning that it is possible for an author to publish a proof partially, omitting or reordering frames before publication.

Generic. By making the movie generic, creation and display do not depend on a specific PA or PA version: this does require that one can specify a transformation from the PA's interaction model generating discrete frames.

To implement the movie data structure, we used the eXtensible Markup Language (XML). An example frame in this implementation can be found in

```
<frame frameNumber="2">
  <command>
    intros A x.
  </command>
  <response>
    1 subgoal

    A : Type
    x : A
    ============================
    A
  </response>
</frame>
```

Fig. 3. An example frame of a Coq movie

Figure 3. This example contains the notion of a "frame number": a sequence number identifying the order in which the commands were submitted to the PA.

We do not consider the movie to be a complete replacement for a script. Instead, it is a container of a part of the script, together with the output of the PA. This output does not need to be correct, but this does not interfere with our intention of the movie: we see a movie as an explanation of a proof, not as checked proof script *per se*. If a movie contains a complete script, the concatenation of all the command segments of all frames in the movie reproduces the script, which can then be (re-)checked by a PA.

The movie introduces a new use case, *creating a movie*, which we describe in Section 5. Having the movie also changes the use case of reviewing a proof script, and we describe this modified use case first, in Section 4.

4 The Proviola: Watching a Proof Movie

When the reviewer has obtained a proof movie, he wants to access the data within it to review the proof, much like when he obtained a proof script. In effect, the "reading a script" use case described in Section 2 and illustrated in Figure 2 has not changed, only the data structure supporting it has changed.

We call the system used for displaying a movie a **proviola**. Just as in film-making, where an editor might wish to review a film while editing, and moreover quickly fast-forward and rewind the movie to see individual shots, we wish to achieve similar access speed and portability. Indeed, our proviola is not a separate tool: rather, we realize it through the use of HTML and very simple JavaScript.

Reading a proof script, revisited: watching the movie. The movie is self-contained, and can be distributed like a script. Unlike a script, the contents of a movie can be inspected without any external tools except a web browser: the movie can be located anywhere, and inspected from this location. In particular, a PA is not required to compute the proof's state and the movie can be

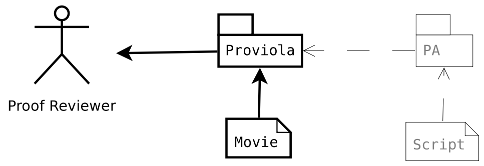
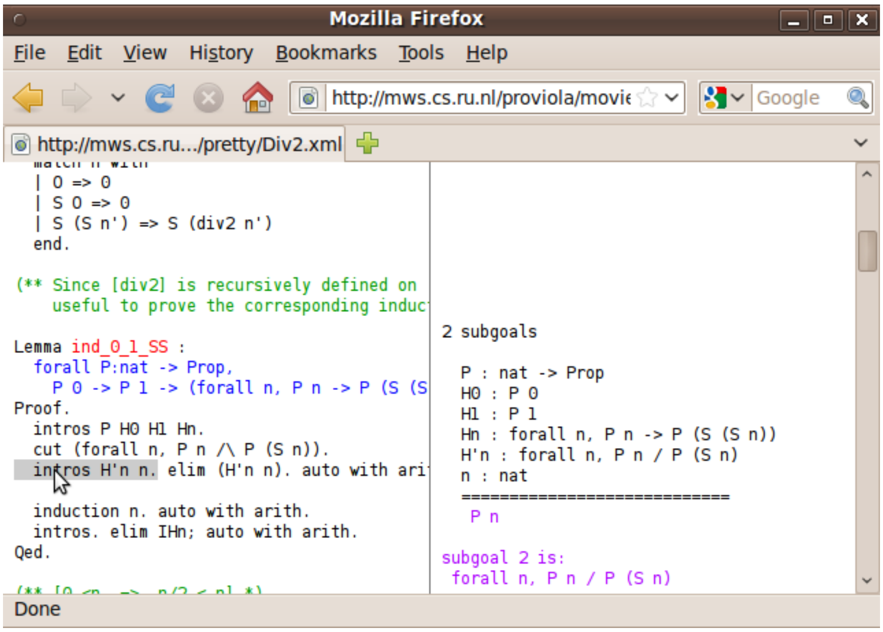


Fig. 4. Watching a movie: proviola and movie proxy PA behaviour

watched offline, at any time. After the reviewer loads a movie in the proviola, he wants to step through the proof much like when a PA was loaded with a script: by indicating for which command he wants to see the response.

As illustrated in Figure 4, the proviola and the movie together **proxy** [5, Chapter 5] the behaviour of a PA: the responses shown to the proof reviewer are stored (or cached) in the movie, after having been computed by the PA.

A prototype implementation of the proviola. We implemented a prototype of the proviola as an XSLT transformation from the movie into an HTML file containing embedded JavaScript. This page initially shows the commands within the movie, much like a proof script. Figure 5 illustrates this: when the reviewer places his cursor over a command, the corresponding response is revealed dynamically. The command pointed to is highlighted as a visual reminder. The prototype proviola can be inspected at `http://mws.cs.ru.nl/proviola/movies/movies.html`


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://mws.cs.ru.nl/proviola/movie
Google
http://mws.cs.ru.../pretty/Div2.xml
Done

```
  | 0 => 0
  | S 0 => 0
  | S (S n') => S (div2 n')
  end.

(** Since [div2] is recursively defined on
    useful to prove the corresponding induc

Lemma ind_0_1_SS :
  forall P:nat -> Prop,
    P 0 -> P 1 -> (forall n, P n -> P (S (S
Proof.
  intros P H0 H1 Hn.
  cut (forall n, P n /\ P (S n)).
  intros H'n n. elim (H'n n). auto with ari

  induction n. auto with arith.
  intros. elim IHn; auto with arith.
Qed.
```

```
2 subgoals

  P : nat -> Prop
  H0 : P 0
  H1 : P 1
  Hn : forall n, P n -> P (S (S n))
  H'n : forall n, P n / P (S n)
  n : nat
  ============================
   P n

subgoal 2 is:
 forall n, P n / P (S n)
```

Fig. 5. Screenshot of a proviola

5 The Camera: Creating a Proof Movie

Before a movie can be replayed, it must be created from a proof script by a tool we call a **camera**. Such *creation of a movie* is a new use case, shown in Figure 6.

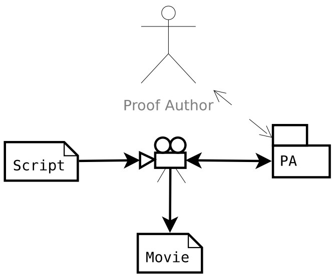


Fig. 6. Creating a movie: camera and script proxy user behaviour

Third Use Case: Creating a movie. This is non-interactive: the user of the camera invokes it on a script, after which the tool does all the work, yielding a movie.

After it has been invoked, the camera parses the proof script given to it into separate commands. These commands are stored in a frame and sent to the PA. When the PA responds to a command, the response is recorded alongside the command. The frame is subsequently appended to the movie.

From the perspective of the PA, the camera and the script behave like an actual proof author. In other words: in creating the movie, the camera and script together *proxy* the behaviour of a proof author.

Because the author holds the original script, it seems natural that she invokes the camera on it to obtain a movie, for distribution to reviewers. However, a reviewer might also play the role of cameraman, given access to the script.

Prototype implementation. We implemented the camera as a client to the ProofWeb system, available at `http://mws.cs.ru.nl/proviola/camera/camera.html`. Although it is possible for the camera to communicate directly with each PA, we believe that using a generic wrapper like ProofWeb has the following advantage: ProofWeb provides a generic interface to different PAs: the communication protocol is the same for each PA, and the only PA-specific knowledge the camera needs to have is how to split a proof script into separate commands.

