

Florian Rabe · William M. Farmer
Grant O. Passmore · Abdou Youssef (Eds.)

LNAI 11006

Intelligent Computer Mathematics

11th International Conference, CICM 2018
Hagenberg, Austria, August 13–17, 2018
Proceedings

 Springer

Lecture Notes in Artificial Intelligence

11006

Subseries of Lecture Notes in Computer Science

LNAI Series Editors

Randy Goebel

University of Alberta, Edmonton, Canada

Yuzuru Tanaka

Hokkaido University, Sapporo, Japan

Wolfgang Wahlster

DFKI and Saarland University, Saarbrücken, Germany

LNAI Founding Series Editor

Joerg Siekmann

DFKI and Saarland University, Saarbrücken, Germany

More information about this series at <http://www.springer.com/series/1244>

Florian Rabe · William M. Farmer
Grant O. Passmore · Abdou Youssef (Eds.)

Intelligent Computer Mathematics

11th International Conference, CICM 2018
Hagenberg, Austria, August 13–17, 2018
Proceedings

Editors

Florian Rabe
Department of Computer Science
Friedrich-Alexander-Universität
Erlangen-Nürnberg
Erlangen, Bayern
Germany

William M. Farmer
Department of Computing and Software
McMaster University
Hamilton, ON
Canada

Grant O. Passmore
University of Cambridge
Cambridge
UK

Abdou Youssef
Department of Computer Science
The George Washington University
Washington, DC
USA

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Artificial Intelligence
ISBN 978-3-319-96811-7 ISBN 978-3-319-96812-4 (eBook)
<https://doi.org/10.1007/978-3-319-96812-4>

Library of Congress Control Number: 2018949030

LNCS Sublibrary: SL7 – Artificial Intelligence

© Springer International Publishing AG, part of Springer Nature 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This volume contains the formally reviewed papers presented at CICM 2018, the 11th Conference on Intelligent Computer Mathematics, which was held during August 13–17, 2018 at the Research Institute for Symbolic Computation (RISC) in Hagenberg, Austria.

Mathematics is “the queen of the sciences” (Friedrich Gauss), and “the language with which God has written the universe” (Galileo Galilei). But the collection of mathematical knowledge is exploding — each year there are over 100,000 new articles. Digital solutions are becoming the prevalent means for the generation, communication, processing, storage, and curation of mathematical information. CICM brings together the many separate communities that have developed theoretical and practical solutions for these challenges including computation, deduction, narration, and data management.

CICM is the result of merging three series of meetings that were previously held independently: *Calculus* (concerned with the integration of symbolic computation and mechanized reasoning), *Digital Mathematical Libraries (DML)*, and *Mathematical Knowledge Management (MKM)*. CICM has been held annually since 2008, with previous meetings in Birmingham (UK, 2008), Grand Bend (Canada, 2009), Paris (France, 2010), Bertinoro (Italy, 2011), Bremen (Germany, 2012), Bath (UK, 2013), Coimbra (Portugal, 2014), Washington DC (USA, 2015), Białystok (Poland, 2016), and Edinburgh (UK, 2017).

CICM 2018 solicited formal submissions in several tracks. The *Calculus*, *Digital Mathematics Libraries*, and *Mathematical Knowledge Management* tracks corresponded to the subject areas of the predecessor meetings. Orthogonally, the *Systems and Projects* track called for descriptions of digital resources, such as data and systems, and of projects, whether old, current, or new, as well as survey papers covering any topics of relevance to the CICM community.

This year, CICM received 36 submissions, of which 23 were accepted after formal review. These submissions consisted of nine *Calculus* (five accepted), two *DML* (one accepted), and ten *MKM* papers (seven accepted), as well as ten system/dataset descriptions (eight accepted), and five surveys and project descriptions (two accepted). Each submission received at least three reviews. The reviewing included a response period, in which authors could clarify points raised by the reviewers. This made for a highly productive round of deliberations before the final decisions were taken. In three cases an open-ended shepherding phase was used, during which the authors were allowed to improve their papers under the guidance of a designated Program Committee (PC) member. All shepherded papers were improved sufficiently to be eventually accepted. The resulting set of papers forms the content of these proceedings.

The PC was chaired by the editors of this volume. Florian Rabe served as general chair and as track chair of the *DML* track. Grant Passmore, William Farmer, and

Abdou Youssef served as track chairs of the Calculemus, MKM, and Systems and Projects tracks, respectively.

The PC work was managed using the EasyChair system. This year CICM used a single track with two submission categories for classifying papers into Calculemus, DML, and MKM, as well as into regular papers, surveys/projects, and system/dataset descriptions.

Submissions co-authored by PC members were allowed, and all conflicts of interest were managed via EasyChair: After registering the conflicts, the submissions were completely hidden from conflicted PC members. This procedure was also used for submissions of PC chairs, in which case submissions were handled by one of the other chairs.

In addition to presentations of the accepted papers, CICM 2018 had three invited presentations by Akiko Aizawa, Bruno Buchberger, and Adri Olde Daalhuis. Moreover, six workshops were held as part of the conference:

- Workshop on Computer Algebra in the Age of Types (CAAT), featuring a tutorial on the Idris programming language, and an introduction to the MMT system, while inviting practicing researchers in computer algebra, formal methods, and programming languages to present and discuss approaches and form a common vision for the next generation of computer algebra
- Workshop on Computer Mathematics in Education — Enlightenment or Incantation (CME-EI), considering recent developments in computer mathematics while discussing potential impact of respective tools and reconsidering developers' responsibility for such impact
- Workshop on Formal Mathematics for Mathematicians (FMM), allowing mathematicians interested in computer assistance and researchers in formal and computer-understandable mathematics to meet and exchange ideas
- Workshop on Formal Verification of Physical Systems (FVPS), focusing on formal verification techniques for the modeling, analysis, and verification of safety and security critical physical systems
- Workshop on Mathematical Models and Mathematical Software as Research Data (M3SRD), presenting and discussing ideas, concepts, standardization, and service development for important classes of mathematical research data, especially mathematical models and mathematical software
- OpenMath Workshop, focusing on new research about and description of new content dictionaries for OpenMath, a language for exchanging mathematical formulas across systems

The workshop programs were managed independently by the respective organizers, and the workshops were coordinated by Osman Hasan, who served as the workshop chair.

Furthermore, CICM 2018 featured a Doctoral Program, which provided a dedicated forum for PhD students to present their research and receive advice from senior researchers serving as mentors. Diane Gallois-Wong served as doctoral program chair.

Finally, CICM recognizes that relevant research can be conducted and communicated in various forms that are not always fitting for publication as a formal paper.

Therefore, CICM 2018 also solicited other contributions including work-in-progress papers, system demos, posters, and tutorials.

The proceedings of these events are not part of this volume and are published separately. The entire program and additional materials are available at <http://cicm-conference.org/2018/>.

All local organizational aspects were managed by Wolfgang Windsteiger, who served as the conference chair. Serge Autexier served as the publicity chair of the conference.

The organizers want to thank the team of RISC, notably Tanja Gutenbrunner, Ramona Öhme-Pöchinger, and Alexander Maletzky, for their valuable support in the local organization of the meeting, as well as the team at the University of Applied Sciences Hagenberg for providing lecture halls and workshop rooms including all the equipment. The conference received financial support through the University of Linz (JKU), the Doctoral Program “Computational Mathematics” (W1214) at JKU, the Regional Government of Upper Austria, and the Raiffeisen Landesbank Oberösterreich (RLB OÖ).



June 2018

Florian Rabe
William Farmer
Grant Passmore
Abdou Youssef

Organization

Program Committee

Waqar Ahmad	National University of Sciences and Technology (NUST), Pakistan
Yves Bertot	Inria, France
Thierry Bouche	Université Grenoble Alpes, France
Jacques Carette	McMaster University, Computing and Software, Canada
Howard Cohl	NIST
Joseph Corneli	Aesthetic Integration
Simon Cruanes	University of Bath, UK
James H. Davenport	Nuance Comms and University of Birmingham, UK
Valeria de Paiva	ENSIIE-Samovar
Catherine Dubois	Coventry University, UK
Matthew England	Institute e-Austria Timisoara and West University of Timisoara, Romania
Madalina Erascu	McMaster University, Canada
William Farmer	Mathematical Reviews/AMS
Patrick Ion	The University of Edinburgh, UK
Paul Jackson	University of Cambridge, UK
Mateja Jamnik	University of Innsbruck, Austria
Cezary Kaliszyk	University of Birmingham, UK
Manfred Kerber	University of Applied Sciences Neu-Ulm, Germany
Andrea Kohlhase	FAU Erlangen-Nürnberg, Germany
Michael Kohlhase	Vienna University of Technology, Austria
Laura Kovacs	University of Cambridge, UK
Wenda Li	Inria, France
Assia Mahboubi	University of Bialystok, Poland
Adam Naumowicz	Aesthetic Integration
Grant Passmore	Carnegie Mellon University, USA
André Platzer	FAU Erlangen-Nürnberg, Germany and LRI Paris, France
Florian Rabe	University of Bologna, Italy
Claudio Sacerdoti Coen	Universität Konstanz, Germany
Moritz Schubotz	Masaryk University, Czech Republic
Petr Sojka	FIZ Karlsruhe, Germany
Wolfram Sperber	Czech Technical University in Prague, Czech Republic
Josef Urban	RISC Institute, JKU Linz, Austria
Wolfgang Windsteiger	The George Washington University, USA
Abdou Youssef	

Additional Reviewers

Betzendahl, Jonas
Brown, Chad
Condoluci, Andrea
Cordwell, Katherine
Dimanov, Botty
Eades Iii, Harley
Gleiss, Bernhard
Humenberger, Andreas
Immler, Fabian
Kalvala, Sara

Müller, Dennis
Novotný, Vít
Raggi, Daniel
Schaefer, Jan Frederik
Shams, Zohreh
Siddique, Umair
Sogokon, Andrew
Théry, Laurent
Turaga, Prathamesh
Wang, Duo

Contents

System Description: XSL-Based Translator of Mizar to LaTeX.	1
<i>Grzegorz Bancerek, Adam Naumowicz, and Josef Urban</i>	
Translating the IMPS Theory Library to MMT/OMDoc.	7
<i>Jonas Betzendahl and Michael Kohlhase</i>	
Using the Isabelle Ontology Framework: Linking the Formal with the Informal	23
<i>Achim D. Brucker, Idir Ait-Sadoune, Paolo Crisafulli, and Burkhart Wolff</i>	
Automated Symbolic and Numerical Testing of DLMF Formulae Using Computer Algebra Systems.	39
<i>Howard S. Cohl, André Greiner-Petter, and Moritz Schubotz</i>	
Concrete Semantics with Coq and CoqHammer	53
<i>Lukasz Czajka, Burak Ekici, and Cezary Kaliszyk</i>	
Automated Determination of Isoptics with Dynamic Geometry	60
<i>Thierry Dana-Picard and Zoltán Kovács</i>	
Biform Theories: Project Description	76
<i>Jacques Carette, William M. Farmer, and Yasmine Sharoda</i>	
A Coq Formalization of Digital Filters.	87
<i>Diane Gallois-Wong, Sylvie Boldo, and Thibault Hilaire</i>	
<i>MathTools</i> : An Open API for Convenient MATHML Handling	104
<i>André Greiner-Petter, Moritz Schubotz, Howard S. Cohl, and Bela Gipp</i>	
Aligator.jl – A Julia Package for Loop Invariant Generation.	111
<i>Andreas Humenberger, Maximilian Jaroschek, and Laura Kovács</i>	
Enhancing ENIGMA Given Clause Guidance.	118
<i>Jan Jakubův and Josef Urban</i>	
Formalized Mathematical Content in Lecture Notes on Modelling and Analysis.	125
<i>Michael Junk, Stefan Hölle, and Sebastian Sahli</i>	

Isabelle Import Infrastructure for the Mizar Mathematical Library 131
Cezary Kaliszyk and Karol Pqk

Discourse Phenomena in Mathematical Documents 147
*Andrea Kohlhase, Michael Kohlhase,
and Taweechai Ouypornkochagorn*

Finding and Proving New Geometry Theorems in Regular Polygons
with Dynamic Geometry and Automated Reasoning Tools 164
Zoltán Kovács

Gröbner Bases of Modules and Faugère’s F_4 Algorithm
in Isabelle/HOL 178
Alexander Maletzky and Fabian Immler

MathChat: Computational Mathematics via a Social Machine 194
Manfred Minimair

Automatically Finding Theory Morphisms for Knowledge Management. 209
Dennis Müller, Michael Kohlhase, and Florian Rabe

Goal-Oriented Conjecturing for Isabelle/HOL 225
Yutaka Nagashima and Julian Parsert

Knowledge Amalgamation for Computational Science and Engineering 232
Theresa Pollinger, Michael Kohlhase, and Harald Köstler

Validating Mathematical Theorems and Algorithms with RISCAL. 248
Wolfgang Schreiner

First Experiments with Neural Translation of Informal to Formal
Mathematics 255
Qingxiang Wang, Cezary Kaliszyk, and Josef Urban

Deep Learning for Math Knowledge Processing 271
Abdou Youssef and Bruce R. Miller

Author Index 287



System Description: XSL-Based Translator of Mizar to LaTeX

Grzegorz Bancerek², Adam Naumowicz^{1(✉)}, and Josef Urban²

¹ University of Białystok, Białystok, Poland
adamn@math.uwb.edu.pl

² Czech Technical University in Prague, Prague, Czech Republic

Abstract. We describe a new version of the Mizar-to-L^AT_EX translator. The system has been re-implemented as XSL stylesheets instead of as Pascal programs, allowing greater flexibility. It can now be used to generate both L^AT_EX/PDF and HTML with MathJax code. We also experimentally support generation of full proofs. Finally, the system is now available online and through the Mizar Emacs interface.

1 Introduction

The Mizar project [4] has since its inception in 1973 focused on formal representation of mathematics that would be as close possible to its representation in natural language. After the Mizar language emerged and a larger number of articles have been collected in the Mizar Mathematical Library (MML) [3, 7], the Mizar team started to experiment with translating the Mizar articles into L^AT_EX. This translation has been evolving for the last three decades [1, 5, 10].

Here we describe the most recent large re-implementation of the system which switches from custom Pascal programs to more flexible XSL processing, done primarily by the first author over the last eight years. This in turn builds on the XML-ization of the Mizar processing, done over the past decade [9, 11]. The system can now produce both L^AT_EX/PDF and HTML using MathJax for displaying formulas and terms. We experimentally support also translation of full proofs. The system is used for in-house production of the journal *Formalized Mathematics*¹ [8] and it is newly also available as a remote PDF/HTML service for Mizar authors via the Emacs authoring environment for Mizar [12], similarly to the MizAR [13] ATP service for Mizar.

G. Bancerek—Deceased.

A. Naumowicz—Partially supported by the project N° 2017/01/X/ST6/00012 financed by the National Science Center, Poland and the COST Action EUTypes CA15123. The Mizar processing has been performed using the infrastructure of the University of Białystok High Performance Computing Center.

J. Urban—Supported by the ERC Consolidator grant no. 649043 AI4REASON, and by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15 003/0000466 and the European Regional Development Fund.

¹ <https://www.degruyter.com/view/j/forma>.

2 Summary of the Old Mizar-to- \LaTeX Translation

The previous incarnation of the translator, as well as its history, are described in detail in [1]. The complete process was carried out by a set of eight custom programs (`fmparse`, `newfmfrm`, `addfmfrm`, `fmfrm`, `resvar`, `fmnotats`, `fmanalyz` and `jformath`) run in succession after preparing the article with the standard Mizar accommodator utility and before producing the final output with \LaTeX using \BIBTeX metadata provided by the user/editor.