The main disadvantage to this approach is that to support new PAs with the camera, these need to be made compatible with ProofWeb, which imposes stricter requirements on the interaction model than the movie design needs. In our design, the camera behaves as a straightforward client to a ProofWeb server, wrapping the commands in the annotation expected by that server and unwrapping the responses. A step beyond this, which we investigate in Section 6, is creating a movie automatically and on-demand.

6 Proxying Movie-Making

Until now, we have created proof movies by submitting a complete script to a PA. In particular, we have filmed the Coq standard library [3]. We now wish to go beyond this simple case of filming completed scripts, and investigate how to create a movie dynamically, by observing the interaction between proof author and the PA. Based on these observations, we redesign this architecture to support the desired behaviour. This architecture will be implemented in future work.

As we have mentioned in Section 2, a user taking the role of a proof author can also take on that of proof reviewer. If she takes on these two roles simultaneously, we get the situation depicted in Figure 7. This figure is constructed by composing Figures 4 and 6, replacing the proxied components from each figure with their implemented counterpart in the other figure.

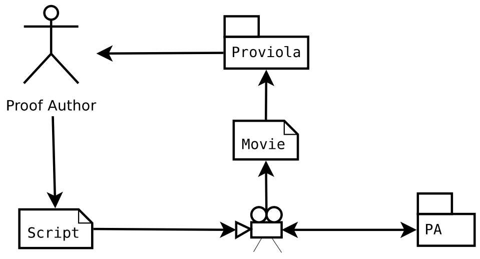


Fig. 7. Interactively creating a movie: instantiating the proxies

In this figure, the proof author writes commands into a script, which is submitted to the camera command-by-command. The command is then handled by the camera as described in Section 5. Because the proof author has the movie corresponding to the script loaded in the proviola, it updates whenever the script updates or when the PA responds to a command.

The camera as designed in the previous section requires an explicit action by the creator of the movie: the camera is a tool that needs to be invoked on a proof script to create a movie from it. Such a design requires the proof author to constantly update the movie when updating the script, to keep them in sync. In fact, if we follow the information flow in Figure 7, the author cannot even see the result of her own changes to the script if she does not use the camera first.

As a solution for this disadvantage we merge the script and movie into a single data structure, that is manipulated by both the proof author (through the proviola) and the PA. To obtain this, we will need to modify the movie, the camera and the proviola in the following ways:

Movie. The movie does not need to change in any radical way. The only change necessary is that a movie be editable after it has been created. This way, the proof author can write commands into the movie as she would do into the script.

Proviola. The proviola already provides a display of the movie, giving the proof reviewer access to the script and the proof state at that point. To allow an author to update the movie, the proviola needs to be extended with an interface to update the commands in the frames in the movie. This extension is done by adding the notion of a **focused frame**, which can currently be edited. To manipulate this frame, we add gestures for the following actions to the proviola:

Create a movie. This action creates a new, empty movie.
Focus frame. The proof author can use this action to change the focused frame to be the frame she is interested in editing.
Edit frame. The only frame that can be edited is the focused frame.
Add frame to a movie. When the author finishes a command in the focused frame, a new frame is tacitly added to the movie and given focus. The previously focused frame is submitted to the camera for further processing.
Remove frame from a movie. This is an inverse action to adding a frame.

We consider submitting a frame for further processing an implicit action: the proof author does not indicate in any way that she is finished with a frame, but the system recognizes a frame to be complete and processes it further, including rechecking commands later in the movie, if these depend on the frame that was changed. What "depends on" actually means depends on the script-management model (and more generally: the interaction model) of the PA used.

Camera. To keep the proviola as lightweight as possible, the PA's manipulation of the movie is brokered through the camera. This includes periodically checking whether the author completed a command in the focused frame. Frames containing completed commands are then split off the focused frame and submitted to the PA. This means the camera evolves from batch-processing a script (as in Section 5) to continuously reading the movie, updating its contents as needed.

By merging script and movie, implementing the changes above, we obtain the architecture shown in Figure 8.

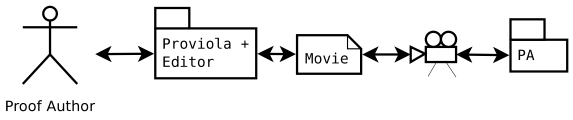


Fig. 8. Interactively creating a movie: cutting out the script

Design and implementation of the system. We have embarked upon a prototype implementation of the architecture described above. We decided to base it on ProofWeb's architecture, placing the camera on a server, with the proviola as a client to the camera, as an interface to the services offered at the server.

The movie is kept both as a local copy by the proviola and as a remote copy by the camera, and the protocol between camera and proviola is meant to keep these copies synchronized: each action of the author is executed on the local copy and then communicated to the camera, which communicates the change to the PA. As such, the movie becomes a *proxy* for the PA's state, and the interaction of proof author with PA is proxied by editing the movie.

By being placed between client and PA, the camera plays a similar role as the Broker in the PG Kit framework [1]. The main differences between that system and our proviola-based system, are based on the design decision to make the movie the main entity of the architecture. This has the following implications:

Cached history. The history of the interaction with the PA is stored as a movie. Because the client has a local copy of the movie, it provides instant access to the history of the proof development.

Implicit proof navigation. The author can freely choose the frames of the movie to edit instead of having to request the PA to unlock or process a line.

To synchronise the movie between camera and proviola, we follow a version of the *publisher/subscriber* [5, Chapter 4] design pattern:

– The proviola holds the local copy of the movie, and records when a user focuses on a specific frame.
– When a frame's content changes, for example, when a PA responds to a command, the camera notifies the proviola of this.
– If this frame has not been loaded before, or has been updated, it is requested from the camera, and cached locally.
– If the frame has been requested before, and has not been updated, it is loaded from cache.
– When the user updates the focused frame, it is sent to the camera.

7 Applications

7.1 The Camera as a Service Broker

In our design of interactive movie-making, the camera takes a centralized place at the server, containing the "master copy" of the movie and providing write access to it. This access is not only usable by the proviola or the PA, but could also be used by other tools that work with a formal proof.

To provide access to the movie, the camera needs to allow arbitrary tools to register as either a *subscriber* or as a *publisher* for a movie.

Subscribers. A **subscriber** to a movie can obtain the frames of the movie in which they are interested. When a frame is changed in the camera-side movie, it notifies each subscriber of the changed frame. Subscribers can then update the frame. Our intention is that subscriber access be read-only.

Publishers. By contrast, when registered as a **publisher**, a tool can write in the movie. We do not believe there should be any restriction to the types of content that publishers can write, but do require that the movie should be *extended*: if the tool produces information of a different type than already exists in the movie, it should be added alongside the existing data, not replace it. So, although it is possible for multiple tools to write into the command section of a frame, for example a tool processing a script and the proviola+editor of Section 6, a tool that does not produce commands should not write in the command section.

The idea of tools listening for changes in proof state is not new: it was previously mentioned by Aspinall, Lüth and Winterstein [1] as future work for their broker architecture. Using the camera as a broker is similar, with the advantage that history does not need to be kept by the individual tools but can be requested from the camera: an additional benefit is that subscribers coming late to the party still have access to the full history of a proof, which we consider especially interesting for our ongoing work towards a repository of formal proofs.

7.2 Towards an Encyclopedia of Formal Proof

With some modifications, the proof movie can be used as the data structure underlying an encyclopedia that we envisage containing formal proofs together with an informal narrative explanation, and provide a toolbox for using and manipulating such composite "articles", as originally sketched in [4].

The movie provides a generic structural view of different kinds of script, so that for a tool wanting to manipulate a proof on a structural level, it is not necessary to know the exact details of the PA concerned, and the history contained in the movie implies that tools developed later are not required to replay the entire script through a PA. Furthermore, because the camera provides a central interface for accessing the movie, tools do not have to be on the server hosting the encyclopedia, allowing others to use the proofs without undue restrictions.

To implement the encyclopedia, we do need to create several tools ourselves, that together form the backbone of the encyclopedia.