Let us only briefly recall that the translation of the non-formula text parts was done as a static mapping of the keywords. For example the `commutativity` property for functors was expressed by the sentence *Let us observe that the functor is commutative*. The translation of atomic formulas, terms, and types was done according to a database of \LaTeX *translation patterns* for the Mizar symbols with arities (a *format* in the Mizar terminology). Each pattern consists of a control header (deciding e.g. about using the math mode in \LaTeX , bracketing, etc.) and one or more proper patterns that map the format with arguments to \LaTeX in various contexts. Following is an example of such a pattern for a typical `PartUnion` functor:²

```
01 0 2
0PartUnion
mol@s#1#2; \bigcup_{\beta<_R B} \beta
```

This means that the Mizar term `PartUnion(B,R)` would be displayed as $\bigcup_{\beta <_R B} \beta$. Note that in this case the translation is quite nontrivial and it reveals information about how the symbol `PartUnion` is defined in Mizar.

All top-level elements of an article, i.e. theorems, definitions, schemes, reservations and global shortcuts were presented in the rendering, but proofs were not translated as a rule. Single letter variables occurring in an article were preserved while others were abbreviated into single letters with indices.

3 Description of the New Technology

The new technology of automated translation currently used for Mizar texts published in the *Formalized Mathematics* journal is based on XSL translation templates applied to the XML representation of the weakly-strict Mizar [9] encoding of the original Mizar input file (`*.wsx` file). However, the semantic representation generated by the Mizar verifier (`*.xml` file) is also used to decode links to external articles. All bibliographic metadata are first translated to special XML format and merged with information extracted from the Mizar article. Global (for the journal) \LaTeX translation patterns are also kept in the XML `pub.xml` file. In the following sections we describe the basic functions of the main XSL stylesheets. They have been designed to perform well-defined simple iterative tasks within the process of generating the final \LaTeX rendering for

² http://mizar.uwb.edu.pl/version/current/html/pcomps_2.html#K1, see also [6].

PDF and MathJax-enabled HTML presentation. Let us note that the presented XSL translation is not a 1-1 reimplementaion of former Pascal-based code. The current multi-pass method has been implemented from scratch to make use of available XML representation formats, whereas the former was bound to the internal structures of the Mizar verifier.

3.1 addformat.xsl

This is the main stylesheet responsible for selecting information to be translated from the weakly-strict Mizar representation. Identified symbols are matched with their format specification to be later replaced by concrete translations.

3.2 addtranslation.xsl

This script augments the processed file with available translation patterns. An example of a concrete pattern (for the previously mentioned PartUnion functor) as extracted from the `pub.xml` file looks as follows:

```
<Translation
  voc="PCOMPS_2" kind="0" symbolnr="1" symbol="0PartUnion" argsnr="2"
  leftargsnr="0" rightargsnr="4" format="01 0 2" header="mol@s#1#2;"
  priority="8" forcing="rqw" context1="1" context2="" TeX-mode="m">
  <pattern>
    \bigcup_{\beta{\&lt;_}_{X pos="lower" locus="2"/>}}{X locus="1"/>}\beta
  </pattern>
</Translation>
```

The system also proposes formats for new definitions introduced in the current article. The `unknown_patterns.xsl` stylesheet handles unknown patterns.

3.3 varrepr.xsl

The task of this stylesheet is to replace any identifiers that contain names of Greek letters into corresponding \LaTeX symbols. Longer variable identifiers are given standardized representations with subscripts (e.g. AA becomes A_1).

3.4 multipred.xsl

This procedure is technically split into several passes of `multipred.xsl`, `multipred2.xsl` and `multipred3.xsl` run in succession together with `prune.xsl`. The goal is to locate in the input text a list of constructs with a qualifying format that can be printed together in a shortened form. For example, in the PCOMPS_2 article, instead of the literal translation: “ G_9 is cover of P_6 , and G_9 is finer than F_9 ” (cf. [6]) the script produces a shortened phrase “ G_9 is cover of P_6 and finer than F_9 ”.

3.5 transitive.xsl

This stylesheet improves the quality of the translation by generating output of the form: “ $x < y < z$ ” instead of “ $x < y$ and $y < z$ ”, i.e. it joins consecutive predicates with shared arguments as is usually done in informal mathematics.

3.6 compress.xsl

There are several independent stylesheets that compress and thus make more natural for the reader the occurrences of particular constructs in a common context. These are: `compress_let.xsl`, `compress_for.xsl`, `compress_assume.xsl`, `brackets.xsl`, `compress_func.xsl`, and `compressres.xsl` for generalizations, quantifiers, assumptions, brackets, functor definitions with common arguments, and variable reservations, respectively.

3.7 recognize-programs.xsl

The set of templates `recognize-programs.xsl`, `recognize-programs2.xsl` and `recognize-programs3.xsl` is used to generate custom encoding of specific complex terms - representation of programs. For example, the \LaTeX translation of the following Mizar theorem statement from the AOFA_I00 article about an algorithm defined in terms of a custom *if-while* algebra:

```
theorem
  for n,s,i being Variable of g st ex d being Function st d.n = 1 & d.s
    = 2 & d.i = 3 & d.b = 4 holds s:=1\;for-do(i:=2, i leq n, i+=1, s**=i)
    is_terminating_wrt g
```

is rendered in PDF as follows (cf. [2]):

Now let Γ denotes the program

```
s:=1;
for i:=2 until i leq n step i+=1 do
  s*=i
done
```

Then we state the propositions:

(56) Let us consider variables n, s, i in g . Suppose there exists a function d such that $d(n) = 1$ and $d(s) = 2$ and $d(i) = 3$ and $d(b) = 4$. Then Γ is terminating w.r.t. g .

3.8 article2latex.xsl and article2html.xsl

Depending on the output format, based on the preparatory tasks performed by the previously mentioned stylesheets, one may choose to generate either \LaTeX code for a self-contained PDF article, or code embedded in an HTML document to be rendered by MathJax. If \LaTeX code is chosen, we finally run `pdflatex` using several Mizar-specific headers.

3.9 Remote Service and Processing Times

Since the translation toolchain has many components and relies on a number of custom tools and their versions installed, we make it available as an online service. This is similar to the solution taken for the MizAR [13] ATP

service for Mizar. The Mizar users can now send their articles from Emacs to the service for PDF and HTML translation of their current work independently of the publication process of *Formalized Mathematics*. This is done by selecting the *Mizar*→*Remote solving*→*Produce PDF online* and *Mizar*→*Remote solving*→*Produce MathJax online* menu options, respectively.³ The remote processing of a basic Mizar article such as XBOOLE_1 takes about 15s and the whole MML can be processed locally on the server overnight. This is quite comparable to the speed of the earlier Pascal-written version of the toolchain.

4 Conclusion and Future Work

The described re-implementation of the Mizar to \LaTeX translation system is based on the flexible and easily extensible XML/XSL technology rather than custom Pascal programs tied to the specific implementation of the core Mizar system. Thanks to this approach, a set of shared scripts and stylesheets can be used for the production of the printed editions of the journal *Formalized Mathematics* but also as an on-demand remote PDF/HTML service for Mizar authors. Users of the Emacs authoring environment for Mizar can now experiment with the PDF and HTML presentation of their current work based on the shared \LaTeX translation.

Future work on translation will focus on adding more natural language based linguistic features to the generated mathematical text. In particular, the human-friendly presentation of the structure of (nested with varying levels of importance) proofs will be investigated. It is now much easier to experiment with more natural translation patterns as a step by step improvement of the translation by just modifying the `addformat.xml` stylesheet which controls filtering relevant information from the original Mizar article and either the `article2latex.xml` or `article2html.xml` templates that generate the final specific rendering in \LaTeX or MathJax/HTML format.

References

1. Bancerek, G.: Automatic translation in Formalized Mathematics. *Mech. Math. Appl.* **5**(2), 19–31 (2006)
2. Bancerek, G.: Mizar analysis of algorithms: algorithms over integers. *Formaliz. Math.* **16**(2), 177–194 (2008). <https://doi.org/10.2478/v10037-008-0024-0>
3. Bancerek, G., Bylinski, C., Grabowski, A., Kornilowicz, A., Matuszewski, R., Naumowicz, A., Pak, K.: The role of the Mizar Mathematical Library for interactive proof development in Mizar. *J. Autom. Reason.* **61**(1–4), 9–32 (2018). <https://doi.org/10.1007/s10817-017-9440-6>
4. Bancerek, G., et al.: Mizar: state-of-the-art and beyond. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) *CICM 2015. LNCS (LNAI)*, vol. 9150, pp. 261–279. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20615-8_17

³ Both menu items trigger the `mizar-tex-remote` function in the current `mizar.el` Emacs Lisp mode for Mizar available at <https://github.com/JUrban/mizarmode/blob/master/mizar.el>.

5. Bancerek, G., Carlson, P.: Mizar and the machine translation of mathematics documents. http://www.mizar.org/project/banc_carl93.ps
6. Borys, L.: On paracompactness of metrizable spaces. *Formaliz. Math.* **3**(1), 81–84 (1992). <http://fm.mizar.org/1992-3/pdf3-1/pcomps.2.pdf>
7. Grabowski, A., Kornilowicz, A., Naumowicz, A.: Four decades of Mizar - foreword. *J. Autom. Reason.* **55**(3), 191–198 (2015). <https://doi.org/10.1007/s10817-015-9345-1>
8. Grabowski, A., Schwarzweller, C.: Revisions as an essential tool to maintain mathematical repositories. In: Kauters, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) *Calculemus/MKM - 2007. LNCS (LNAI)*, vol. 4573, pp. 235–249. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73086-6_20
9. Naumowicz, A., Piliszek, R.: Accessing the Mizar library with a weakly strict Mizar parser. In: Kohlhase, M., Johansson, M., Miller, B., de Moura, L., Tompa, F. (eds.) *CICM 2016. LNCS (LNAI)*, vol. 9791, pp. 77–82. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42547-4_6
10. Trybulec, A., Rudnicki, P.: A collection of $\text{T}_{\text{E}}\text{X}$ ed Mizar abstracts. <http://www.mizar.org/project/TR-89-18.pdf>
11. Urban, J.: XML-izing Mizar: making semantic processing and presentation of MML easy. In: Kohlhase, M. (ed.) *MKM 2005. LNCS (LNAI)*, vol. 3863, pp. 346–360. Springer, Heidelberg (2006). https://doi.org/10.1007/11618027_23
12. Urban, J.: MizarMode - an integrated proof assistance tool for the Mizar way of formalizing mathematics. *J. Appl. Log.* **4**(4), 414–427 (2006)
13. Urban, J., Rudnicki, P., Sutcliffe, G.: ATP and presentation service for Mizar formalizations. *J. Autom. Reason.* **50**, 229–241 (2013). <https://doi.org/10.1007/s10817-012-9269-y>



Translating the IMPS Theory Library to MMT/OMDoc

Jonas Betzendahl^(✉) and Michael Kohlhase^(✉)

Computer Science, FAU Erlangen-Nürnberg, Erlangen, Germany
{jonas.betzendahl,michael.kohlhase}@fau.de

Abstract. The IMPS system by Farmer, Guttman and Thayer was an influential automated reasoning system, pioneering mechanisations of features like theory morphisms, partial functions with subsorts, and the little theories approach to the axiomatic method. It comes with a large library of formalised mathematical knowledge covering a broad spectrum of different fields. Since IMPS is no longer under development, this library is in danger of being lost. In its present form, it is also not compatible for use with any other mathematical system.

To remedy that, we formalise the logic of IMPS (LUTINS), and draw on both the original theory library source files as well as the internal data structures of the system to generate a representation in a modern knowledge management format. Using this approach, we translate the library to OMDoc/MMT and verify the result using type-checking in the MMT system against our implementation of LUTINS.

1 Introduction

There are many libraries of formal knowledge, but unfortunately, each one needs a different mathematical software system to interpret it – the system it was originally written for. This barrier of non-interoperability severely limits possible uses of any one mathematical software system as well as progress on that system itself. Helpful tooling that was implemented for one system often can not be used for developing another. Some translations between different systems exist but they are often ad-hoc, only one-directional (although both systems are in frequent use) or overly restricted by the logical frameworks or foundations. Approaching the problem with only direct system-to-system translations without common ground also means inviting scaling problems: for n systems we would need $\mathcal{O}(n^2)$ translations.

Most theorem proving systems today tend to fix (in what we will refer to as the *big theories approach*) one particular logical foundation along with its primitives (i.e. types, axioms, rules, . . .) and only use conservative extensions to model domain knowledge (i.e. theorems, definitions, . . .). This goes against the way that modern mathematics is usually done on chalkboard or paper, where the foundation is often hidden and almost never directly referred to. It also makes it harder for two systems with different logical frameworks to successfully interact.

Another danger to effective use of mathematical knowledge libraries across software systems is that if one of these systems eventually falls out of use, the library is in danger of being lost to bitrot as fewer and fewer machines are actually capable of running the required software to interpret it.

In this paper we attempt to “rescue” one library in particular from this fate and make it interoperable at the same time. IMPS is an *Interactive Mathematical Proof System*, originally developed at The MITRE Corporation by William M. Farmer, Joshua Guttman, and Javier Thayer. Its library is home to a large amount of well-developed formalised mathematics with over 180 different theories and over 1200 distinct theorems and their proofs.

The IMPS system itself [TMI] has not been in active development or regular use for well over 20 years now and thus the library is in acute danger of being lost. The system (and its library) is especially interesting because it was the first theorem proving assistant to make heavy use of *theory morphisms* and with emphasis on the *little theories* approach to mathematics.

Concretely, we present a translation of the IMPS library into OMDoc/MMT [Koh06, RK13], a content markup scheme for (collections of) mathematical documents (including articles, textbooks, and theorem prover libraries) that shares key design choices with the IMPS system, such as the focus on theory morphisms and adherence to the little theories approach.

Contribution. We build on and complete the earlier and incomplete efforts by [Li02] of translating the IMPS library to OMDoc. Our work differs from Li’s previous attempt in that it does not try to translate from *only* the internal IMPS data structures. Instead, our implementation *also* reads the corresponding source files to extract additional structure from them. This allows us to compare and corroborate data from one direction of inquiry with data from the other.

Concretely we extend and adapt Li’s export mechanism to create a JSON representation of the internal IMPS data structures. Both this and the original source files of the mathematical library are then parsed by our importer extension to MMT to create a structured and typed representation of almost all mathematical objects in the source files, with the only notable exception being proof scripts and macetes (theory-aware tactics).

This representation can then easily be translated into the OMDoc/MMT language, with a formalisation of the foundational logic of IMPS (called LUTINS, see Sect. 2.1) serving as a formal basis for the translation. The generated output is verified (i.e. type-checked wrt. the **LF** meta-logic [HHP93]) by the MMT system against the implementation of the underlying logic LUTINS to establish a certain, if partial, level of correctness of the translation progress.

This two-layered implementation has the benefit of future-proofing the OMDoc export against potential changes in the format.

The OMDoc/MMT output of the translation not only offers a semantically self-contained archive format, it could also be used in various ways by mathematical knowledge management systems or function as a reference point for other knowledge in a (partially) shared meaning space.

Related Work. There have been multiple attempts at translating libraries from one theorem proving system to another in an ad-hoc manner. Examples include translations from HOL Light to Coq [KW10], from Isabelle/HOL to Isabelle/ZF [KS10] (benefiting from the shared logical framework), from HOL to Isabelle/HOL [OS06] and from Mizar to Isabelle [KPU16].

The translation approach using OMDoc/MMT has been previously used (and shown to be successful) in a number of importers for the MMT system for different mathematical systems, such as PVS [Koh+17], Mizar [Ian+13] and HOL Light [KR14], as part of the OAF (Open Archive of Formalisations) project [OAF].