First, we need to store and retrieve the movie. Possible candidates for storage are the file system, a database or a version control system which keeps track of the history of a proof. Retrieval cannot easily be expressed by communicating with the camera, and will most likely warrant an extension to that concept.

The second addition is aimed at adding informal explanation to a proof: a "commentary track", where an author can comment on the proof script and the output that the PA produces. This requires a revision of the movie-concept. Because several frames in a movie might be explained in a single, continuous narrative and commands might be repeated several times in the narrative, we will add *scenes* to the movie. A **scene** contains a single, informal description of a group of frames, and a reference to these frames. A scene can refer to zero or more frames, and a frame may belong to zero or more scenes.

As well as linear narrative, scenes could be used to store alternative problem-solving approaches, such as using different lemmas or automated proof search.

As previously mentioned, we have investigated the movie's ability to capture a large body of proofs by filming the Coq standard library [3]. The filming took less than one hour, including a two-second sleep between processing each file, which was inserted to prevent overloading the server running ProofWeb. The resulting films can be inspected at `http://mws.cs.ru.nl/proviola`, which serves the films quickly, even for large developments such as that of the Riemann integral, at around two MB in size. Movies typically are much smaller, however, up to five hundred kB. The movie size scales in the size of the original script: a long script that uses tactics producing many subgoals also creates a large movie.

In further work [12], we elaborate the concepts of scenes and commentary, and describe tooling for creation and display of a commented movie, by investigating movies of course notes on Software Foundations by Pierce *et al.*[11], available at `http://mws.cs.ru.nl/proviola/movies/sf`

8 Related Work

We have already mentioned the PG kit system and its protocol, PGIP [1]. Interactive movie-making is similar to its broker architecture. While PGIP focuses on the message structure for the dynamics of communication between author and PA, a movie freezes a sequence of such messages into a static data structure. In that respect, our approach is orthogonal to that of PGIP: the movie stores the result of an interactive session while PGIP deals with the messages and protocol needed to generate it. Instead of a prototype based on the camera and ProofWeb, we could just as well have filmed an interaction with the PG kit broker.

Wenzel's Jedit editor [13] also considers a finer grained analysis between editor and Isabelle. But this is more oriented towards parallelisation of proof steps than the human-orientated details of interaction and representation of proofs considered here.

The movie file format is an XML-based representation of formal mathematical documents. The best-known and most versatile XML-based format for mathematical documents is OMDoc [9]. We decided not to use this format internally in our system. OMDoc is about document *structure*, while the content of a movie is unstructured system input/output. Still, if needed, movies could be easily converted to OMDoc, so our choice of a simpler format is not a limitation.

The history stored in a movie is not the same as history in, say, a web browser: the movie stores both the messages sent and the responses received, while a browser only stores a reference to pages visited, and needs to obtain the page again when the user requests it. This approach of "recalculating" a result is similar to what happens when a user sends undo commands, both when using a normal program (such as a text editor) and when using a PA. In our proviola there is no recalculation, because all the "history" is stored in the movie.

A command in a PA can consist of a combination of more primitive tactics, for example two tactics sequentially composed with the ";" tactical in Coq. One might wish to see the intermediate state after the first tactic in such a sequence. Coq does not support this small step execution model, so in our implementation, the movie cannot provide this information. A system like Matita [10] does, so a proviola based on Matita could potentially expose this refined execution.

Integrating a "commentary track" into the movie, by collecting frames into "scenes" and adding a narrative text to it, goes in the direction of earlier work on tmEgg [6]. There we started with a mathematical document, written in the editor

TEXmacs, as the backbone with a Coq proof script (a `.v` file) underneath it. At any point one could consult the formal proof by opening a Coq session within the document and executing Coq commands. This means Coq is started up and brought into the required state, and then the selected commands are executed. This works quite well for small scripts, but can take a lot of time for larger proofs. Also, it requires the PA to be available, with the right libraries, which makes the mathematical document less "self-contained". Another hindrance is that tmEgg relies heavily on TEXmacs, which means that one is dependent on yet another application to run smoothly. The present work allows movies to play the role of the Coq interaction (as a proxy) within an interactive mathematical document. This can basically be done within any editor and thus relieves the interactive mathematical document from being tied to either TEXmacs or Coq.

9 Conclusion

In conclusion, we claim that our refactored interaction model and its associated data structure are an important contribution in their own right. But our interest is in how this data structure may be further extended to support richer interaction and display as part of a MathWiki.

Acknowledgments. The idea of movie-making emerged from discussions with Dan Synek about the absence of a lightweight way to communicate a formal proof to others. We would like to thank the anonymous referees, whose questions and suggestions helped improve this paper. This research is partially funded by the NWO project "MathWiki: a Web-based Collaborative Authoring Environment for Formal Proofs", the NWO BRICKS/FOCUS project "ARPA: Advancing the Real use of Proof Assistants" and the NWO cluster DIAMANT.

References

1. Aspinall, D., Lüth, C., Winterstein, D.: A framework for interactive proof. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, pp. 161–175. Springer, Heidelberg (2007)
2. Coq-Club Mailing List: The Coq-Club mailing list. Mailing List, `http://logical.saclay.inria.fr/coq-puma/topics`
3. Coq Development Team, T.: The Coq standard library. Library documented on, `http://coq.inria.fr/stdlib` (obtained on March 5, 2010)
4. Corbineau, P., Kaliszyk, C.: Cooperative repositories for formal proofs. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, pp. 221–234. Springer, Heidelberg (2007), `http://www4.in.tum.de/~kaliszyk/docs/cek_p3.pdf`
5. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns – Elements of Reusable Object-Oriented Software, 1st edn. Addison-Wesley, Reading (1994)
6. Geuvers, H., Mamane, L.: A document-oriented Coq plugin for TeXmacs. In: Libbrecht, P. (ed.) MathUI Workshop, MKM 2006 Conference, Wokingham, UK (2006), `http://www.activemath.org/~paul/MathUI06/`

7. Kaliszyk, C.: Web interfaces for proof assistants. In: Autexier, S., Benzmüller, C. (eds.) Proceedings of UITP 2006, Seattle. ENTCS, vol. 174(2), pp. 49–61 (2007), `http://www4.in.tum.de/~kaliszyk/docs/cek_p2.pdf`
8. Kaliszyk, C.: Correctness and Availability. Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web. Ph.D. thesis, Radboud University Nijmegen (2009), `http://www4.in.tum.de/~kaliszyk/docs/ck_thesis_webdoc.pdf`
9. Kohlhase, M.: OMDoc – An Open Markup Format for Mathematical Documents (version 1.2). LNCS (LNAI), vol. 4180. Springer, Heidelberg (2006)
10. Matita Team: Matita interactive theorem prover. Web page, obtained from, `http://matita.cs.unibo.it/`
11. Pierce, B.C., Casinghino, C., Greenberg, M.: Software foundations. Course notes (2010), `http://www.cis.upenn.edu/~bcpierce/sf/`
12. Tankink, C., Geuvers, H., McKinna, J.: Narrating formal proof (work in progress). In: Submitted to UITP 2010 (2010), `http://cs.ru.nl/~carst/files/narration.pdf`
13. Wenzel, M.: Parallel proof checking in Isabelle/Isar. In: Reis, G.D., Théry, L. (eds.) PLMMS 2009. ACM, Munich (2009), `http://www4.in.tum.de/~wenzelm/papers/parallel-isabelle.pdf`

A Wiki for Mizar: Motivation, Considerations, and Initial Prototype

Josef Urban[1,*], Jesse Alama[2,**], Piotr Rudnicki[3,***], and Herman Geuvers[1]

[1] Radboud University, Nijmegen, The Netherlands
[2] New University of Lisbon, Portugal, Portugal
[3] University of Alberta, Edmonton, Canada

Abstract. Formal mathematics has so far not taken full advantage of ideas from collaborative tools such as wikis and distributed version control systems (DVCS). We argue that the field could profit from such tools, serving both newcomers and experts alike. We describe a preliminary system for such collaborative development based on the Git DVCS. We focus, initially, on the Mizar system and its library of formalized mathematics.