In all of these, the underlying logical foundation of the system has first been formalised natively in OMDoc/MMT as part of the LATIN library – an OMDoc-based atlas of formal logics, type theories, foundations and various translations between them ([Cod+11,Rab14], available online at [LATIN]). The resulting theory is then used as a meta-theory for importing the corresponding libraries. These imports tend to focus on translating the *statements* of theorems only, and pay less attention to the *proofs*, since proofs are often highly system-specific and difficult to translate without also reproducing all of the machinery of the system in question.

Li previously made an attempt to translate the IMPS math library to OMDoc in [Li02]. This is incomplete in a number of ways: In particular, Li’s approach did not handle quasi-constructors (a unique and important feature of IMPS, see Sect. 3.2) and other important aspects (like theory morphisms with additional assumptions, see Sect. 3.2), often because they are not represented in a useful manner in the internal data structures. Furthermore, Li was only able to check the *syntactic* validity of the generated XML, which makes the faithfulness of the translation difficult to judge. Finally, there have been a number of substantial representational changes from pure OMDoc to OMDoc/MMT, which renders this translation unusable.

Overview. This paper is a refined and condensed version of [Bet18], to which we refer for details and code. In Sect. 2, we recap all involved systems, including MMT, IMPS and OMDoc. After that, we outline the general idea and some theoretical as well as implementation-related specifics of the translation process in Sect. 3. Section 4 presents some applications for the OMDoc/MMT library and Sect. 5 concludes the paper.

2 Preliminaries

2.1 Preliminaries: LUTINS

LUTINS (pronounced as in French, short for “Logic of Undefined Terms for Inference in a Natural Style”) is the underlying logic of the IMPS system.

LUTINS is a variant of Church’s simple theory of types [Chu40]. It was developed to allow computerised mathematical reasoning that closely follows mathematical practise as performed by mathematicians “in the wild”. And since

standard mathematical reasoning often focuses on functions, their properties and operators on them, LUTINS allows for partial functions, and features a (partial) definite description operator as well as a system of subtypes.

LUTINS is a classical logic in the sense that it allows non-constructive reasoning, but non-classical in the sense that terms in LUTINS can be non-denoting. It also supports λ -notation for functions, an infinite hierarchy of function types for higher-order functions, and full quantification (existential and universal) over all function types.

Languages, Sorts, and Expressions. The notion of languages is central to LUTINS. They contain two classes of objects: sorts and expressions. Sorts denote (non-empty) domains of mathematical objects and expressions denote members of these domains. Expressions can be used to directly reference mathematical objects and to make statements about them using a LUTINS language given by a set of sort declarations and (sorted) constant declarations (see [FGT98]).

We differentiate between *atomic sorts* (e.g. `ind`, `zz`, ...) and *compound sorts*, the latter denoting the domain of n -ary functions for an arbitrary n (e.g. `[zz, ind]` for $n = 2$). Sorts may overlap, but they cannot be empty. Every language includes the base type \star (sometimes also defined as $*$ and always denoted as such in the implementation), denoting the set $\{T, F\}$ of standard truth values.

Sorts are also divided into two *kinds*, \star (read: *star* or *prop*) and ι (read: *ind*). A given sort α is of kind \star if either $\alpha = \star$ or α is a compound sort *into* \star (i.e. a compound sort of the form $[\alpha_1, \dots, \alpha_n, \star]$, sometimes also called a *predicate*). In *all* other cases α is of kind ι . This includes all atomic sorts except \star itself.

LUTINS allows for sorts to be defined as *subsorts* of other sorts in multiple ways. For instance, the natural numbers \mathbb{N} form a subsort of the real numbers \mathbb{R} and the continuous (real) functions a subsort of the functions from \mathbb{R} to \mathbb{R} .

Each atomic sort is assigned a unique *enclosing sort* by the language that defines it. This gives rise to a particular partial order on its sorts, which we will denote with \preceq and call “the subsort relation”. It is intended to denote set inclusion. A sort that is maximal in relation to \preceq is called a *type*. The type of a given sort α has the notation $\tau(\alpha)$.

Subsorting also applies to compound sorts. In particular, if $\sigma_0 \preceq \tau_0$ and $\sigma_1 \preceq \tau_1$, then $[\sigma_0, \sigma_1] \preceq [\tau_0, \tau_1]$. This makes subsorting in LUTINS *covariant* in its arguments, not *contravariant* as in settings with only total functions. The compound sort $[\sigma_0, \sigma_1]$ contains exactly those partial functions that are never defined outside of σ_0 and never return values outside σ_1 . For example, you could pass any real number to a function expecting a natural number (given that \mathbb{N} and \mathbb{R} have the same type). If the number is indeed not a natural number, the expression will be undefined (see below).

All of this is helpful for mechanised deduction because the subsort relation can give important information about the value of an expression, should it be defined. Furthermore, many theorems have constraints that can easily be expressed in terms of a subtype and the prover can be programmed to handle these with special algorithms.

Partial Functions, Undefined and Non-denoting Values. The stated goal of IMPS (and therefore LUTINS) is to allow for reasoning that is very close to mathematical practice. This means that there needs to be a way to deal with partial functions and undefined values since these make frequent appearances in chalk-and-whiteboard mathematics. For example, all of the terms $\frac{5}{0}$, $\sqrt{-3}$, $\ln(-4)$ are undefined in the standard theory of arithmetic over the real numbers.

Note that there is a subtle philosophical difference between a term that is “undefined” and one that is “non-denoting”. According to Farmer, a term is undefined if it is not assigned a “natural” meaning and non-denoting if it is not to be assigned any meaning at all. Often, an undefined term is also non-denoting, but it *can* still have a denotation. For example, the term $\frac{5}{0}$ does not have a “natural meaning” in standard real arithmetic, but is sometimes assigned a value in practice anyway. In particular, IMPS follows the approach of *partial valuation for terms but total valuation for formulas* (see [Far90] for more details). This means a term of type \star *always* has a denotation (if one of its constituents is undefined, that denotation is F).

Definite Description. One of the more prominent features of LUTINS is the possibility of reasoning with definite description via the ι (or *iota*) constructor. Given a variable v of sort α of *kind* ι (not to be confused with the constructor itself) and an unary predicate φ over α , the expression $\iota v : \alpha . \varphi(v)$ denotes the *unique* v , such that $\varphi(v)$, if there exists such a v . If there is no or more than one v that fulfils the predicate, the ι -expression is undefined.

For example, the expression $\iota x : \mathbb{R} . (0 \leq x) \wedge (x \cdot x = 2)$ denotes $\sqrt{2} \in \mathbb{R}$, while the expression $\iota x : \mathbb{R} . x \cdot x = 2$ is undefined.

Definite description can be very useful for dealing with functions, especially partial functions, which is why it is featured so prominently in IMPS.

2.2 Preliminaries: IMPS

IMPS is an interactive theorem prover developed by William Farmer, Joshua Guttman and Javier Thayer from 1990 to 1993 [TMI]. It was one of the influential systems in the era of automated reasoning.

One of the goals in developing IMPS was to create a mathematical system that gave computational support to mathematical techniques common among actual mathematicians.

The development of the IMPS system has been heavily influenced (see [FGT98]) by three insights into real-life mathematics:

- Mathematics emphasises the *axiomatic method*. The characteristics of mathematical structures are captured in axioms. Theorems are then derived from these axioms for *all* structures that satisfy the axioms. Often, what is needed for a proof is a clever change of perspective to see that one structure is indeed an instance of another theory, bringing additional theorems to bear.

- Many branches of mathematics emphasise *functions*, including partial functions. Moreover, the classes of objects studied may be nested, as are the integers and the real numbers; or overlapping, as are the bounded functions and the continuous functions.
- Mathematical proofs usually employ a mixture of both *formal inference* and *computation*.

Special attention is directed at the interplay of *computation* and *proof*. Farmer, Guttman and Thayer emphasise that, for example, a mathematician might devote considerable effort into proving lemmas that justify computational procedures¹ but are ultimately uninterested in the part of the derivation that is the “implementation” of these procedures.

Therefore, IMPS also allows for inferences based on sound computation and not merely formal inference. These are treated as atomic inferences, although a full formalisation in – for example – a Gentzen-style system might require hundreds or thousands of inference steps.

Little Theories. When following the axiomatic method to do mathematics – that is, logically reasoning from a given set of sentences in a formal language – there are two prominent approaches to choose from, which we will refer to as the “little theories” and “big theories” approach.

In the “big theories” version of the axiomatic method, all reasoning is carried out in one highly expressive axiomatic theory. The set of axioms selected is powerful enough, such that any model of them will contain all the mathematical objects that are of interest to us, and deduction from these powerful axioms will be enough to prove the relevant theorems in the theory. Popular examples for a “big” axiomatic theory would be ZFC or the Calculus of Inductive Constructions.

Contrasted with that, the “little theories” approach uses a number of different theories with smaller, less powerful sets of axioms, to develop mathematics in. For example, one theorem could be true for all semi-rings, while another is only true in the theories of *commutative* rings. Theorems are proved by logical derivation from the axioms of whatever theory supplies the necessary structure for the proof.

Both IMPS and MMT subscribe to the “little theories” approach to formal mathematics, a design choice that was informed by the fact that the little theories approach lends itself well to the mechanism of theory interpretations [FGT92].

Theories are the basic unit of representing mathematical knowledge in IMPS. In fact, Farmer (in [FGT98]) calls IMPS “a system for developing, exploring, and relating theories”.

Theory Morphisms. A *theory morphism* (sometimes also called a *theory interpretation*) is a translation between two theories that maps expression from the one theory to expressions in the other, with the additional property that theorems are always mapped to theorems [Far93, FGT98].

¹ [FGT98] gives the example of the algorithm for differentiating polynomials for this.

This is an integral part of the “little theories” approach as theory morphisms are the tool to use to make results of one theory available in the other.

It is also close to mathematical practice, since seeing one structure as an instance of another (and therefore bringing all theorems of the other structure into play) is often the critical insight in non-trivial mathematical proofs.

2.3 Preliminaries: OMDoc/MMT

OMDoc (short for **O**pen **M**athematical **D**ocuments) is a semantics-oriented markup format for STEM-related documents extending OpenMath developed by the KWARC work group (see [Koh06]). OMDoc/MMT [RK13] re-conceptualises the formal/modular fragment of OMDoc and greatly enhances its expressive power. OMDoc/MMT retains OMDoc’s three distinct levels for expressions of mathematical knowledge: **Object Level** Expressions (e.g. terms and formulae) expressed in OpenMath, **Declaration Level** Constants (functions, types, judgements) with an optional (object-level) type and/or definition and **Module Level** Theories and Views; sets of declarations that inhabit a common name-space and context.

Theories in OMDoc/MMT are structurally similar to theories in IMPS and can include other theories. Hence MMT-theories allow for library development in concordance to the little theories paradigm. Views in MMT behave (for all purposes relevant in this paper) analogously to theory morphisms in IMPS.

The MMT System. The OMDoc/MMT language is implemented in the MMT system [Rab18], which provides an API to handle OMDoc/MMT content and services such as type checking, rewriting of expressions and computation, as well as notation-based presentation of OMDoc/MMT content and a general infrastructure for inspecting and browsing libraries.

Since OMDoc/MMT avoids committing to a specific semantics or logical foundation, foundation-dependent services and features (e.g. type checking, presentation) are implemented using (foundation-independent) generic algorithms extensible by foundation-dependent calculus rules via plug-ins (e.g. for handling content imported from external systems such as IMPS).

Theory Graphs. Theories and theory morphisms naturally lead to *theory graphs*, with theories as vertices and morphisms as edges. In fact, OMDoc/MMT-theories and morphisms form a category, which is exploited by the MMT-system to induce and translate knowledge in/between theories analogously to IMPS.

The possible arrows in OMDoc/MMT are **inclusions**, which import all declarations from the domain to the co-domain, **views**, which are judgement-preserving maps from the declarations in the domain to expressions over the co-domain, **structures**, which are omitted for this paper because they do not occur in IMPS, and the **meta-theory**-relation, which behaves like an include for most purposes².

² The meta-theory-relation connects theories that live on different meta-levels; e.g. domain knowledge to its logical foundation and conversely the logical foundation to the logical framework it is formalised in.

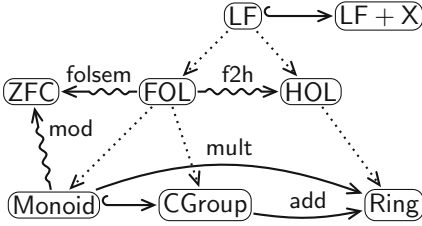


Fig. 1. Meta-levels in OMDoc/MMT

logical framework **LF** (see [HHP93]), since it is particularly well supported by the MMT system.

3 Implementation

The IMPS-to-OMDoc translator consists of two parts: the representation of LUTINS in the meta-logic and the programs to translate the sources against this.

3.1 The LUTINS Theory in LF

To formalise LUTINS in MMT, we use the logical framework **LF**, which provides a dependently typed lambda calculus with (i) two universes **type** and **kind** with **type:kind** and (ii) dependent function types $\prod_{x:A} T(x)$ (in **LF**-syntax: $\{x:A\}T(x)$). If T does not contain the variable x , this is the same as the function type $A \rightarrow T$. Dependent function types are inhabited by lambda expressions $\lambda x : A.t(x)$ (in **LF**-syntax: $[x:A]t(x)$). The usual rules in a lambda calculus (extensionality, beta-reduction, ...) hold.

To represent LUTINS, we created a **LF** meta-theory³ that, for every concept in the logic itself (like quantifiers, logical constructors, the primitive sorts, ...), has a corresponding constant (44 of them). Furthermore, we declare:

1. a new **LF**-type $\text{tp}:\text{type}$, which serves as the universe of maximal IMPS-sorts,
2. a function $\text{sort} : \text{tp} \rightarrow \text{type}$, and
3. a function $\text{exp} : \{A : \text{tp}\} \text{sort } A \rightarrow \text{type}$.

Given some maximal IMPS-sort **A**, the **LF**-type $\text{sort } A$ then serves as the type of all subsorts of that IMPS-type, and given a sort declaration $\mathbf{a} : \text{sort } A$, the type $\text{exp } A \ \mathbf{a}$ corresponds to the **LF**-type of all IMPS-expressions of sort \mathbf{a} .

We use the principles of *higher-order abstract syntax* to specify *binders* in IMPS. For example, consider an IMPS expression $\lambda x : A.t$, where the λ -constructor binds a new variable $x : A$. We formalise this behaviour by declaring the IMPS lambda to be an **LF** function `lambda`, that takes an **LF** lambda expression as

³ This formalisation is part of the LATIN foundations see <https://gl.mathhub.info/MMT/LATIN/blob/master/source/foundations/imps/lutins.mmt>.

An example graph is given in Fig. 1. Dotted lines represent the meta-theory-relation, hooked arrows are includes, squiggly arrows represent views, and the normal (labelled) arrows represent structures. The MMT system also provides a theory graph viewer (see [RKM17]), an example for which is given in Fig. 9. For our purposes, we fix as a foundation the logical framework **LF** (see [HHP93]), since it is particularly well supported by the MMT system.

argument which binds the variable x . As a result we get the **LF** expression `lambda ([x:A] t)` being the application of the function `lambda` to the **LF** function `[x:A]t`, effectively “embedding” an **LF** function on IMPS expressions as an IMPS function. Application in IMPS, quantifiers and other binders are treated analogously.

For propositional judgements (i.e. axioms and theorems) in IMPS, we use the *judgements-as-types* paradigm by introducing an operator `thm : exp bool → type`, assigning to each proposition a type which we can think of as the “type of proofs” for that proposition. Correspondingly, we consider a proposition A to be “true” if the type `thm A` is inhabited. Axioms correspond to undefined constants of type `thm A`, whereas theorems correspond to defined constants of that type, their definition being a proof (although proofs are omitted in this paper).