1 Introduction and Motivation

Formal mathematics is becoming increasingly well-known, used, and experimented with [9,10]. Verification of major mathematical theorems such as the Kepler Conjecture [8], the Four Color Theorem [7], and the increasing use of verification for critical software and hardware [11,5] are pushing the development of interactive verification tools. Indeed, there are already various online repositories of formal proofs [1,15,4] and catalogs of deductive tools [21].

The goal of the work presented here is to make formal mathematics *accessible online* to interested parties by making the subject widely available through online tools and interfaces. We are particularly interested in providing fast *server-based* tools for verification, web presentation, and collaborative development of formal mathematics.

We believe that such tools are important for our field to attract newcomers: they provide an attractive environment for exploring the world of formal reasoning and the tools of the trade. The technology we have in mind is vital also for existing members of the formal mathematics community simply by making systems for formal mathematics easier to access and use. Ideally, the system should

* Supported by the NWO project *MathWiki: A Web-based Collaborative Authoring Environment for Formal Proofs.*

** Partially supported by the European Science Foundation research project *Dialogical Foundations of Semantics* within the ESF EUROCORES program *LogICCC: Logic in the Humanities, Social and Computational Sciences* (funded by the Portuguese Science Foundation, FCT LogICCC/0001/2007).

*** Partially supported by NSERC.

be so straightforward that, to satisfy some momentary curiosity about a formalism or a proof or a theorem, one could just visit a web page instead of suffering through the installation of a new system. In the long run, we foresee a web-based repository of mathematics realizing the vision of the QED Manifesto [20].

Our effort has three principal features: it

– is based on the notion of a wiki (understood here as a support for distributed, web-based collaborative project)
– uses distributed version control system(s), and
– uses server-based software to do the "heavy lifting" of verification and proof checking.

Let us briefly characterize these features of our approach.

The first main feature of our approach is the use of wikis. Wikis are well known; they offer collaborative web-based information resources. The original wiki [2] is nearly 15 years old. This new genre of web site has grown enormously. Wikipedia is, evidently, the most prominent example of a wiki: even casual Internet users are not only aware of Wikipedia but use it often: links to Wikipedia pages are often among the top results of web searches.

Mathematics is, thankfully, no exception to this trend. Wikis and other online repositories and communities for mathematics abound: arXiv, MathOverflow [12], T. Gowers' PolyMath [18], Wolfram MathWorld [13], PlanetMath [17], ProofWiki [19], etc.[1] These constitute just a sample of mathematics' web presence; it does not seem feasible to give a complete survey of the subject. Yet, at present there are no mathematical web sites for *formal* mathematics that permit collaborative editing and other online tools (though we know of one prototypical effort [3]). We aim to fill this gap.

The second main feature of our approach is the use of distributed version control systems (DVCS). Such systems are eminently suitable for large collaborative efforts: they allow one to maintain repositories of texts or code in the presence of multiple people working independently or cooperatively. DVCSs are becoming more widely used, both in the commercial sector, in academia, and in the free software community. Our approach is novel because the application of DVCSs to maintaining large bodies of formal mathematical proofs—objects that are both computer code and human-readable text—is largely underdeveloped.

The third and final main feature of our approach is the use of server-based tools. Like DVCSs and wikis, such systems are becoming widely used. The general reason for this is that the Internet is becoming ubiquitous, faster, and more reliable. Server-based approaches can arguably help to attract newcomers to the subject because it spares them the burden of locally installing formal mathematics software and its associated libraries. Moreover, computations that one might want to carry out are sometimes large and should (or must) be carried out on servers rather than less powerful client hardware. In our case, proof checking and the generation of rich semantic HTML presentations of formal proofs is, often, quite expensive.

[1] Note that majority of these on-line services are non-profit efforts.

The rest of the paper is organized as follows:

- Section 2 briefly describes the primary applications of a wiki for formal mathematics.
- Section 3 lists the essential features of DVCSs, and how we use them to provide robust and flexible back-end for our formal mathematical wiki.
- Section 4 briefly discusses the current methods for developing the Mizar Mathematical Library (MML) and identifies the main bottlenecks of the current model for massive distributed formalization efforts.
- In Section 5 we discuss the features that a formal mathematical wiki should have and the requirements it should satisfy; we focus on the problem of maintaining the correctness of a formal library.
- The initial implementation of the wiki for Mizar, based on the Git DVCS, is given in Section 6. We explain the Git repository structure, the communication and division of work between the different repositories and branches, how we use Git hooks for guarding and updating the repositories, how we extract and recompute dependencies from the changed library and its re-verification and regeneration of its HTML presentation.
- Possible future extensions are discussed in Section 7.

2 Use Cases

We intend to provide tools for various collaborative tasks in the formal mathematics communities. Our efforts should also aim to include the whole mathematical community and be beneficial to it: formal mathematics is a natural extension of traditional mathematics in which the gaps, metaphors, and inaccuracies are made "computer-understandable" and thus, in principle, mechanically verifiable. The major use cases that we have in mind are

- public browsing of a human-readable web presentation of computer-verified mathematics;
- facilitating the entrance to the formal mathematics community;
- library refactoring (from small rearrangements of individual items and minor changes, to major overhauls);
- authoring contributions, both large and small;
- supporting the design of a formalization structure (concept formation, fleshing out definitions, bookkeeping postulates and conjectures);
- offering tools to get help with simple proof steps;
- gradually formalizing informal texts;
- translating contributions between/among proof assistants, archives, and natural languages;
- merging independently developed libraries.

In this paper, we narrow our focus to examine how some of the above tasks apply to the Mizar system. Thus, we are interested in the aspects of the above problem that arise when working within a single, fixed formal framework. However, we keep in mind the rather grand scale of the issues at hand.

3 Towards Distributed Collaboration

One of the most exciting but apparently unexplored topics in formal mathematics is the use of distributed version control systems (DVCSs). Such systems support tracking the development of work among authors whose paths can be non-linear, proceeding down a variety of routes/ideas that may or may not converge upon a final, conventionally agreed-upon goal.

When thinking about DVCSs, a potential misunderstanding can arise that we should correct. One might get the impression that DVCSs simply encourage chaos: without a central repository, who is to say what is correct and what is not? And would not DVCSs lead to a fragmented community and wasted labor?

Although we can conceive such dystopian possibilities, we prefer another point of view. We want to emphasize the *organized* in *organized chaos*, rather than the troubling noun. It is helpful to think of DVCSs not as a source of chaos, but rather as a tool for putting some structure on the distributed efforts of a group of people sharing a common interest. In practice, DVCSs are used to organize the efforts of many people working on crucial projects, such as the Linux kernel[2]. Although the DVCS model does not require a central, standard repository to which everyone refers, there often are strong conventions that prevent the disarray and confusion that can come to mind when one first hears about DVCSs. In the case of the Linux kernel, for example, the entire community practically revolves around the efforts of a single person—Linus Torvalds—and a rather small group of highly trusted programmers. A fairly wide number of people still contribute to the Linux kernel, but in the end, essentially all proposed contributions, if they are accepted at all, pass through the core developers. At least for the foreseeable future we propose to follow the same approach: a handful of experienced Mizarians[3] will decide what constitutes the current official release of the distributively developed system.

Foremost, we need to ensure that the current practices of developing the Mizar Mathematical Library (MML), which evolved over a number of years, can still be supported. Indeed, the core Mizar developers can easily continue their current practices using DVCSs. We would like to switch to distributed development in an incremental fashion and we foresee deep involvement of the current Mizar developers while switching to the new technology.