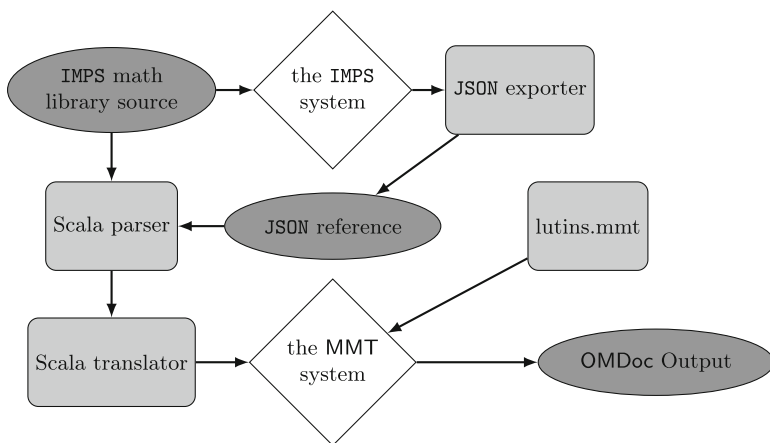


Fig. 2. Overview: *dark*: static files (sources and generated), *medium*: our contributions, *light*: independent systems

3.2 Translation

We now present the actual transformation process. It starts with IMPS library files⁴ and uses several software systems over a number of different steps, which are outlined below. Figure 2 gives a high-level schematic view of all involved systems and processes. The individual steps of the translation process are as follows:

- **Generate JSON from IMPS data structures:** For this, we modified Li’s exporter to export (the relevant aspects of) the internal data structures in IMPS directly to JSON, which is easy to read (for both human and machine) and gives us direct access to the data in the internal data structures, instead of an outdated OMDoc *translation* of those structures.

⁴ Which – like the original IMPS system – are written in the T language – a dialect of Scheme – and are hence often referred to simply as “T-files” in the following sections.

- **Import and combine JSON and IMPS sources:** Parsing from both IMPS library source files and JSON generated from internal data structures, gives the possibility of including more data in the translation, even data that is not represented on a symbolic level within IMPS (e.g. macetes or proofs).
- **Translate combined structures to MMT/OMDoc:** The last step uses the **LF**-implementation of LUTINS. In this form, they can also be type-checked by MMT to verify their correctness. The final OMDoc output is also generated by the MMT system, which always produces OMDoc in the *current* standard of the format.

```

(def-atomic-sort nn                ;; Name
 "lambda(x:zz, 0<=x)"            ;; Defining Expression
 (theory h-o-real-arithmetic)    ;; Home Theory
 (witness "0"))                 ;; Witness to show the sort non-empty

```

Fig. 3. IMPS source code: def-form defining the atomic sort `nn` via predicate

Def-Forms. IMPS source files contain information in so-called “def-forms” (short for “*definition forms*”). Each def-form is essentially the specification of one IMPS object, from constants, theories, languages to translations. Figure 3 shows an example.

Group	Amount	foundation	imps-math-library
foundational	17	414 (100%)	1918 (93.8%)
advanced	10	0 (0%)	119 (6.2%)
unused	5	0 (0%)	0 (0%)

Fig. 4. Survey results for usage of each def-form

To avoid unnecessary work in implementation, we surveyed the library to determine which def-forms were used how often. In Fig. 4 we consider a def-form “*unused*”, if it

does not appear in the library, even if it *is* supported by IMPS. We classify a def-form as “*foundational*” if it appears in the `foundation` sub-library. All other def-forms (called “*advanced*” in this context) were initially given low priority.

S-Expressions. There are different ways of representation in which IMPS displays mathematical objects to the user. One of the most important features of the JSON export mechanism is the export of mathematical expressions in *s-expression syntax* (as popularised by LISP) instead of *string syntax*.

For example, consider the axiom `commutative-law-for-addition` of the theory `h-o-real-arithmetic`. In string presentation, it is printed like this:

$$\text{forall}(y,x:rr, x+y=y+x)$$

In s-expression syntax, however, this axiom is printed as follows:

```
(forall ((rr y x)) (= (apply-operator + x y)
                      (apply-operator + y x)))
```

While the string representation might be more familiar to the human eye, s-expressions are considerably easier to parse mechanically and make dealing with binding strength and operator precedence unnecessary. They also simplify parsing function applications and quasi-constructors.

Quasi-Constructors. In addition to the LUTINS core logical constructors it is also possible for a user of IMPS to define additional constructor-like forms called “quasi-constructors”. These are implemented as “macros” or “abbreviations”.

For example, in IMPS, there exists the notion of quasi-equality: two expressions are quasi-equal if and only if they are either both undefined or are both defined with the same value. In mathematical notation, this would be captured by the following biconditional (where $x \downarrow$ means “ x is defined”):

$$E_1 \simeq E_2 \equiv (E_1 \downarrow \vee E_2 \downarrow) \supset E_1 = E_2$$

More precisely, a quasi-constructor consists of three elements: a *name* (something like `quasi-equals`), a list of variables (E_1 and E_2) and a schema (the right hand side above, see Fig. 5 for an example from the library).

In addition to the user-defined quasi-constructors, the IMPS system also has a small number of so-called “system quasi-constructors” that are hard-wired into the deductive machinery. Quasi-equality is one of them. Quasi-constructors are polymorphic in their schema variables, even if this polymorphism is not made explicit in the notation.

The translation of quasi-constructors turned out to be quite challenging. As Li states in [Li02], the corresponding lambda expressions for quasi-constructors are not represented as symbols in IMPS and can therefore not be translated into JSON directly, like other expressions.

```
(def-quasi-constructor I-IN
  "lambda(x:uu,a:sets[uu], #(a(x)))"
  (language indicators)
  (fixed-theories the-kernel-theory))
```

Fig. 5. The quasi-constructor `i-in`, as declared in IMPS

However, user-defined quasi-constructors are used *extensively* throughout the source. A survey of the T source files and the JSON output of just the `foundation` section identified 58 quasi-constructors used hundreds of times within the

library-section `imps-math-library`. Thus any effort to translate this library would be incomplete without a rigorous treatment of quasi-constructors.

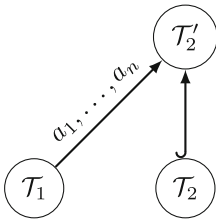
Instead of manually adding each individual quasi-constructor to the theory that defines it, or automatically resolving them immediately when parsed (which would need a lot of typing information not easily available at that stage of the translation) we decided to formulate one global **LF**-theory (called *QuasiLutins*) for them. There, we implemented all quasi-constructors as an instance of the same data type that also represents ordinary constructors (as seen in Fig. 6).

$$\begin{aligned} \text{inQC} : \{ A, \alpha : \text{sort } A \} \text{exp } \alpha &\longrightarrow \text{exp } (\text{sets}[\alpha]) \longrightarrow \text{exp } \text{bool} \\ &= [\text{U}, \text{u}, \text{x}, \text{a}] (\text{a} @ \text{x}) \downarrow \end{aligned}$$

Fig. 6. The same quasi-constructor, implemented in **LF**

This turned out to be the most effective and most faithful approach to the original sources, since the separation of theories makes clear what is part of the original LUTINS and what is not. It is also possible to stick to genuine polymorphism this way, without having to re-derive too much typing information during translation.

Theory Morphisms. IMPS translations (which are all *interpretations* in the IMPS library, i.e. all the obligations of the translation are theorems in the target theory) are translated as MMT *views*.



Some theory morphisms in IMPS (see Fig. 8 for an example showing the translation of groups to subgroups) have a collection of assumptions that need to be fulfilled (i.e. need to be theorems in the target theory for the morphism to be applicable). These assumptions can be used to state that certain conditions must be met (e.g. in the example from above, the target set (indicator function) must not be empty).

Fig. 7. Theory morphisms with axioms

Views in MMT, however, are not designed to have assumptions. To circumvent this obstacle, we create a copy of the target theory \mathcal{T}_2 , called \mathcal{T}'_2 that includes \mathcal{T}_2 , but also has all the assumptions associated with the theory morphism as additional axioms (see Fig. 7).

```
(def-translation GROUPS->SUBGROUP
  (source groups)
  (target groups)
  (assumptions
    "with(a:sets[gg], nonempty_indic_q{a})"
    "with(a:sets[gg], forall(g,h:gg, (g in a) and (h in a)
      implies (g mul h) in a)")
    "with(a:sets[gg], forall(g:gg, (g in a) implies (inv(g) in a))")
  (fixed-theories h-o-real-arithmetic)
  (sort-pairs
    (gg (indic "with(a:sets[gg], a))))
  (constant-pairs
    (mul "with(a:sets[gg], lambda(x,y:gg, if((x in a) and (y in a),
      x mul y, ?gg))))
    (inv "with(a:sets[gg], lambda(x:gg, if(x in a, inv(x), ?gg))))")
  force-under-quick-load
  (theory-interpretation-check using-simplification))
```

Fig. 8. IMPS source code (from “subgroups.t”): subgroup translation

4 Applications

Continued Theory Library Development. Translating the IMPS theory library into OMDoc/MMT format allows us to use the theories contained therein in future projects and enables the continuing development of other theories that build upon them (without depending directly on the IMPS system itself). They are now also available to other tools and automated methods (e.g. as data for machine-learning approaches to auto-formalisation).

Alignments. Using flexible alignments (see [Mül+17]) between different libraries (such as those of the PVS, HOL Light, and Mizar projects, for which there exist similar translation efforts), we can guide library developers to corresponding parts of other formalisations, give decent approximate translations of content across libraries, or help users more familiar with IMPS towards content of other systems by re-using notations that would otherwise be system specific.

OMDoc/MMT Services. With the OMDoc/MMT translation of the IMPS theory library, IMPS also gains access to library management facilities implemented at the OMDoc/MMT level. There are two ways to exploit this: publishing the translated IMPS libraries on a dedicated server, like the MathHub system, or running the OMDoc/MMT stack locally.

Browsing and Interaction. The transformed IMPS content can be browsed interactively in the document-oriented MathHub presentation pages (theories as active documents) and in the MMT web browser. Both allow interaction with the IMPS content via a generic Javascript-based interface.

Graph Viewer. The MMT system includes a theory graph viewer [RKM17] that allows interactive, web-based exploration of the OMDoc/MMT theory graphs. It builds on the `vis.js` JavaScript visualisation library, which uses the HTML5 canvas to layout and interact with graphs client-side in the browser.

The IMPS theory library relies substantially on theories as a structuring mechanism (as a consequence of taking the *little theories* approach), which makes a graph viewer particularly attractive. Figure 9 shows the full graph of the `foundation` library section, generated from only the OMDoc translated from IMPS⁵.

⁵ Note that the theories shown here are all part of the library; they are not duplicates created by the process from Fig. 7.

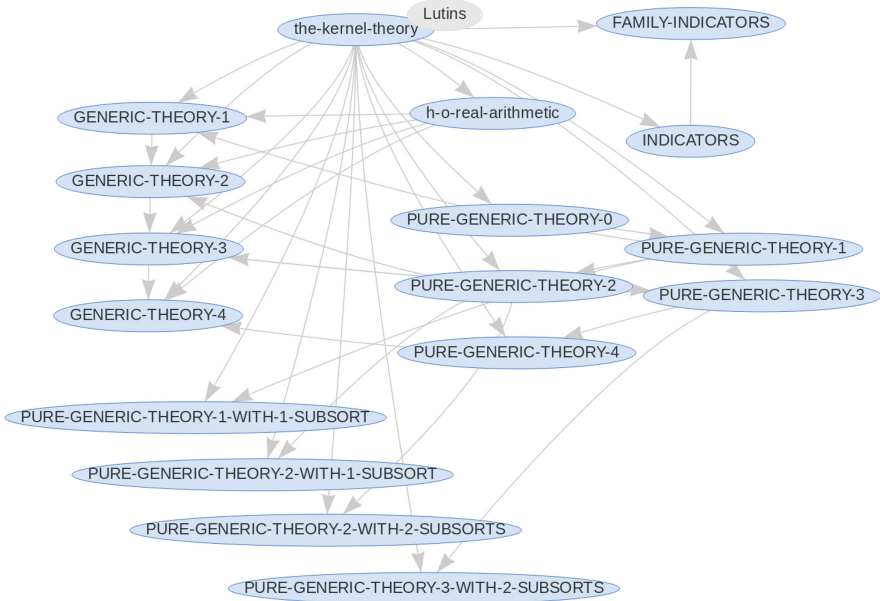


Fig. 9. Theory graph of the foundation section

5 Conclusion

We have developed a representation of the IMPS logic LUTINS and an automated translation of the IMPS mathematical theory library in the OMDoc/MMT format. This saves the IMPS library from becoming inaccessible and allows continued development and cross-fertilisation.

This information architecture is essential for system interoperability. In our case we have shown that we can use the language-independent MMT tool chain for IMPS. In particular, with the library browser and the theory graph viewer, we have instantiated two generic periphery systems for IMPS.

Future Work. Our results can also easily be extended to use LFX ($\mathbf{LF} + \mathbf{X}$, an extension to the \mathbf{LF} framework, see [LFX]) to give shallower (i.e. more structure-preserving) encodings of IMPS features without having to sacrifice the advantages of logical frameworks via the use of *structural features* (see [Ian17]).

Finally, in future efforts, we would like to extend the current export to also include proofs and macetes of the IMPS system in a usable format. For this to be possible, we would have to represent the IMPS proof calculus in \mathbf{LF} , and develop a \mathbf{LF} representation for proof commands. In our experience, both tactic-level proof scripts as well as full proofs are even harder to make interoperable than the statement level of libraries and may thus be less useful.

Software Sources. All software that is mentioned in this paper is available online: (i) **imps2json**: <https://gl.mathhub.info/IMPS/theories> (ii) **MMT extension**: <https://github.com/UniFormal/MMT/tree/imps> (iii) **MMT archive**: <https://gl.mathhub.info/IMPS/imps>.

Acknowledgments. The authors gratefully acknowledge financial support from DFG-funded project OAF: An Open Archive for Formalizations (KO 2428/13-1) and fruitful discussions and clarifications from Bill Farmer, Dennis Müller, and Florian Rabe.