For our first pass at a formal mathematics collaborative tool, we have chosen the Git system [6]. Git is originally developed by Linus Torvalds, the original author and primary maintainer of the Linux kernel, for working with the large number of contributions made to the Linux kernel. Git enjoys widespread use in various software communities (commercial, academic, and open-source).

Our choice of Git is, to some extent, arbitrary. A number of widely used DVCSs are available (monotone, bzr, arch, mercurial) that could support the collaborative tasks we envision for a formal mathematics wiki. (Indeed, one project with aims

[2] Just to give some feeling for the size, the compressed sources of the Linux kernel are roughly 53 MB; the compressed sources of the proofs in the Mizar Mathematical Library are 14 MB.

[3] We thank Yuji Sakai for stressing the charm of this name.

similar to ours–vdash [25]—has proposed to use monotone.) We cannot, therefore, robustly defend our choice of Git except to say that *some* choice of DVCS must be made, and any other choice of DVCS would likely be just as arbitrary. We choose Git because of our familiarity with it, and its wide usage in many projects that we know of.

A full description of Git can be found on its homepage [6] and the system can be learned through a number of tutorials prepared by the Git community. We believe that Git's concepts and operations are a good match for the kinds of collaborative tasks that a formal mathematics wiki should support. Here we limit ourselves to a skeletal presentation of Git's main features.

Like other version control systems, the Git system is based on the notion of a *repository*, a structured collection of objects (texts, computer code, documentation, etc.) being worked on by members of a team. As a developer makes changes to the repository, a sequence of *commits* are made. The sequence can split at various stages, as when the developer or the community wish to take on a subproject or otherwise pursue some line of investigation. The full history of a repository can thus be more aptly understood as a tree rather than a simple sequence of changes.

A single developer can either start a new project afresh, or can *clone* the repository of some other developer working on the project of interest. Unlike traditional non-distributed version control systems such as rcs, cvs and subversion, Git repositories (and those of some other DVCSs, too) are "complete" in the sense that the developer has essentially unlimited access, even when offline, to the entire history of a repository. When online, the developer can share his changes with others by *pushing* up his changes to another repository (or repositories), and he can stay connected to his colleagues by *pulling* down their changes.

The Git system also provides *hooks* that can be used to implement various guards and notifications during the various repository actions (*commit*, *push*, etc.). This can be used to allow only certain kind of changes in the repository. For example, our initial wiki prototype for Mizar (See Section 6) makes use of suitable hooks to ensure that the updated articles and the whole updated library repository are always in a correct state.

In our initial prototype, we introduce a publicly accessible, central repository—in the spirit of Wikipedia or the main Linux kernel branch—that serves as a correct and verified snapshot of the current MML. Our tools are targeted at public editing of the repository, while ensuring the library's coherence[4] and correctness as the changes are made.

4 A Special Case: The Mizar Developers

Among the first targets for our implementation are the core Mizar developers. It will be worthwhile, then, to explain how Mizar is currently developed.

[4] Coherence of MML is closely related to formal correctness and is a narrower notion than integrity of MML, cf. [22].

The history of the Mizar project is briefly presented in [14] and a brief overview of the current state of the project can be found in [16].

The development of the Mizar Mathematical Library (MML), the principal, authoritative collection of Mizar contributions (called "articles"), has been the main activity of the Mizar project[5] since the late 1980s, as it has been believed within the project team that only substantial experience may help in improving (actually building) the system. An article is presented as a text-file and contains theorems and definitions. (At the time of writing, there are 1073 articles in MML, containing 49548 facts/theorems and 9487 definitions.) The articles of MML—in source form—are approximately 81MB of data.

The current development of MML is steered by the Association of Mizar Users (in Polish: Stowarzyszenie Użytkowników Mizara, abbreviated SUM), which owns the copyright to the library. SUM appoints the MML Library Committee—the body responsible for accepting articles, arranging reviews, and maintaining official releases of the library. MML is organized in an old-fashioned way, resembling printed repositories of mathematical papers. Since MML articles exist electronically, it is relatively easy to re-factor the MML contents by, say, changing items such as definitions and theorems, or by deleting, or by moving them. After such modifications, the Mizar processor is run on the entire MML to verify its coherence. Currently, the Library Committee does the greatest amount of refactoring of MML, which includes the following activities:

– updating existing articles to take advantage of changes in the Mizar language, the Mizar processor, and changes in MML;
– moving library items where they more appropriate locations;
– building an encyclopedia of mathematics in Mizar which consists of creating new articles by moving topically related items from all over the MML into one place;
– generalizing definitions and theorems;
– removing redundant items; and
– merging different formalizations of the same material.

This process is under control of the Head of the Library Committee; about three people do most of the revisions. It can also happen that the original Mizar authors re-factor their own past contributions after they notice a better way to formalize the material. The remaining MML revisions are usually suggested by posting to the Mizar mailing lists, or mailing directly to someone on the Library Committee.

Information about what each of the MML refactorers is doing is not broadcast in any widely accessible way. The typical method is through an email announcement: "I am now formalizing XYZ, so please wait for my changes". Such announcements are frequently accompanied by a solicitation for remarks and discussion.

Such a process for collaboratively editing MML can be problematic and is far from ideal. The main problem with the present method is that it does not scale up: although it can be used with some success among few people, we imagine a

[5] http://mizar.org

much larger community where the collaboration problem surely cannot be effectively solved by the present approach. Moreover, new users face an unpleasant experience of developing a new article (or a group of interrelated articles) and, before finishing their work, they learn of a newer, substantially different official MML version that is (partly) incompatible with their working articles. Updating such an unfinished development can be quite time-consuming (but the Library Committee offers help in such cases).

5 Formal Wiki Features and Issues

The experience of Mizar authors is that while developing some new formalizations they notice that many MML items can be improved by, say, weakening the premise of an existing theorem or widening the type of a parameter in a definition. Ideally, an author should be able to introduce such small changes to the repository instead of asking the Library Committee and waiting for a new version of MMLto be released. With a DVCS such small changes can be incorporated into a daily build as there are mechanical means for verifying the correctness of such edits.

For the benefit of users, there must be something like an official MML version. We believe that with the DVCS development model, such official versions can be produced less frequently, subsequent official versions could differ more substantially, and all differences could be documented.

We foresee the following goals when treating MML as a wiki:

– content *correctness*;
– *incremental* editing; and
– *unified presentation* of the library for browsing.

Finding the right trade-off among these goals is not trivial. Allowing incremental editing of a formal text can lead to its incorrectness. On the other hand incrementality is an important feature of wikis. Thus the formal wiki should be able to save one's work even if it is a major rewrite of the existing code, and not completely correct. But the library is also a source of information for other users/readers and its current stable/correct version should be well presented. This means, however, that all kinds of unfinished work (active works-in-progress, abandoned attempts) should be reasonably hidden from the readers who just came looking for information.

Thus it seems that we have to combine the methods used for distributed software development—allowing many people to work on different parts, possibly breaking things, and easily accessing and merging each other's work–with the methods and mechanisms used for development of informal wikis like Wikipedia, where the content is primarily intended for human reading, and things cannot be "formally broken" too much (obviously, large projects like Wikipedia do have some automated structural checks, but evidently most of the review is done by humans). As of now we are only collecting experience on how to build and maintain a wiki of contents with formally defined semantics and correctness. It is not clear to what degree issues raised in designs of less formal repositories are relevant to our task.

5.1 Degrees of Formal Correctness and Coherence

The important difference between text-based wikis like Wikipedia and formal libraries like MML is the strong, well-defined notion of *formal correctness*. As regards MML, one can consider several aspects/scopes of formal correctness that are not available (not even in principle) for traditional wikis such as Wikipedia.