References

- [Bet18] Betzendahl, J.: Translating the IMPS theory library to MMT/OMDoc. Master’s Thesis. Informatik, Universität Bielefeld, April 2018. <https://gl.kwarc.info/supervision/MSc-archive/blob/master/2018/jbetzendahl/thesisimps2omdoc.pdf>
- [Chu40] Church, A.: A formulation of the simple theory of types. *J. Symb. Log.* **5**, 56–68 (1940)
- [Cod+11] Codescu, M., Horozal, F., Kohlhase, M., Mossakowski, T., Rabe, F.: Project abstract: logic atlas and integrator (LATIN). In: Davenport, J.H., Farmer, W.M., Urban, J., Rabe, F. (eds.) *CICM 2011. LNCS (LNAI)*, vol. 6824, pp. 289–291. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22673-1_24. https://kwarc.info/people/frabe/Research/CHKMR_latinabs_11.pdf
- [Far90] Farmer, W.M.: A partial-function version of Church’s simple theory of types. *J. Symb. Log.* **55**, 1269–1291 (1990)
- [Far93] Farmer, W.M.: Theory interpretation in simple type theory. In: Heering, J., Meinke, K., Möller, B., Nipkow, T. (eds.) *HOA 1993. LNCS*, vol. 816, pp. 96–123. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58233-9_6
- [FGT92] Farmer, W.M., Guttman, J.D., Javier Thayer, F.: Little theories. In: Kapur, D. (ed.) *CADE 1992. LNCS*, vol. 607, pp. 567–581. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55602-8_192
- [FGT98] Farmer, W.M., Guttman, J.D., Javier Thayer, F.: *The IMPS 2.0 Users Manual*, 1st edn. The MITRE Corporation, Bedford (1998)
- [HHP93] Harper, R., Honsell, F., Plotkin, G.: A framework for defining logics. *J. Assoc. Comput. Mach.* **40**(1), 143–184 (1993)
- [Ian+13] Iancu, M., Kohlhase, M., Rabe, F., Urban, J.: The Mizar mathematical library in OMDoc: translation and applications. *J. Autom. Reason.* **50**(2), 191–202 (2013). <https://doi.org/10.1007/s10817-012-9271-4>
- [Ian17] Iancu, M.: *Towards flexiformal mathematics*. Ph.D. thesis. Bremen, Germany: Jacobs University (2017). <https://opus.jacobs-university.de/frontdoor/index/index/docId/721>
- [Koh+17] Kohlhase, M., Müller, D., Owre, S., Rabe, F.: Making PVS accessible to generic services by interpretation in a universal format. In: Ayala-Rincón, M., Muñoz, C.A. (eds.) *ITP 2017. LNCS*, vol. 10499, pp. 319–335. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66107-0_21. <http://kwarc.info/kohlhase/submit/itp17-pvs.pdf>

- [Koh06] Kohlhase, M.: OMDoc – An Open Markup Format for Mathematical Documents [version 1.2]. LNCS (LNAI), vol. 4180. Springer, Heidelberg (2006). <https://doi.org/10.1007/11826095>. <http://omdoc.org/pubs/omdoc1.2.pdf>
- [KPU16] Kaliszzyk, C., Pał, K., Urban, J.: Towards a Mizar environment for Isabelle: foundations and language. In: Avigad, J., Chlipala, A. (eds.) Proceedings of the 5th Conference on Certified Programs and Proofs (CPP 2016), pp. 58–65. ACM (2016). <https://doi.org/10.1145/2854065.2854070>
- [KR14] Kaliszzyk, C., Rabe, F.: Towards knowledge management for HOL light. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) CICM 2014. LNCS (LNAI), vol. 8543, pp. 357–372. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08434-3_26. http://kwarc.info/frabe/Research/KR_hollight_14.pdf
- [KS10] Krauss, A., Schropp, A.: A mechanized translation from higher-order logic to set theory. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 323–338. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14052-5_23
- [KW10] Keller, C., Werner, B.: Importing HOL light into Coq. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 307–322. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14052-5_22
- [LATIN] The LATIN Logic Atlas. <https://gl.mathhub.info/MMT/LATIN>. Accessed 02 June 2017
- [LFX] MathHub MMT/LFX Git Repository. <http://gl.mathhub.info/MMT/LFX>. Accessed 15 May 2015
- [Li02] Li, Y.: IMPS to OMDoc translation. Bachelor’s Thesis. McMaster University, August 2002
- [Mül+17] Müller, D., Gauthier, T., Kaliszzyk, C., Kohlhase, M., Rabe, F.: Classification of alignments between concepts of formal mathematical systems. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) CICM 2017. LNCS (LNAI), vol. 10383, pp. 83–98. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_7
- [OAF] The OAF Project & System. <http://oaf.mathhub.info>. Accessed 23 April 2015
- [OS06] Obua, S., Skalberg, S.: Importing HOL into Isabelle/HOL. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 298–302. Springer, Heidelberg (2006). https://doi.org/10.1007/11814771_27
- [Rab14] Rabe, F.: How to identify, translate, and combine logics? J. Log. Comput. **27**(6), 1753–1798 (2014). <https://doi.org/10.1093/logcom/exu079>
- [Rab18] Rabe, F.: MMT: a foundation-independent logical framework (2018). https://kwarc.info/people/frabe/Research/rabe_mmtsys_18.pdf
- [RK13] Rabe, F., Kohlhase, M.: A scalable module system. Inf. Comput. **230**, 1–54 (2013). <http://kwarc.info/frabe/Research/mmt.pdf>
- [RKM17] Rupprecht, M., Kohlhase, M., Müller, D.: A flexible, interactive theory-graph viewer. In: Kohlhase, A., Pollanen, M. (eds.) 12th Workshop on Mathematical User Interfaces, MathUI 2017 (2017). <http://kwarc.info/kohlhase/papers/mathui17-tgview.pdf>
- [TMI] Theorem Prover Museum - IMPS. <https://github.com/theoremprover-museum/imps>. Accessed 28 April 2018



Using the Isabelle Ontology Framework Linking the Formal with the Informal

Achim D. Brucker¹ , Idir Ait-Sadoune², Paolo Crisafulli³,
and Burkhart Wolff⁴

¹ The University of Sheffield, Sheffield, UK
a.brucker@sheffield.ac.uk

² CentraleSupélec, Paris, France
idir.aitsadoune@centralesupelec.fr

³ IRT-SystemX, Paris, France
paolo.crisafulli@irt-systemx.fr

⁴ Université Paris-Sud, Paris, France
wolff@lri.fr

Abstract. While Isabelle is mostly known as part of Isabelle/HOL (an interactive theorem prover), it actually provides a framework for developing a wide spectrum of applications. A particular strength of the Isabelle framework is the combination of text editing, formal verification, and code generation.

Up to now, Isabelle’s document preparation system lacks a mechanism for ensuring the structure of different document types (as, e.g., required in certification processes) in general and, in particular, mechanism for linking informal and formal parts of a document.

In this paper, we present Isabelle/DOF, a novel Document Ontology Framework on top of Isabelle. Isabelle/DOF allows for conventional typesetting *as well* as formal development. We show how to model document ontologies inside Isabelle/DOF, how to use the resulting meta-information for enforcing a certain document structure, and discuss ontology-specific IDE support.

Keywords: Ontology · Ontological modeling · Isabelle/DOF

1 Introduction

The linking of the *formal* to the *informal* is perhaps the most pervasive challenge in the digitization of knowledge and its propagation. This challenge incites numerous research efforts summarized under the labels “semantic web”, “data mining”, or any form of advanced “semantic” text processing. A key role in structuring this linking play *document ontologies* (also called *vocabulary* in the semantic web community [3]), i.e., a machine-readable form of the structure of documents as well as the document discourse. Such ontologies can be used for the scientific discourse within scholarly articles, mathematical libraries, and in the engineering discourse of standardized software certification documents [9, 10].

Further applications are the domain-specific discourse in juridical texts or medical reports. In general, an ontology is a formal explicit description of *concepts* in a domain of discourse (called *classes*), properties of each concept describing *attributes* of the concept, as well as *links* between them. A particular link between concepts is the *is-a* relation declaring the instances of a subclass to be instances of the super-class.

The main objective of this paper is to present Isabelle/DOF, a novel framework to *model* typed ontologies and to *enforce* them during document evolution. Based on Isabelle, ontologies may refer to types, terms, proven theorems, code, or established assertions. Based on a novel adaption of the Isabelle IDE, a document is checked to be *conform* to a particular ontology—Isabelle/DOF is designed to give fast user-feedback *during the capture of content*. This is particularly valuable in case of document changes, where the *coherence* between the formal and the informal parts of the content can be mechanically checked.

To avoid any misunderstanding: Isabelle/DOF is *not a theory in HOL* on ontologies and operations to track and trace links in texts, it is an *environment to write structured text* which *may contain* Isabelle/HOL definitions and proofs like mathematical articles, tech-reports and scientific papers—as the present one, which is written in Isabelle/DOF itself. Isabelle/DOF is a plugin into the Isabelle/Isar framework in the style of [14].

The plan of the paper is follows: we start by introducing the underlying Isabelle system (Sect. 2) followed by presenting the essentials of Isabelle/DOF and its ontology language (Sect. 3). It follows Sect. 4, where we present three application scenarios from the point of view of the ontology modeling. In Sect. 5 we discuss the user-interaction generated from the ontological definitions. Finally, we draw conclusions and discuss related work in Sect. 6.

2 Background: The Isabelle System

While Isabelle is widely perceived as an interactive theorem prover for HOL (Higher-order Logic) [11], we would like to emphasize the view that Isabelle is far more than that: it is the *Eclipse of Formal Methods Tools*. This refers to the “*generic system framework of Isabelle/Isar underlying recent versions of Isabelle. Among other things, Isar provides an infrastructure for Isabelle plugins, comprising extensible state components and extensible syntax that can be bound to ML programs. Thus, the Isabelle/Isar architecture may be understood as an extension and refinement of the traditional ‘LCF approach’, with explicit infrastructure for building derivative systems*” [14].

The current system framework offers moreover the following features:

- a build management grouping components into to pre-compiled sessions,
- a prover IDE (PIDE) framework [12] with various front-ends
- documentation - and code generators,
- an extensible front-end language Isabelle/Isar, and,
- last but not least, an LCF style, generic theorem prover kernel as the most prominent and deeply integrated system component.

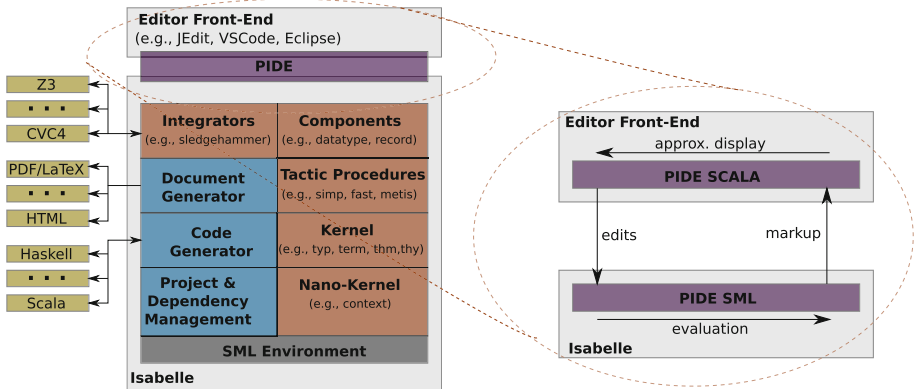


Fig. 1. The system architecture of Isabelle (left-hand side) and the asynchronous communication between the Isabelle system and the IDE (right-hand side).

The Isabelle system architecture shown in Fig. 1 comes with many layers, with Standard ML (SML) at the bottom layer as implementation language. The architecture actually foresees a *Nano-Kernel* (our terminology) which resides in the SML structure **Context**. This structure provides a kind of container called *context* providing an identity, an ancestor-list as well as typed, user-defined state for components (plugins) such as Isabelle/DOF. On top of the latter, the LCF-Kernel, tactics, automated proof procedures as well as specific support for higher specification constructs were built.

We would like to detail the documentation generation of the architecture, which is based on literate specification commands such as **section** ..., **subsection** ..., **text** ..., etc. Thus, a user can add a simple text:

```
text(This is a description.)
```

These text-commands can be arbitrarily mixed with other commands stating definitions, proofs, code, etc., and will result in the corresponding output in generated $\text{L}^{\text{T}}_{\text{E}}\text{X}$ or HTML documents. Now, *inside* the textual content, it is possible to embed a *text-antiquotation*:

```
text(According to the reflexivity axiom @{thm refl}, we obtain in  $\Gamma$ 
for @{term "fac 5"} the result @{value "fac 5"}.)
```

which is represented in the generated output by:

```
According to the reflexivity axiom  $x = x$ , we obtain in  $\Gamma$  for fac 5
the result 120.
```

where **refl** is actually the reference to the axiom of reflexivity in HOL. For the antiquotation @**{value "fac 5"}** we assume the usual definition for **fac** in HOL. Thus, antiquotations can refer to formal content, can be type-checked before being displayed and can be used for calculations before actually being typeset.

When editing, Isabelle’s PIDE offers auto-completion and error-messages while typing the above *semi-formal* content.

3 Isabelle/DOF

An Isabelle/DOF document consists of three components:

- the *ontology definition*, which is an Isabelle theory file with definitions for document-classes and all auxiliary datatypes.
- the *core* of the document itself which is an Isabelle theory importing the ontology definition. Isabelle/DOF provides an own family of text-element commands such as `title*`, `section*`, `text*`, etc., which can be annotated with meta-information defined in the underlying ontology definition.
- the *layout definition* for the given ontology exploiting this meta-information.

Isabelle/DOF is a novel Isabelle system component providing specific support for all these three parts. Note that the document core *may*, but *must* not use Isabelle definitions or proofs for checking the formal content—the present paper is actually an example of a document not containing any proof.

The document generation process of Isabelle/DOF is currently restricted to \LaTeX , which means that the layout is defined by a set of \LaTeX style files. Several layout definitions for one ontology are possible and pave the way that different *views* for the same central document were generated, addressing the needs of different purposes and/or target readers.

While the ontology and the layout definition will have to be developed by an expert with knowledge over Isabelle and Isabelle/DOF and the back end technology depending on the layout definition, the core is intended to require only minimal knowledge of these two. Document core authors *can* use \LaTeX commands in their source, but this limits the possibility of using different representation technologies, e.g., HTML, and increases the risk of arcane error-messages in generated \LaTeX .

The Isabelle/DOF ontology specification language consists basically on a notation for document classes, where the attributes were typed with HOL-types and can be instantiated by terms HOL-terms, i.e., the actual parsers and type-checkers of the Isabelle system were reused. This has the particular advantage that Isabelle/DOF commands can be arbitrarily mixed with Isabelle/HOL commands providing the machinery for type declarations and term specifications such as enumerations. In particular, document class definitions provide:

- a HOL-type for each document class as well as inheritance,
- support for attributes with HOL-types and optional default values,
- support for overriding of attribute defaults but not overloading, and
- text-elements annotated with document classes; they are mutable instances of document classes.

Attributes referring to other ontological concepts are called *links*. The HOL-types inside the document specification language support built-in types for

Isabelle/HOL `typ`'s, `term`'s, and `thm`'s reflecting internal Isabelle's internal types for these entities; when denoted in HOL-terms to instantiate an attribute, for example, there is a specific syntax (called *inner syntax antiquotations*) that is checked by Isabelle/DOF for consistency.

Document classes can have a **where**-clause containing a regular expression over class names. Classes with such a **where** are called *monitor classes*. While document classes and their inheritance relation structure meta-data of text-elements, monitor classes enforce structural organization of documents via the language specified by the regular expression enforcing a sequence of text-elements that must belong to the corresponding classes.

To start using Isabelle/DOF, one creates an Isabelle project (with the name `IsaDofApplications`):

```
isabelle DOF_mkroot -o scholarly_paper -t lncs -d IsaDofApplications
```

where the `-o scholarly_paper` specifies the ontology for writing scientific articles and `-t lncs` specifies the use of Springer's L^AT_EX-configuration for the Lecture Notes in Computer Science series. The project can be formally checked, including the generation of the article in PDF using the following command:

```
isabelle build -d . IsaDofApplications
```

4 Modeling Ontologies in Isabelle/DOF

In this section, we will use the Isabelle/DOF document ontology language for three different application scenarios: for scholarly papers, for mathematical exam sheets as well as standardization documents where the concepts of the standard are captured in the ontology. For space reasons, we will concentrate in all three cases on aspects of the modeling due to space limitations.

4.1 The Scholar Paper Scenario: Eating One's Own Dog Food

The following ontology is a simple ontology modeling scientific papers. In this Isabelle/DOF application scenario, we deliberately refrain from integrating references to (Isabelle) formal content in order demonstrate that Isabelle/DOF is not a framework from Isabelle users to Isabelle users only. Of course, such references can be added easily and represent a particular strength of Isabelle/DOF.