To begin, consider the notion of *link coherence* on, say, Wikipedia. Suppose someone edits only the body text of the Wikipedia entry for the Eiffel Tower. The article's internal coherence is unchanged, and the link coherence of Wikipedia as a whole is unaffected by the edit. However, other kinds of edits are permitted:

– if one deletes the Eiffel Tower article or renames it, then (part of) Wikipedia becomes link incoherent because some internal links become broken.
– if one deletes or renames a section of the Eiffel Tower article, then, as in the first case, it is quite possible that other Wikipedia articles become "tainted" by this edit by containing links to a subsection that no longer exists.

Internal link coherence can be enforced in various ways. Some methods are entirely computational (that is, can be carried out autonomously by a machine and require no human intervention); others depend in part on the (human) Wikipedia community. Further, this notion of coherence can be enforced by responding to potentially problematic edits, or by simply not giving users the means to make them. For example, the problem of deleting pages can be addressed by simply disallowing deletions. If such edits are allowed (perhaps only by certain users), an autonomous wikibot, constantly patrolling Wikipedia, can enforce internal link consistency by updating every Wikipedia article that refers to the Eiffel Tower article. This kind of bot-based approach could also respond to internal section renaming or deletion. Finally, internal link coherence can be enforced by Wikipedia's editor's tools, which give humans an environment for editing that does not straightforwardly permit such section renaming in the first place.

A wiki based on formal texts like Mizar articles permits one to define the notion of *change propagation*. When a user changes a piece of the Mizar library, we must take into account how this change affects other parts of MML. For example, if out of curiosity one changes the definition of the constant $\tau = (1 + \sqrt{5})/2$ to, say, $1/2$, then, obviously, this edit will have some effect on the library (some theorems about Fibonacci numbers that involve this classical ratio will be invalidated). Likewise, if one rearranges the theorems or definitions in an article, this too can affect parts of the library. On the other hand, some edits are safe: one can append new content to the end of existing articles without affecting the library's coherence.

The problem is that some modifications preserve the coherence of the wiki, but some do not. The typical task we face in maintaining the coherence of MML, considered as a wiki, is that a user has changed or added some articles to the library, and we want to verify that these changes are admissible, that is, that they maintain the coherence of MML. In extreme cases, checking the admissibility of a user's edit can, conceivably, require re-verifying of the entire library (for example, imagine changing the the fundamental set-theoretical axioms on which MML rests). The cost of such admissibility checks is correspondingly large. However, there are a number of more or less typical cases that can be dealt with less expensively. In the next section we explain how we have implemented these checks.

6 Prototype

Suitability of DVCSs and wikis
A natural object of interest when thinking about wikis and DVCSs together are wikis based on DVCS. According to a recent listing,[6] there are today more than a dozen wikis based on the Git DVCS or using Git as a back-end. Our design decision about wiki for Mizar is: the data (articles) in the Git repository should always be self-sufficient (independent of the wiki functionalities), and easily exchangable between various installations (that can be used just for local development, as a back-end for another wiki-like system, etc.). Wikis form a dynamic field of interest, so tight coupling the Mizar data to a specialized and possibly not widely adopted wiki storage implementation could cause difficulties for developing alternatives and for integrating formal mathematics into other, larger frameworks. All such alternatives should provide convenient communication at the DVCS (data) level with others. This seems to be a reasonable invariant for building all kinds of web sites and tools for creation of formal mathematics (and probably for collaborative creation of code in general).

ikiwiki is one of the most developed an probably best-known *wiki compilers*. Wiki compilers process a file or set of files written in a special (usually simplified) syntax and turn them into static HTML pages. It is possible to build an initial prototype of a wiki for Mizar by customizing ikiwiki. We have explored this path and are aware of many functionalities of ikiwiki useful for our task. We have also considered the peculiarities that make our domain different from the typical (one-page-at-a-time) paradigm of such wiki compilers, and decided to gradually re-use and plug-in the interesting ikiwiki functionalities, rather than trying to directly customize the large ikiwiki codebase to our nonstandard task.

Prototype Overview: As we discussed earlier, our prototype (see Figure 1) is initially focused on the Mizar developers who edit their work mainly offline and submit their changes to be viewed online[7] by other developers. The repository structure (and suggested future features) of our prototype is as follows:[8]

- There is a central repository on our server with the *stable* branch, and the *master* branch. This repository can be (using post-commit hook) kept in sync with a version published at, say, GitHub (`http://github.com`, a high-profile web site for collaborative Git-based development) or other easily accessible public Git repository web sites. Anybody can clone the published copy from

[6] `http://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools`
[7] `http://mws.cs.ru.nl/mwiki`
[8] The code is available from `http://github.com/JUrban/mwiki`

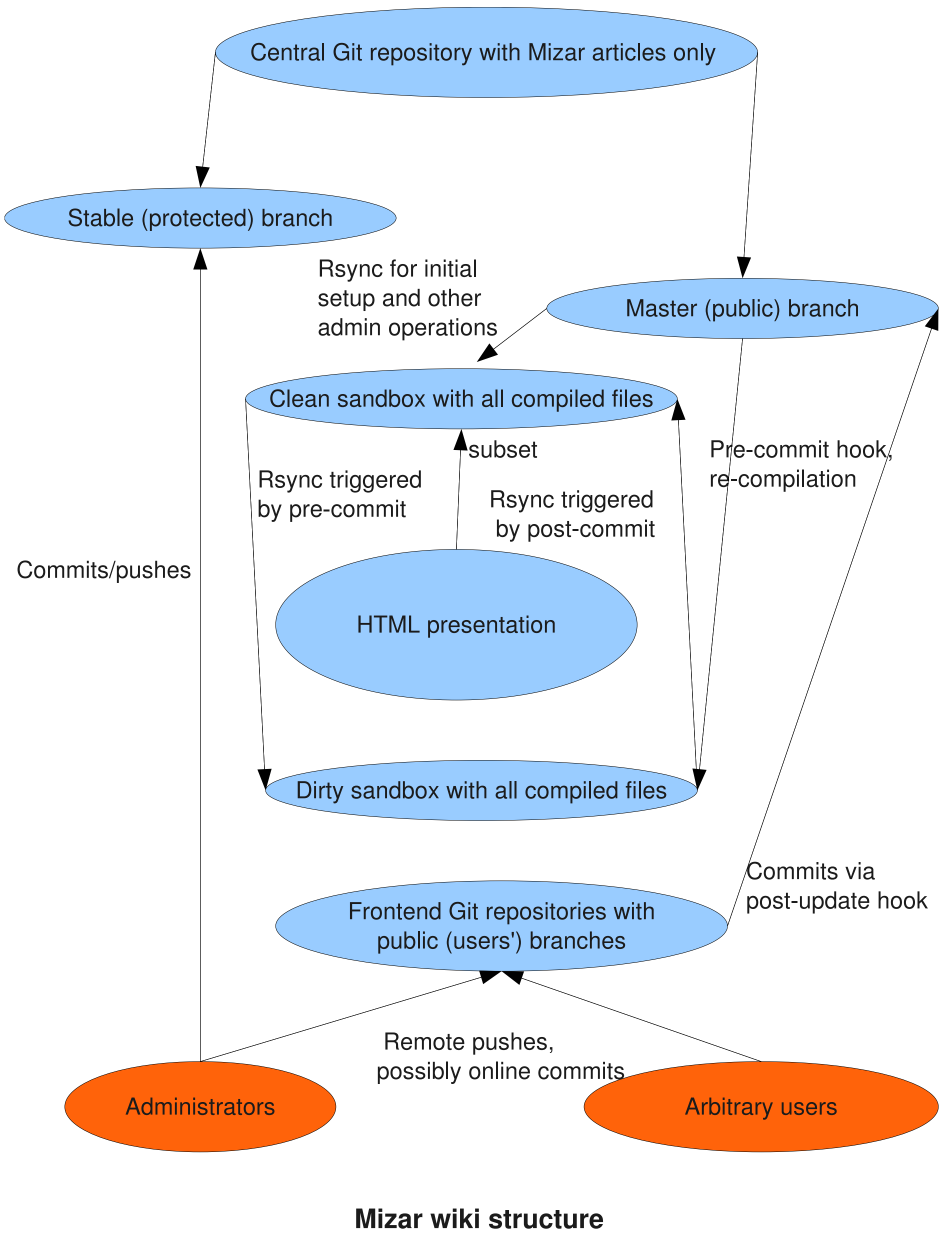

Central Git repository with Mizar articles only
Stable (protected) branch
Rsync for initial setup and other admin operations
Master (public) branch
Clean sandbox with all compiled files
subset
Rsync triggered by pre-commit
Rsync triggered by post-commit
Pre-commit hook, re-compilation
Commits/pushes
HTML presentation
Dirty sandbox with all compiled files
Commits via post-update hook
Frontend Git repositories with public (users') branches
Remote pushes, possibly online commits
Administrators
Arbitrary users
Mizar wiki structure

Fig. 1. Mizar wiki structure

such public locations, saving us some of the responsibility for maintaining access and sufficient bandwidth to our repositories.
- The repository contains basically only the source `.miz` files (Mizar formalizations), just as the Linux repositories only contain the source `.c`/`.h` files. All files that depend on the sources, and possibly intermediate meta-data constructible from them, are not part of the repository, and are excluded from the repository. We achieve this using Git's `.gitignore` feature.
- The *stable* branch of the repository is only available to administrators. New official versions of MML are pushed there.
- The *master* branch is the default one and it can be updated by any formally correct change (guarded by a suitable Git pre-commit hook). Essentially anybody can make changes (this is the wiki feature) using the mechanisms and checks described later. The *stable* branch and the Git history should serve as tools for easily reverting bad changes.
- The central repository is not directly updatable by users. Instead, it is cloned to another (frontend) repository on the server to which Git remote pushes can be made without any correctness checking. Limited control should be exercised to prevent malicious use[9]. Upon a successful remote push to the frontend, the Git post-receive hook is triggered. This hook attempts a commit to the master branch of the central repository, triggering in turn its pre-commit hook, and the formal verification of the updated library.
- Upon successful commit into the central repository, a post-commit hook is triggered. This hook generates HTML for the updated library, publishes it on the web, and does possible further related updates (updates of related clones on GitHub, notifications, etc.)
- The `gitweb` graphical web interface can be used for browsing the repository (comparing different changes and versions, and watching its history, etc.). Alternatives to `gitweb` abound: one is to use all the Git-related tools maintained and developed for GitHub.
- Editing is, initially, done locally, using a local customized editor like Emacs (used by most current Mizar users today); later, the articles are submitted using Git mechanisms. Similar "offline editing" is not uncommon in the wiki world: in the Emacs environment, for example, there are tools for similar offline interaction with Wikipedia. There are a number of options for providing additional direct in-browser editing (useful for smaller edits), ranging from the basic text-box and submit button to more advanced general web-based editors like in Wikipedia, to specialized online mathematical editors like ProofWeb.

6.1 Prototype Implementation

The implementation of the pre-commit checks, the post-commit cleanup, and related infrastructure is based on the following. We make essential use of makefiles

[9] It is possible to have a number of such frontends, and with sufficient infrastructure in place to actually move the main non-verifying frontends again to public hubs like GitHub.

to specify which files need to be rebuilt, how to build them, and how they depend on other files. In the central repository we have one central makefile M and, for each article a, a makefile M_a that specifies on which other articles a depends (e.g., what notations, definitions, theorems a uses). The master makefile M has targets that ensure that the whole of the (about-to-be-submitted) MML is coherent and without errors. Naturally, when one submits a small change, it is generally not necessary to re-verify the entire MML, and the makefiles are designed to re-verify the minimal necessary set of dependencies. The verification is carried out in a "fully compiled" MML, so all the auxiliary results of previous verifications (analogous to `.o` files for GCC) are available, from which the make program can infer whether they need to be re-computed.

As the library changes, the dependencies among the articles can change, and re-writing the makefiles by hand each time a dependency changes would be tedious. Following tools like `makedepend` for C and other languages (and probably some similar tools for other proof assistants), we have created the `envget` tool based on the Mizar codebase for quickly gathering the dependencies that are explicitly declared by an article[10]. Thus the makefiles M_a for each article a are themselves generated automatically using makefile targets defined in the master makefile M. The XML output of `envget` is combined with a suitable XSL stylesheet, producing M_a for an article a, containing only one target, and specifying the articles on which a depends. These dependency makefiles are refreshed only when the corresponding article is changed. This leads to a reasonably efficient Makefile-based setup: only the dependencies of changed files get possibly changed, and if that happens, only the (dynamically changed) dependency subgraph of MML influenced by the committed changes gets re-verified.

Where is the make program (governing verification of changes) invoked? How do we ensure that the central repository, assumed to be a coherent copy of the Mizar distribution, does not get "tainted" by incoherent user updates? In addition to a fully-compiled, coherent clean Mizar "sandbox" directory S_c, we maintain a (possibly) dirty sandbox directory S_d that may or may not be aligned with S_c. The two directories vary in the following manner. When a new user commit is proposed, we use `rsync` tool [23] to efficiently synchronize S_c and S_d, thereby ensuring that it is clean; we then copy all new Mizar source files to the dirty sandbox (that was just cleaned). Note that using `rsync` for this task provides a reasonable trade-off for solving several different concerns:

– to check the newly arrived changes without possibly destroying the previous correct version, we need to have a fresh directory with the most recent correct version;

[10] We are making use of the fact that in each verifiable Mizar article, one must declare what kinds of notations, definitions, proofs, etc., one depends on. If Mizar did not have this feature, then calculating dependencies would, presumably, be more difficult. Also note that these are just dependencies between articles, while it is certainly interesting future work to also calculate the precise dependencies between smaller-scale article *items* and use this information for smarter and leaner re-verification. The MPTP system [24] can be used for extracting information of this kind.

- the directory should contain as much pre-computed information as possible, because re-verifying and HTML-izing the whole library (more than 1000 articles) from scratch is an expensive operation (taking hours, even with high-end equipment), while we want to be as real-time as possible and re-use all available precomputed information;
- while the directory containing just the Mizar articles is reasonably small (81MB at the moment), the directory with the complete pre-computed information is prohibitively big: the internal library created from the Mizar articles, the environment for each article, and the HTML files are together more than 10GB.[11] Simply copying all this pre-compiled information would rule out a real-time experience, especially in cases when the user wants to change an article on which not many other articles depend.

Using `rsync` into a fresh directory addresses the first two issues (keeping the clean install intact and not re-verifying all from scratch). Using `rsync`, on average, reasonably ensures (by relying on smart `rsync` mechanisms) that of the clean sandbox over the dirty one does not take too long.

Since the clean sandbox S_c contained *everything*—Mizar source files and all generated auxiliary files—the dirty sandbox S_d now contains (after adding the newly modified Mizar files) nearly everything, too. All that is missing are up-to-date auxiliary files[12] to be generated from the modified Mizar source files. For that we invoke the master makefile M to generate possibly new dependency makefiles M_a, and then we invoke the master makefile to request a re-build/re-verification of the entire MML. Since we are working in a sandbox S_d that contains the all results of previous builds of the entire MML, "most" of the Mizar source files will not need to be looked at. (We put "most" in quotes because if one proposes to commit an update to a sufficiently basic Mizar article, then a good deal of computation is required to verify and propagate the change over the whole library, even though it could actually be a rather small edit. But such "foundational" edits are, apparently, uncommon.)

By using makefiles this way we are also able to exploit their ability to run multiple jobs in parallel. The dependency graph of the full MML is wide and rich enough that, when making "normal" edits, we can indeed benefit from this feature of make when running on multi-core machines or when employing grid/cloud-computing. Indeed, this is crucial for us to provide a sufficiently quick response to users who submit "normal" changes to MML.