The first part of the ontology `scholarly_paper` (see Fig. 2) contains the document class definitions with the usual text-elements of a scientific paper. The attributes `short_title`, `abbrev` etc. are introduced with their types as well as their default values. Our model prescribes an optional `main_author` and a todo-list attached to an arbitrary text section; since instances of this class are mutable (meta)-objects of text-elements, they can be modified arbitrarily through subsequent text and of course globally during text evolution. Since `author` is a HOL-type internally generated by Isabelle/DOF framework and can therefore appear in the `main_author` attribute of the `text_section` class; semantic links between concepts can be modeled this way.

```

doc_class title =
  short_title :: "string option" <= None

doc_class subtitle =
  abbrev :: "string option" <= None

doc_class author =
  affiliation :: "string"

doc_class abstract =
  keyword_list :: "string list" <= None

doc_class text_section =
  main_author :: "author option" <= None
  todo_list :: "string list" <= "[]"

```

Fig. 2. The core of the ontology definition for writing scholarly papers.

The translation of its content to, e.g., Springer’s \LaTeX setup for the Lecture Notes in Computer Science Series, as required by many scientific conferences, is mostly straight-forward.

Figure 3 shows the corresponding view in the Isabelle/PIDE of the present paper. Note that the text uses Isabelle/DOF’s own text-commands containing the meta-information provided by the underlying ontology. We proceed by a definition of `introduction`’s, which we define as the extension of `text_section` which is intended to capture common infrastructure:

```

doc_class introduction = text_section +
  comment :: string

```

As a consequence of the definition as extension, the `introduction` class inherits the attributes `main_author` and `todo_list` together with the corresponding default values.

As a variant of the introduction, we could add here an attribute that contains the formal claims of the article—either here, or, for example, in the keyword list of the abstract. As type, one could use either the built-in type `term` (for syntactically correct, but not necessarily proven entity) or `thm` (for formally proven entities). It suffices to add the line:

```

claims :: "thm list"

```

and to extent the \LaTeX -style accordingly to handle the additional field. Note that `term` and `thm` are types reflecting the core-types of the Isabelle kernel. In a corresponding conclusion section, one could model analogously an achievement section; by programming a specific compliance check in SML, the implementation of automated forms of validation check for specific categories of papers is envisageable. Since this requires deeper knowledge in Isabelle programming, however, we consider this out of the scope of this paper.


```

1 1(*<+*)
2 theory IsaDofApplications
3 imports "Isabelle_DOF/ontologies/scholarly_paper"
4 begin
5 (*>*)
6
7 title*[tit::title]<Using The Isabelle Ontology Framework>
8 subtitle*[stit::subtitle]<Linking the Formal with the Informal>
9
10 text*[adb::author, email="'a.brucker@sheffield.ac.uk'", orcid="'0000-0002-6355-1200'",
11 affiliation="'University of Sheffield, Sheffield, UK'"] <Achim D. Brucker>
12 text*[auth2::author, email="'idir.aitsadoune@centralesupelec.fr'",
13 affiliation = "'CentraleSupélec, Paris, France'"] <Idir Ait-Sadoune>
14 text*[auth3::author, email="'paolo.crisafulli@irt-systemx.fr'",
15 affiliation = "'IRT-SystemX, Paris, France'"] <Paolo Crisafulli>
16 text*[bu::author, email="'wolff@lri.fr'",
17 affiliation="'Université Paris-Sud, Paris, France'"] <Burkhard Wolff>
18
19
20 text*[abs::abstract, keywordlist=["'Isabelle/Isar'", "'HOL'", "'Ontologies'"]]{*

```

Fig. 3. Ouroboros I: this paper from inside ...

We proceed more or less conventionally by the subsequent sections (Fig. 4) and finish with a monitor class definition that enforces a textual ordering in the document core by a regular expression (Fig. 5).

```

doc_class technical = text_section +
  definition_list :: "string list" <= "[]"

doc_class example = text_section +
  comment :: string

doc_class conclusion = text_section +
  main_author :: "author option" <= None

doc_class related_work = conclusion +
  main_author :: "author option" <= None

doc_class bibliography =
  style :: "string option" <= "'LNCS'"

```

Fig. 4. Various types of sections of a scholarly papers.

We might wish to add a component into our ontology that models figures to be included into the document. This boils down to the exercise of modeling structured data in the style of a functional programming language in HOL and to reuse the implicit HOL-type inside a suitable document class **figure**:

```

doc_class article =
  trace :: "(title + subtitle + author+ abstract +
            introduction + technical + example +
            conclusion + bibliography) list"
  where "(title ~ [subtitle] ~ {author}^+ ~ abstract ~
         introduction ~ {technical || example}^+ ~ conclusion ~
         bibliography)"

```

Fig. 5. A monitor for the scholarly paper ontology.

```

datatype placement = h | t | b | ht | hb
doc_class figure = text_section +
  relative_width :: "string" (* percent of textwidth *)
  src :: "string"
  placement :: placement
  spawn_columns :: bool <= True

```

Alternatively, by including the HOL-libraries for rationals, it is possible to use fractions or even mathematical reals. This must be counterbalanced by syntactic and semantic convenience. Choosing the mathematical reals, e.g., would have the drawback that attribute evaluation could be substantially more complicated.

```

326
327 figure* [fig1::figure, spawn_columns=False, relative_width="'90'",
328          src="'figures/Dogfood-Intro'"]
329 {* Ouroboros I: This paper from inside \dots *}
330

```

Fig. 6. Ouroboros II: figures ...

The document class `figure`—supported by the Isabelle/DOF text command `figure*`—makes it possible to express the pictures and diagrams in this paper such as Fig. 6.

4.2 The Math-Exam Scenario

The Math-Exam Scenario is an application with mixed formal and semi-formal content. It addresses applications where the author of the exam is not present during the exam and the preparation requires a very rigorous process, as the French *baccalauréat* and exams at The University of Sheffield.

We assume that the content has four different types of addressees, which have a different *view* on the integrated document

- the *setter*, i.e., the author of the exam,
- the *checker*, i.e., an internal person that checks the exam for feasibility and non-ambiguity,

- the *external examiner*, i.e., an external person that checks the exam for feasibility and non-ambiguity, and
- the *student*, i.e., the addressee of the exam.

The latter quality assurance mechanism is used in many universities, where for organizational reasons the execution of an exam takes place in facilities where the author of the exam is not expected to be physically present. Furthermore, we assume a simple grade system (thus, some calculation is required).

```

doc_class Author = ...
datatype Subject = algebra | geometry | statistical
datatype Grade = A1 | A2 | A3

doc_class Header = examTitle :: string
                  examSubject :: Subject
                  date      :: string
                  timeAllowed :: int -- minutes

datatype ContentClass = setter
                      | checker
                      | external_examiner
                      | student

doc_class Exam_item =
  concerns :: "ContentClass set"

doc_class Exam_item =
  concerns :: "ContentClass set"

type_synonym SubQuestion = string

```

Fig. 7. The core of the ontology modeling math exams.

The heart of this ontology (see Fig. 7) is an alternation of questions and answers, where the answers can consist of simple yes-no answers (QCM style check-boxes) or lists of formulas. Since we do not assume familiarity of the students with Isabelle (**term** would assume that this is a parse-able and type-checkable entity), we basically model a derivation as a sequence of strings (see Fig. 8).

In many institutions, it makes sense to have a rigorous process of validation for exam subjects: is the initial question correct? Is a proof in the sense of the question possible? We model the possibility that the *examiner* validates a question by a sample proof validated by Isabelle (see Fig. 9). In our scenario this sample proofs are completely *intern*, i.e., not exposed to the students but just additional material for the internal review process of the exam.

Using the \LaTeX package `hyperref`, it is possible to conceive an interactive exam-sheets with multiple-choice and/or free-response elements (see Fig. 10).

```

doc_class Answer_Formal_Step = Exam_item +
  justification :: string
  "term"       :: "string"

doc_class Answer_YesNo = Exam_item +
  step_label :: string
  yes_no    :: bool -- \isa{for\ checkboxes}

datatype Question_Type =
  formal | informal | mixed

doc_class Task = Exam_item +
  level  :: Level
  type   :: Question_Type
  subitems :: "(SubQuestion *
              (Answer_Formal_Step list + Answer_YesNo) list) list"
  concerns :: "ContentClass set" <= "UNIV"
  mark    :: int

doc_class Exercise = Exam_item +
  type   :: Question_Type
  content :: "(Task) list"
  concerns :: "ContentClass set" <= "UNIV"
  mark    :: int

```

Fig. 8. An exam can contain different types of questions.

```

doc_class Validation =
  tests :: "term list" <= "[]"
  proofs :: "thm list" <= "[]"

doc_class Solution = Exam_item +
  content :: "Exercise list"
  valids :: "Validation list"
  concerns :: "ContentClass set" <= "{setter,checker,external_examiner}"

doc_class MathExam=
  content :: "(Header + Author + Exercise) list"
  global_grade :: Grade
  where "{Author}^+ ~ Header ~ {Exercise ~ Solution}^+ "

```

Fig. 9. Validating exams.

With the help of the latter, it is possible that students write in a browser a formal mathematical derivation—as part of an algebra exercise, for example—which is submitted to the examiners electronically.

Exercise 2

Find the roots of the polynome: $x^3 - (6::'a) * x^2 + (5::'a) * x + (12::'a)$. Note the intermediate steps in the following fields and submit the solution.

From $x^3 - (6::'a) * x^2 + (5::'a) * x + (12::'a)$

have 1

have 2

have 3

finally show

Has the polynomial as many solutions as its degree ?

Fig. 10. A generated QCM fragment . . .

4.3 The Certification Scenario Following CENELEC

Documents to be provided in formal certifications (such as CENELEC 50126/50128, the DO-178B/C, or Common Criteria) can much profit from the control of ontological consistency: a lot of an evaluators work consists in tracing down the links from requirements over assumptions down to elements of evidence, be it in the models, the code, or the tests. In a certification process, traceability becomes a major concern; and providing mechanisms to ensure complete traceability already at the development of the global document will clearly increase speed and reduce risk and cost of a certification process. Making the link-structure machine-checkable, be it between requirements, assumptions, their implementation and their discharge by evidence (be it tests, proofs, or authoritative arguments), is therefore natural and has the potential to decrease the cost of developments targeting certifications. Continuously checking the links between the formal and the semi-formal parts of such documents is particularly valuable during the (usually collaborative) development effort.

As in many other cases, formal certification documents come with an own terminology and pragmatics of what has to be demonstrated and where, and how the trace-ability of requirements through design-models over code to system environment assumptions has to be assured.

In the sequel, we present a simplified version of an ontological model used in a case-study [8]. We start with an introduction of the concept of requirement (see Fig. 11). Such ontologies can be enriched by larger explanations and examples, which may help the team of engineers substantially when developing the central document for a certification, like an explication what is precisely the difference between an *hypothesis* and an *assumption* in the context of the evaluation standard. Since the PIDE makes for each document class its definition available by a simple mouse-click, this kind on meta-knowledge can be made far more accessible during the document evolution.

```

doc_class requirement = long_name :: "string option"

doc_class requirement_analysis = no :: "nat"
  where "requirement_item +"

doc_class hypothesis = requirement +
  hyp_type :: hyp_type <= physical (* default *)

datatype ass_kind = informal | semiformal | formal

doc_class assumption = requirement +
  assumption_kind :: ass_kind <= informal

```

Fig. 11. Modeling requirements.

For example, the term of category *assumption* is used for domain-specific assumptions. It has formal, semi-formal and informal sub-categories. They have to be tracked and discharged by appropriate validation procedures within a certification process, by it by test or proof. It is different from a hypothesis, which is globally assumed and accepted.

In the sequel, the category *exported constraint* (or *ec* for short) is used for formal assumptions, that arise during the analysis, design or implementation and have to be tracked till the final evaluation target, and discharged by appropriate validation procedures within the certification process, by it by test or proof. A particular class of interest is the category *safety related application condition* (or *srac* for short) which is used for *ec*'s that establish safety properties of the evaluation target. Their track-ability throughout the certification is therefore particularly critical. This is naturally modeled as follows:

```

doc_class ec = assumption +
  assumption_kind :: ass_kind <= (*default *) formal

doc_class srac = ec +
  assumption_kind :: ass_kind <= (*default *) formal

```

5 Ontology-Based IDE Support

We present a selection of interaction scenarios Sects. 4.1 and 4.3 with Isabelle/PIDE instrumented by Isabelle/DOF.

5.1 A Scholarly Paper

In Fig. 12 we show how hovering over links permits to explore its meta-information. Clicking on a document class identifier permits to hyperlink into

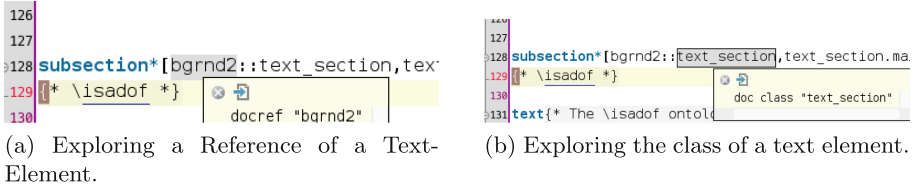


Fig. 12. Exploring text element.

the corresponding class definition (Fig. 13a); hovering over an attribute-definition (which is qualified in order to disambiguate; Fig. 13b).

An ontological reference application in Fig. 14: the ontology-dependant antiquotation `@{example ...}` refers to the corresponding text-elements. Hovering allows for inspection, clicking for jumping to the definition. If the link does not exist or has a non-compatible type, the text is not validated.

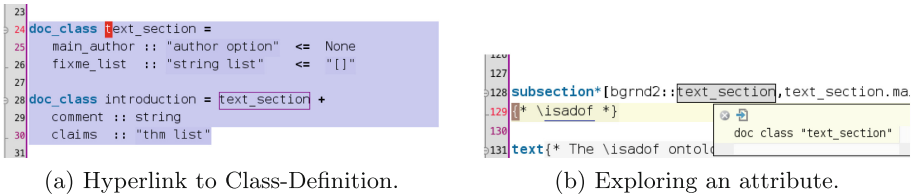


Fig. 13. Hyperlinks.

5.2 CENELEC

The corresponding view in Fig. 15 shows core part of a document, coherent to the Sect. 4.3. The first sample shows standard Isabelle antiquotations [13] into formal entities of a theory. This way, the informal parts of a document get “formal content” and become more robust under change.

The subsequent sample in Fig. 16 shows the definition of an *safety-related application condition*, a side-condition of a theorem which has the consequence that a certain calculation must be executed sufficiently fast on an embedded

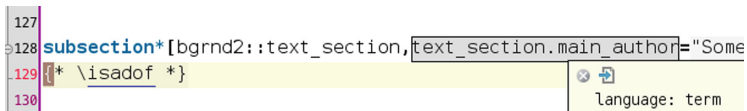


Fig. 14. Exploring an attribute (hyperlinked to the class).

```

1035 text{"
1036 The resolution of time, distance, speed and acceleration data, in International System Unit,
1037 shall be:
1038   • @term Time: 10$^{-2}$s
1039     the resolution needed for calculation.
1040   • @term Distance: 10$^{-3}$m (i.e. 1mm)
1041   • @const Speed: 1.3 x 10$^{3}$m/s (i.e. 0.005 km/h)
1042   • @const Acceleration: 0.005m/s$^{2}$
1043   • @const Jerk
1044   constant "Odo_ReqAna.Acceleration"
1045 The precision of calculations and propagated to the external
1046 interface data.

```

Fig. 15. Standard antiquotations referring to theory elements.

```

814 text*[enough_samples::srac]{* Note that the theorem above establishes a constraint between
815 @const w_circ, @const tpw, @const Speed_max and sample_frequency; since this
816 exported constraint is fundamental for the safe functioning of odometer and therefore
817 a safety-related exported application constraint. It is formally expressed as follows:
818 *}
819

```

Fig. 16. Defining a SRAC reference . . .

device. This condition can not be established inside the formal theory but has to be checked by system integration tests.

Now we reference in Fig. 17 this safety-related condition; however, this happens in a context where general *exported constraints* are listed. Isabelle/DOF's checks establish that this is legal in the given ontology.