Further opportunities for parallelization, based on a finer-grained analysis of Mizar articles than the one discussed here (where the dependency relation is between an entire Mizar article on other articles), are being investigated.

[11] This blow-up is caused by creating a local (XML) environment files for every article, and by having detailed XML and HTML representations of the article. A lot of information is kept in such files in a verbose form.

[12] That is, files witnessing the correctness of the verification, updated library files, re-generated HTML files, etc.

7 Future Work and Summary

A number of features can be added to our prototype. As already discussed, the world of wikis is dynamic and it is likely that all kinds of web frameworks and technologies will be used in the future for presenting the Mizar library, and collaboratively growing it. This is also why we stress a good basic model for dealing with the main data (articles) and processes (collaboration), i.e., using Git as a good mainstream DVCS, with a rapidly growing number of tools, frontends, and public collaborative platforms based on it.

The features that are being implemented at the time of writing this paper include: basic web-based editing; finer itemization, dependencies, parallelization and thus verification and HTML-ization speed for both simple (one-article) commits and multi-article commits; plugging in all kinds of powerful proof advice tools (automated theorem provers, library search engines, AI systems, etc.). Obviously these require nontrivial effort on all these tools and interfaces to them. Nontrivial and continuing work is actually already providing good HTML presentation of the formal articles, with all kinds of additional useful information and functions (following the symbols to their definitions being the most obvious one).

We have already mentioned the variety of tools available for Git and how public collaborative sites like GitHub rapidly develop all kinds of collaborative functions on top of Git. A similar remark is also true about, e.g., ikiwiki, and possibly about other wikis based on DVCSs. Thus, it seems to us that the mission of formal mathematical wiki builders should be to watch these rapidly developing collaborative technologies, customizing them (by providing suitable commit hooks, HTML-izers, dependency utilities, change propagation models, etc.) and complementing them (by providing all kinds of support tools developed for formal mathematics) rather than competing with them by starting from scratch.

While this paper is mainly about the Mizar proof assistant and its library, it should be clear that the functionalities we discussed (tools for good dependency extraction and focused re-verification and HTML-ization, possibly itemization and parallelization, all kinds of useful proof advice tools, etc.), the ideas and work presented here could be instantiated also to other proof assistants (Coq, Isabelle, HOL, etc.). When this is done, formal wikis could become the place where various formalisms and alternative formalizations meet, allowing collaboration on large formal projects, possibly having mechanisms also for gradual formalization of informal mathematical texts, and also allowing formal texts that are not completely correct. Our hope is that this infrastructure will attract more “normal/traditional” mathematicians to the advantages of formal mathematics, gradually making it more digestible, and possibly thus allowing further important steps in the inevitable computerization of human mathematics.

References

1. The Archive of Formal Proofs, http://afp.sourceforge.net/
2. Wikiwikiweb, http://c2.com/cgi/wiki?WikiWikiWeb

3. Corbineau, P., Kaliszyk, C.: Cooperative repositories for formal proofs. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, pp. 221–234. Springer, Heidelberg (2007)
4. Cruz-Filipe, L., Geuvers, H., Wiedijk, F.: C-CoRN, the Constructive coq Repository at Nijmegen. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 88–103. Springer, Heidelberg (2004)
5. D'Silva, V., Kroening, D., Weissenbacher, G.: A Survey of Automated Techniques for Formal Software Verification. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems 27(7), 1165–1178 (2008)
6. Git - fast version control system, http://git-scm.com/
7. Gonthier, G.: Formal proof—the four-color theorem. Notices of the American Mathematical Society 55(11), 1382–1393 (2008)
8. Hales, T.: A proof of the Kepler conjecture. Annals of Mathematics 162(3), 1065–1185 (2005)
9. Hales, T.: Formal proof. Notices of the American Mathematical Society 55(11), 1370–1381 (2008)
10. Harrison, J.: Formal Proof – Theory and Practice. Notices of the American Mathematical Society 55, 1395–1406 (2008)
11. Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: Formal verification of an OS kernel. In: Anderson, T. (ed.) Proceedings of the 22nd ACM Symposium on Operating Systems Principles, pp. 207–220. ACM Press, New York (2009)
12. Mathoverflow, http://mathoverflow.net/
13. Wolfram MathWorld: The Web's Most Extensive Mathematics Resource, http://mathworld.wolfram.com/
14. Matuszewski, R., Rudnicki, P.: MIZAR: the first 30 years. Mechanized Mathematics and its Applications 4(1), 3–24 (2005)
15. The Mizar Mathematical Library, http://mizar.org/library/
16. Naumowicz, A., Kornilowicz, A.: A Brief Overview of Mizar. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 67–72. Springer, Heidelberg (2009)
17. Planetmath, http://planetmath.org/
18. The polymath blog, http://polymathprojects.org/
19. Proofwiki, http://www.proofwiki.org/wiki/Main_Page
20. The QED Manifesto. In: Bundy, A. (ed.) CADE 1994. LNCS (LNAI), vol. 814, pp. 238–251. Springer, Heidelberg (1994)
21. The QPQ Deductive Software Repository, http://www.qpq.org
22. Rudnicki, P., Trybulec, A.: On the integrity of a repository of formalized mathematics. In: Asperti, A., Buchberger, B., Davenport, J.H. (eds.) MKM 2003. LNCS, vol. 2594, pp. 162–174. Springer, Heidelberg (2003)
23. Trigdell, A.: Efficient Algorithms for Sorting and Synchronization. Ph.D. thesis, Australian National University (1999)
24. Urban, J.: MPTP 0.2: Design, implementation, and initial experiments. J. Autom. Reasoning 37(1-2), 21–43 (2006)
25. vdash: What is vdash?, http://www.vdash.org/intro/

Author Index

Alama, Jesse 455
Asperti, Andrea 147 263

Bensaid, Hicham 19
Bortin, Maksym 34
Butelle, Franck 158

Caferra, Ricardo 19
Calmet, Jacques 14
Campbell, John A. 14
Carette, Jacques 157, 172
Chen, Xiaoyu 278
Cohen, Cyril 189
Collins, Joseph B. 293

Davenport, James H. 1
Domínguez, César 204

Echenim, Mnacho 49

Feinsilver, Philip 64

Geuvers, Herman 440, 455
Grabowski, Adam 300

Heeren, Bastiaan 315
Heras, Jónathan 331
Hivert, Florent 158

Jeffrey, David J. 234
Jeuring, Johan 315
Jucovschi, Constantin 336

Kerber, Manfred 345
Kohlhase, Andrea 355
Kohlhase, Michael 336, 355, 370
Kolev, Mikhail K. 76
Kovalchuk, Andriy 385

Lange, Christoph 355
Levitsky, Vyacheslav 385
Libbrecht, Paul 398
Li, Yongbin 89
Loddo, Jean-Vincent 101
Lucas, Salvador 116

Lüth, Christoph 34
Lyaletski, Alexander 411

Mahboubi, Assia 189
Mayero, Micaela 158
McKinna, James 440

Paşca, Ioana 219
Pascual, Vico 331
Peltier, Nicolas 19, 49

Rabe, Florian 370
Rich, Albert D. 234
Romero, Ana 331
Rowinska-Schwarzweller, Agnieszka 427
Rubio, Julio 204, 331
Rudnicki, Piotr 455

Sacerdoti Coen, Claudio 147
Saiu, Luca 101
Salvy, Bruno 13
Samolyuk, Igor 385
Schott, René 64
Schwarzweller, Christoph 300, 427
Sexton, Alan P. 172
Sorge, Volker 172
Sutcliffe, Geoff 132

Tankink, Carst 440
Tarau, Paul 247
Tassi, Enrico 263
Toumazet, Frédéric 158

Urban, Josef 132, 455

Verchinine, Konstantin 411

Watt, Stephen M. 172
Wiedijk, Freek 440

Yanchuk, Valentyn 385

Zeilberger, Doron 262
Zholudev, Vyacheslav 370