This example shows that ontological modeling is indeed adequate for large technical, collaboratively developed documentations, where modifications can lead easily to incoherence. The current checks help to systematically avoid this type of incoherence between formal and informal parts.

```

822
823 text{" Summing up, the property that the odometer provides sufficient sampling
824 precision --- meaning no wheel encodings were ``lost'' compared to any sampling done with
825 a higher sampling rate --- can be established under the set of general hypothesis captured
826 in @docref <general_hyps> (formally expressed in @thm normally_behaved_distance_function_def))
827 and the SRAC @ec[enough_samples] formally expressed by @thm srac_def. *}
828

```

Fig. 17. Using a SRAC as EC document reference.

6 Conclusion and Related Work

We have demonstrated the use of Isabelle/DOF, a novel ontology modeling and enforcement IDE deeply integrated into the Isabelle/Isar Framework. The two most distinguishing features are

- Isabelle/DOF and its ontology language are a strongly typed language that allows for referring (albeit not reasoning) to entities of Isabelle/HOL, most notably types, terms, and (formally proven) theorems, and

- Isabelle/DOF is supported by the Isabelle/PIDE framework; thus, the advantages of an IDE for text-exploration (which is the type of this link? To which text element does this link refer? Which are the syntactic alternatives here?) were available during editing instead of a post-hoc validation process.

Of course, a conventional batch-process also exists which can be used for the validation of large document bases in a conventional continuous build process. This combination of formal and semi-informal elements, as well as a systematic enforcement of the coherence to a document ontology of the latter, is, as we believe, novel and offers a unique potential for the semantic treatment of scientific texts and technical documentations.

To our knowledge, this is the first ontology-driven framework for editing mathematical and technical documents that focuses particularly on documents mixing formal and informal content—a type of documents that is very common in technical certification processes. We see mainly one area of related works: IDEs and text editors that support editing and checking of documents based on an ontology. There is a large group of ontology editors (e.g., Protégé [5], Fluent Editor [1], NeOn [2], or OWLGrEd [4]). With them, we share the support for defining ontologies as well as auto-completion when editing documents based on an ontology. While our ontology definitions are currently based on a textual definition, widely used ontology editors (e.g., OWLGrEd [4]) also support graphical notations. This could be added to Isabelle/DOF in the future. A unique feature of Isabelle/DOF is the deep integration of formal and informal text parts. The only other work in this area we are aware of is rOntorium [6], a plugin for Protégé that integrates R [7] into an ontology environment. Here, the main motivation behind this integration is to allow for statistically analyze ontological documents. Thus, this is complementary to our work.

Isabelle/DOF in its present form has a number of technical short-comings as well as potentials not yet explored. On the long list of the short-comings is the fact that strings inside HOL-terms do not support, for example, Unicode. For the moment, Isabelle/DOF is conceived as an add-on for Isabelle/HOL; a much deeper integration of Isabelle/DOF into Isabelle could increase both performance and uniformity. Finally, different target presentation (such as HTML) would be highly desirable in particular for the math exam scenarios. And last but not least, it would be desirable that PIDE itself is “ontology-aware” and can, for example, use meta-information to control read- and write accesses of *parts* of documents.

Availability. The implementation of the framework, the discussed ontology definitions, and examples are available at https://git.logicalhacking.com/HOL-OCL/Isabelle_DOF/.

Acknowledgement. This work was partly supported by the framework of IRT SystemX, Paris-Saclay, France, and therefore granted with public funds within the scope of the Program “Investissements d’Avenir”.

References

1. Fluent editor (2018). <http://www.cognitum.eu/Semantics/FluentEditor/>
2. The neon toolkit (2018). <http://neon-toolkit.org>
3. Ontologies (2018). <https://www.w3.org/standards/semanticweb/ontology>
4. Owlged (2018). <http://owlged.lumii.lv/>
5. Protégé (2018). <https://protege.stanford.edu>
6. R language package for fluent editor (rOntorion) (2018). <http://www.cognitum.eu/semantics/FluentEditor/rOntorionFE.aspx>
7. Adler, J.: R in a Nutshell. O'Reilly, Sebastopol (2010)
8. Bezzecchi, S., Crisafulli, P., Pichot, C., Wolff, B.: Making agile development processes fit for V-style certification procedures. In: ERTS Conference Proceedings, ERTS 2018 (2018)
9. Boulanger, J.L.: CENELEC 50128 and IEC 62279 Standards. Wiley-ISTE, Boston (2015). The reference on the standard
10. Common Criteria: Common criteria for information technology security evaluation (version 3.1), part 3: security assurance components (2006). [CCMB-2006-09-003](https://www.common-criteria.org/CCMB-2006-09-003)
11. Nipkow, T., Wenzel, M., Paulson, L.C. (eds.): Isabelle/HOL: A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002). <https://doi.org/10.1007/3-540-45949-9>
12. Wenzel, M.: Asynchronous user interaction and tool integration in Isabelle/PIDE. In: Klein, G., Gamboa, R. (eds.) ITP 2014. LNCS, vol. 8558, pp. 515–530. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08970-6_33
13. Wenzel, M.: The Isabelle/Isar reference manual (2017). Part of the Isabelle distribution
14. Wenzel, M., Wolff, B.: Building formal method tools in the Isabelle/Isar framework. In: Schneider, K., Brandt, J. (eds.) TPHOLs 2007. LNCS, vol. 4732, pp. 352–367. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74591-4_26



Automated Symbolic and Numerical Testing of DLMF Formulae Using Computer Algebra Systems

Howard S. Cohl¹(✉), André Greiner-Petter², and Moritz Schubotz²

¹ Applied and Computational Mathematics Division,
National Institute of Standards and Technology, Mission Viejo, CA, USA
howard.cohl@nist.gov

² Department of Computer and Information Science, University of Konstanz,
Konstanz, Germany
{andre.greiner-petter,moritz.schubotz}@uni-konstanz.de

Abstract. We have developed an automated procedure for symbolic and numerical testing of formulae extracted from the National Institute of Standards and Technology (NIST) Digital Library of Mathematical Functions (DLMF). For the NIST Digital Repository of Mathematical Formulae, we have developed conversion tools from semantic \LaTeX to the Computer Algebra System (CAS) MAPLE which relies on Youssef's part-of-math tagger. We convert a test data subset of 4,078 semantic \LaTeX DLMF formulae extracted from the DLMF to the native CAS representation and then apply an automated scheme for symbolic and numerical testing and verification. Our framework is implemented using Java and MAPLE. We describe in detail the conversion process which is required so that the CAS is able to correctly interpret the mathematical representation of the formulae. We describe the improvement of the effectiveness of our automated scheme through incremental enhancement (making more precise) of the mathematical semantic markup for the formulae.

1 Problem and Current State

The National Institute of Standards and Technology (NIST) Digital Library of Mathematical Functions [1] (DLMF) is an online digital mathematics library which focuses on special functions and orthogonal polynomials. The DLMF is special in that it has been written like a book, but is published online as a digital resource. The DLMF contains a table of contents which links to specific chapters (See Table 1 for a summary of the 36 DLMF Chapters) each of which has been written and edited by mathematicians who are experts in their specific field of focus. Furthermore, the DLMF is constantly being maintained (edited, corrected, updated) by a team of mathematicians, physicists and computer scientists who

are determined to maintain and preserve the high standard of mathematical accuracy and exposition.

The DLMF has been developed in such a way that mathematical metadata is constantly being improved and restructured. For the organizational structure, semantic metadata has been incorporated at the document, chapter, section, subsection, and paragraph levels. An effort has been made to make more precise the nature of the objects which appear in the DLMF. The DLMF's semantic realization of *mathematical* content, has very much in common with the way mathematics is written using \LaTeX , for the preparation of journal publications and actual physical books. In fact, for the construction of the DLMF, Bruce Miller has developed and actively maintains \LaTeXML [2], a program which is able to process a certain flavor of \LaTeX and from it, generate a fully functional website. We refer to this flavor of \LaTeX as semantic \LaTeX . The website is then generated using \LaTeXML , from a collection of semantic \LaTeX source documents, and from other documents including Cascading Style Sheets (CSS), WebGL (Web Graphics Library) controls, Mathematical Markup Language (MATHML), \LaTeX style files, \LaTeXML binding files (\LaTeXML analogue of a style or class file), Java, Javascript, etc. (see for instance [3]). The outcome is a highly sophisticated digital platform for accessing mathematical content on the web.

One main difference between \LaTeX and semantic \LaTeX , is the latter's ability to encapsulate mathematical knowledge in such a way that it can then be extracted by a computer program, namely \LaTeXML , to generate correctly formatted MATHML (and other programs [4]). \LaTeXML is then able to interpret the content written in semantic \LaTeX in order to generate presentation (and content) MATHML. This is then used to construct the DLMF website and to, for instance, display metadata associated with the mathematical content. The mathematical semantic contents of the DLMF includes mathematics which appears within the main body of the text, formulae (in various formulae environments), tables (in which mathematics is displayed in a tabular environment), figures (in which mathematics is visualized), operators, functions/symbols, variables (see [5] for a nice description), etc.

Even though there currently exist software to convert \LaTeX expressions into Computer Algebra System (CAS) formats, these in practice are only really effective if one is dealing with elementary functions defined in terms of compositions of powers-laws, sums, differences, products, quotients, trigonometric/hyperbolic functions, and exponential/logarithmic functions. If one is dealing with more complicated special functions like Bessel functions, hypergeometric functions, Airy functions, elliptic integrals, elliptic functions, etc., then effective conversion software from \LaTeX to CAS really do not exist, and our implementation described below is the first to be able to handle situations such as these. Also, several differences in CAS vs. DLMF implementations must be captured in any effective translation, including: (1) the usage of $m = k^2$ for elliptic integrals/functions; (2) branch cuts for Legendre/Ferrers functions (and other functions); and (3) differences in normalizations of special functions. If one is unable to capture these subtleties in a translation process for the DLMF, which

require careful and detailed mathematical knowledge and implementation, then the translation will fail.

In a previous paper [6], we described the conversion process we have developed to input semantic \LaTeX and output a corresponding MAPLE^1 CAS representation. Not surprisingly, this conversion is continually in development. New ideas are being implemented in order to increase the semantic enhancement of the original source (see for instance [7]), as well as to improve the effectiveness of our conversion software. In order to do this, we are also contributing in order to assist in the development of the part-of-math (POM) tagger [5], which our conversion program relies upon.

In this paper, we describe the process of extracting mathematical formulae from the DLMF, and then take advantage of the powerful mathematical semantic coverage of semantic \LaTeX used in the DLMF to convert to CAS representations whose sole purpose is to drive an automated scheme for verification of DLMF formulae. In order to improve the precision of our DLMF formulae testing reported on in [6], we did the following: (1) updated our test formulae dataset; (2) fixed numerous translation bugs; (3) implemented a new method for handling constraints; and (4) added configuration files which make it easier to change the setup of our symbolic and numerical testing.

2 Extraction of DLMF Formulae

2.1 First Extraction Scan

In the first extraction scan, we extract mathematical formulae from 36 \LaTeX source files which summarize the content of the 36 DLMF chapters. DLMF formulae are extracted out of formula environments, which are (some custom) $\{\text{equation}\}$, $\{\text{equationmix}\}$, $\{\text{equationgroup}\}$, $\{\text{align}\}$. Each formula in the output text file, is represented in semantic \LaTeX , as a single string on one line of the output text file. In order to accomplish this, all lines of the source \LaTeX file are merged, except the following character strings are removed: comments; space (and other) formatting commands (such as \backslash , (used to insert a small horizontal space), $\backslash!$ (used to remove horizontal space), $\backslash\backslash[. . .]$ (used to introduce line breaks), $\&$ (used for alignment purposes), $\backslash*$ (used to force line breaking on multiplication), etc.); $\backslash\text{MarkNotation}$, $\backslash\text{origref}$, $\backslash\text{note}$, $\backslash\text{lxRefDeclaration}$, $\backslash\text{index}$, $\backslash\text{source}$, $\backslash\text{authorproof}$.

Along with the semantic \LaTeX source for the formula, we also extract two pieces of metadata associated with that formula, a $\backslash\text{constraint}$ environment, and also a $\backslash\text{label}$ environment. The $\backslash\text{constraint}$ and $\backslash\text{label}$ environments

¹ The mention of specific products, trademarks, or brand names is for purposes of identification only. Such mention is not to be interpreted in any way as an endorsement or certification of such products or brands by the National Institute of Standards and Technology, nor does it imply that the products so identified are necessarily the best available for the purpose. All trademarks mentioned herein belong to their respective owners.

Table 1. Summary of DLMF chapters (Version 1.0.13, 2016-09-16, see Sect. 2.1) with corresponding 2-letter codes (2C), chapters numbers (C#), chapter names of the DLMF chapters, chapter formulas extracted from the first extraction scan (F1), and second extraction scan (F2).

2C	C#	Chapter Name	F1	F2	2C	C#	Chapter Name	F1	F2
AL	1	Algebraic and Analytic Methods	624	85	OP	18	Orthogonal Polynomials	590	253
AS	2	Asymptotic Approximations	375	56	EL	19	Elliptic Integrals	674	391
NM	3	Numerical Methods	428	41	TH	20	Theta Functions	115	84
EF	4	Elementary Functions	529	466	MT	21	Multidimensional Theta Functions	62	7
GA	5	Gamma Function	181	56	JA	22	Jacobian Elliptic Functions	273	181
EX	6	Exponential, Logarithmic, Sine, and Cosine Integrals	110	44	WE	23	Weierstrass Elliptic and Modular Functions	199	70
ER	7	Error Functions, Dawson's and Fresnel Integrals	158	80	BP	24	Bernoulli and Euler Polynomials	202	60
IG	8	Incomplete Gamma and Related Functions	272	120	ZE	25	Zeta and Related Functions	176	50
AI	9	Airy and Related Functions	250	160	CM	26	Combinatorial Analysis	215	54
BS	10	Bessel Functions	904	355	NT	27	Functions of Number Theory	132	24
ST	11	Struve and Related Functions	168	83	MA	28	Mathieu Functions and Hill's Equation	442	186
PC	12	Parabolic Cylinder Functions	205	100	LA	29	Lamé Functions	251	72
CH	13	Confluent Hypergeometric Functions	391	174	SW	30	Spheroidal Wave Functions	161	46
LE	14	Legendre and Related Functions	294	173	HE	31	Heun Functions	177	33
HY	15	Hypergeometric Function	206	174	PT	32	Painlevé Transcendents	342	59
GH	16	Generalized Hypergeometric Functions & Meijer G -Function	107	32	CW	33	Coulomb Functions	210	92
QH	17	q -Hypergeometric and Related Functions	185	96	TJ	34	$3j$, $6j$, $9j$ Symbols	68	30
					FM	35	Functions of Matrix Argument	62	1
					IC	36	Integrals with Coalescing Saddles	181	90

contain any constraints and \LaTeX labels which have been directly associated with the formula within the semantic \LaTeX formula environment in which it was extracted. If more than one formula is associated with a `\label` environment, we inherit that `\label` to all of those formulae.

There are also certain \LaTeX commands or replacement macros which are specifically defined in the preamble of certain chapter source files (e.g., `\newcommand{\eptipm}[1]{\expe^{\pm2 \pi i i \#1}}` which occur in 65 separate lines). These are replaced in order for our translator to be able to correctly understand the corresponding derived semantic \LaTeX (e.g., in Chapters AI, CH the entity `\gamma` is replaced with the macro `\EulerConstant [1, (5.2.3)]2`; in Chapters JA, MA, LA, EL, the entities K and K' are replaced with the semantic macros `\CompEllIntKk@@{k}` and `\CompEllIntKk@@{k}` respectively³, etc.). See Table 1 for a description of the DLMF chapter codes. An

² The macro `\EulerConstant` is just one of many (currently unpublished) semantic DLMF \LaTeX semantic macros which have been developed by Bruce Miller for utilization in the DLMF. In fact, the macro names used in this manuscript, have been more recently updated. However, the macro set we are utilizing, is for those in usage as of 9/16/2016.

³ These semantic macros represent the complete and complementary elliptic integrals of the first kind [1, (19.2.8-9)].

attempt is made to remove punctuation marks at the end of each formula since these are really not a part of the formula itself. One final time consuming task is the conversion, wherever required, for the replacement of the entities i , e , and π and the \LaTeX command $\backslash\pi$ to the semantic macros $\backslash\text{iunit}$, $\backslash\text{expe}$, and $\backslash\text{cpi}$ ⁴. The result of this first round is to extract 9,919 formulae from the DLMF (see the F1 entries in Table 1 for a chapter specific description of this).

2.2 Second Extraction Scan

In a second round of post processing, a subset of formulae are culled from the first extraction scan (described in the previous section). The formulae which are removed are those which contain the following \LaTeX commands $\backslash\text{sum}$, $\backslash\text{int}$, $\backslash\text{prod}$, $\backslash\text{lim}$, $\backslash\text{dots}$ (including all variants), $\backslash\text{sim}$, or the semantic macros $\backslash\text{BigO}$, $\backslash\text{littleo}$, $\backslash\text{fDiff}$, $\backslash\text{bDiff}$, $\backslash\text{cDiff}$, $\backslash\text{asympt}$, or the environments $\{\text{cases}\}$, $\{\text{array}\}$, $\{\text{bmatrix}\}$, $\{\text{vmatrix}\}$, $\{\text{Bmatrix}\}$, $\{\text{pmatrix}\}$, $\{\text{Matrix}\}$, $\{\text{Lattice}\}$. This is because either our CAS translator is unable to translate these effectively, or the CAS is unable to verify mathematical objects such as these. See Sect. 2.3 below, for ongoing strategies for handling formulae which fall into these categories. Once these strategies are fully implemented, then we will no longer need to cull formulae such as these.

We have also utilized a new semantically enhanced macro $\backslash\text{Wron}$ for 72, two-argument Wronskian relations [1, (1.13.4)], so that the variable which is differentiated against is precisely specified in an updated macro call. In fact, we have continued to develop many new semantically enhanced \LaTeX macros (such as $\backslash\text{Wron}$) which are not in use in the DLMF, but are in use for the NIST Digital Repository of Mathematical Formulae (see [8] for more details about this). We also split equations with multiple equal = signs (e.g., $a = b = c$ is split into two formulae $a = b$ and $a = c$). Furthermore, we split environments which contain $\backslash\text{pm}$ (plus or minus) and $\backslash\text{mp}$ (minus or plus) commands into two separate formulae, each with their correct sign. We also remove mathematical expressions, which do not contain any of the relation commands $=$, $<$, $>$, $\backslash\text{ne}$, $\backslash\text{le}$, $\backslash\text{ge}$, $\backslash\text{to}$, $\backslash\text{equiv}$, since their non-existence implies that there is no logical component for the mathematical expression to be verified against. The result of this second round is to extract 4,078 formulae from the DLMF (see the F2 entries in Table 1 for a chapter specific description of this).

2.3 Ongoing Semantic Enhancement for Second Extraction Scan

As mentioned above, we are currently not able to convert \LaTeX commands such as $\backslash\text{sum}$, $\backslash\text{int}$, $\backslash\text{prod}$, $\backslash\text{lim}$, which represent sums, integrals, products and limits, respectively. In this case, one draws (presentation) subscripts and superscripts in order to provide critical semantic information for these mathematical

⁴ These semantic macros represent the mathematical constants i , e and π [1, (1.9.1), (4.2.11), (3.12.1)].

operators. In this case, there is no certainty in the specificity of the mathematical operation.

Another notation in which semantic capture is challenged, is with the prime (e.g., f'), double prime (e.g., f''), triple prime (e.g., f''') or superscript parenthetical (e.g., $f^{(iv)}$) notations for differentiation a given number of times. This is the same issue that was encountered previously with the Wronskian relation. In order for a CAS to be able to translate expressions such as these, we must provide the variable that one is differentiating with respect to. This is an example where a human is often able to read a formula, and by knowing the context, be able to surmise what the variable is that one is differentiating with respect to. Sometimes there is more than one variable in an expression that one is differentiating with respect to, and if one does not provide the variable, then a translator is unable to disambiguate the expression. It is interesting to note that we were able to enhance the semantics of 74 Wronskian relations by rewriting the macro so that it included the variable that derivatives are taken with respect to as a parameter. A similar semantic enhancement is possible for another 284 formulae where the potentially ambiguous prime notation ‘ $'$ ’ is used for derivatives.

The powerful tool used by the DLMP, \LaTeXML , is often able to guess a syntax tree when the \LaTeX commands `\sum`, `\int`, `\prod`, and `\lim` are used. This is due to, for instance, the utilization of the semantic \LaTeX `\diff` macro for integrals. This is quite effective, since it makes available the variable that one is integrating with respect to. For instance, one may mark up a definite integral such as

$$\int_a^b f(x, y) dx.$$

by using semantic \LaTeX as follows `\int_{a}^{b} f(x, y) \diff{x}`, which provides the variable of integration x . This information is essential for disambiguation of the integration variable, and therefore for converting the definite integral to a CAS representation. Such content is not always easily accessible. For instance, one is still unable to easily guess the variable of differentiation when prime (etc.) notations for derivatives are used. One is able to facilitate the extraction of semantic content in formulae which contain mathematical \LaTeX such as described above, by making the semantic content more easily available. We would like to see this type of semantic content made more fully available for translators, and this is an ongoing effort.

It is possible to incrementally enhance mathematical semantic expressions through optimization and broader usage and development of semantic \LaTeX macros. Take for instance the formula with the label `{eq:OP.CP.SV.LR.1a}` [1, (18.8.21)], given by

$$\lim_{\beta \rightarrow \infty} \text{JacobiP}(\alpha, \beta, n) @ \{1 - (\frac{2x}{\beta})\} = \text{LaguerreL}[\alpha, n] @ \{x\}.$$

In this example, our semantic enhancement process is made more effective by developing and utilizing a new mathematical semantic \LaTeX macro with separate parameters and arguments (before and after the `@` sign respectively) which is described as

$$\backslash\text{Lim}\{\#1\}\{\#2\}@{\#3} := \lim_{\#1 \rightarrow \#2} \#3.$$

Here the first parameter of the macro call #1 is the variable the limit is taken over, the second parameter of the macro call #2 is the destination of the limit for #1, and the argument #3, is the expression that the limit is taken over. Utilizing this new macro, the above formula is then written with enhanced semantics as:

$$\backslash\text{Lim}\{\backslash\text{beta}\}\{\backslash\text{infty}\}@{\backslash\text{JacobiP}\{\backslash\alpha\}\{\backslash\text{beta}\}\{n\}@{\text{1} - (\text{frac}\{2x\}\{\backslash\text{beta}\})\}} \\ = \backslash\text{LaguerreL}\{\backslash\alpha\}\{n\}@{\text{x}}.$$

This new formula will have identical presentation to the previous formula, and the semantics are more easily extracted (using for instance, the POM tagger). We have also developed semantic macros for sums, products, definite (or indefinite) integrals, as well as for antiderivatives, namely

$$\backslash\text{Sum}\{\#1\}\{\#2\}\{\#3\}@{\#4} := \sum_{\#1=\#2}^{\#3} \#4, \quad \backslash\text{Prod}\{\#1\}\{\#2\}\{\#3\}@{\#4} := \prod_{\#1=\#2}^{\#3} \#4, \\ \backslash\text{Int}\{\#1\}\{\#2\}@{\#3}\{\#4\} := \int_{\#1}^{\#2} \#4 \text{ d}\#3, \quad \backslash\text{Antider}@{\#1}\{\#2\} := \int \#2 \text{ d}\#1.$$

The incorporation of these semantic macros into semantic \LaTeX facilitates the translation of the mathematical \LaTeX source expression to a CAS representation. It therefore improves ones ability to automate injection of the mathematical content into CAS.

There often exists alternate usage of mathematical notations. For example, one often writes for a sum,

$$\sum_{n \in A} f(n),$$

where A is some subset of the integers. It is not difficult to customize the macro definitions to be capable of dealing with such notations, assuming that one has a precise description of A .

3 CAS Verification Procedure for DLMF Formulae

We translate the semantic \LaTeX for our test dataset of DLMF formulae to MAPLE CAS representation, using the tool described in [6]. We use easily configurable settings to control our verifications. Using the configuration files, we can control and customize settings for the verification process. For instance, if we want to perform numerical evaluations on differences or divisions, or even for more complex expressions of the left-hand sides and right-hand sides of DLMF formulae.

3.1 Symbolic Verification of DLMF Formulae

Originally, we used the standalone MAPLE `simplify` function directly, to symbolically simplify translated formulae. See [9–12] for other examples of where MAPLE and other CAS simplification procedures have been used elsewhere in the literature. Symbolic simplification is performed either on the difference or the division of the left-hand sides and the right-hand sides of extracted formulae. Thus the expected outcome should be respectively either a 0 or 1. Note that other outcomes, such as other numerical outcomes, are particularly interesting, since these may be an indication of errors in the formulae.

In MAPLE, symbolic simplifications are made using internally stored relations to other functions. If a simplification is available, then in practice, it often has to be performed over multiple defined relevant relations. Often, this process fails and MAPLE is unable to simplify the said expression. We have adopted some techniques which assist MAPLE in this process. For example, by forcing an expression to be converted into another specific representation, in a pre-processing step, could potentially improve the odds that MAPLE is able to recognize a possible simplification. By trial-and-error, we discovered (and implemented) the following pre-processing steps which significantly improve the simplification process:

- conversion to exponential representation;
- conversion to hypergeometric representation;
- expansion of expressions (for example $(x+y)^2$); and
- combined expansion and conversion processes.

3.2 Constraint Handling

Correct assumptions about variable domains are essential for CAS systems, and not surprisingly lead to significant improvements in the CAS ability to simplify. The DLMF provides constraint (variable domain) metadata for formulae, and as mentioned in Sect. 2.1, we have extracted this formula metadata. We have incorporated these constraints as assumptions for the simplification process. Note however, that a direct translation of the constraint metadata is usually not sufficient for a CAS to be able to understand it. Furthermore, testing invalid values for numerical tests returns incorrect results.

For instance different symbols must be interpreted differently depending on the usage. One must be able to interpret correctly certain notations of this kind. For example, one must be able to interpret the command `a,b\in A`, which indicates that both variables `a` and `b` are elements of the set `A` (or more generally `a_1,\dots,a_n\in A`). Similar conventions are often used for variables being elements of other sets such as the sets of rational, real or complex numbers, or for subsets of those sets.

Also, one must be able to interpret the constraints as variables in sets defined using an equals notation such as `n=0,1,2,\dots`, which indicates that the variable `n` is a integer greater than or equal to zero, or together `n,m=0,1,2,\dots`,

both the variables n and m are elements of this set. Since mathematicians who write \LaTeX are often casual about expressions such as these, one should know that $0, 1, 2, \dots$ is the same as $0, 1, \dots$. Consistently, one must also be able to correctly interpret infinite sets (represented as strings) such as $=1, 2, \dots$, $=1, 2, 3, \dots$, $=-1, 0, 1, 2, \dots$, $=0, 2, 4, \dots$, or even $=3, 7, 11, \dots$, or $=5, 9, 13, \dots$. One must also be able to interpret finite sets such as $=1, 2$, $=1, 2, 3$, or $=1, 2, \dots, N$.

An entire language of translation of mathematical notation must be understood in order for CAS to be able to understand constraints. In mathematics, the syntax of constraints is often very compact and contains textual explanations. Translating constraints from \LaTeX to CAS is a complicated task because CAS only allow precise and strict syntax formats. For example, the typical constraint $0 < x < 1$ is invalid if directly translated to MAPLE, because it would need to be translated to two separate constraints, namely $x > 0$ and $x < 1$.

We have improved the handling and translation of variable constraints/assumptions for simplification and numerical evaluation. Adding assumptions about the constrained variables improves the effectiveness of MAPLE's `simplify` function. Our previous approach for constraint handling for numerical tests was to extract a pre-defined set of test values and to filter invalid values according to the constraints. Because of this strategy, there often was no longer any valid values remaining after the filtering. To overcome this issue, instead, we chose a single numerical value for a variable that appears in a pre-defined constraint. For example, if a test case contains the constraint $0 < x < 1$, we chose $x = \frac{1}{2}$.

A naive approach for this strategy, is to apply regular expressions to identify a match between a constraint and a rule. However, we believed that this approach does not scale well when it comes to more and more pre-defined rules and more complex constraints. Hence, we used the POM tagger to create blueprints of the parse trees for pre-defined rules. For the example \LaTeX constraint $0 < x < 1$, rendered as $0 < x < 1$, our textual rule is given by

$$0 < \text{var} < 1 ==> 1/2.$$

The parse tree for this blueprint constraint contains five tokens, where `var` is an alphanumeric token that is considered to be a placeholder for a variable.

We can also distinguish multiple variables by adding an index to the placeholder. For example, the rule we generated for the mathematical \LaTeX constraint $x, y \in \text{\Real}$, where \Real is the semantic macro which represents the set of real numbers, and rendered as $x, y \in \mathbb{R}$, is given by `var1, var2 \in \Real ==> 3/2, 3/2`.

A constraint will match one of the blueprints if the number, the ordering, and the type of the tokens are equal. Allowed matching tokens for the variable placeholders are Latin or Greek letters and alphanumeric tokens.

3.3 Numerical Verification of DLMF Formulae

Due to the fact that CAS simplification verification is not extremely effective, we also used our CAS translation to perform numerical testing as well. To perform automated numerical evaluations, we extracted all variables from the expression. Variables are extracted by identifying all names [13]⁵ from an expression. This will also extract constants which need to be deleted from the list first. Afterwards, we set each variable to a specific numerical value and numerically evaluate the expression. One is given the capability to choose the numerical values of the given variables as either complex numbers, real numbers, or even as integers. Given this typing of variables, we also check the values to ensure that they are subject to the constraints (see Sect. 3.2 above). We allow for the definition of a set of numerical values for variables that we want to verify, and for multiple variable choices. The power-set of these choices is looped over (we evaluate all combinations of variable-value pairs). We consider an evaluation to be successful if the outcome is below a given threshold (currently set at 0.001) compared with a certain precision (currently set at 10).

4 Summary

We have created a test dataset⁶ of 4,078 semantic L^AT_EX formulae, extracted from the DLMF. We translated each test case to a representation in MAPLE and used MAPLE's `simplify` function on the formula difference to verify that the translated formulae remain valid. Our forward translation tool (Sect. 2) was able to translate 2,405 (approx. 59%) test cases. Most likely, a translation failed because it encountered a DLMF/DRMF semantic macro without a known translation to MAPLE (1,021 of non-translated cases, approx. 61%). An example of such a non-supported macros is the (basic) q -hypergeometric function [1, (17.4.1)]. In other cases, translation of expressions failed because they contained insufficient semantic information, such as when the prime and superscript notations for derivatives are used (Sect. 2.3), or encountered assorted errors in the translation engine. Such assorted errors include unimplemented grammar mappings from the POM tagger such as handling sub-superscript outputs or overlining expressions for complex conjugations.

We applied our symbolic verification techniques to the 2,405 translated expressions. The proposed simplification was able to verify 481 of these expressions (20%). Pre-conversion improved the effectiveness of `simplify` and was used to convert the translated expression to a different form before simplification of the formula difference. We used conversions to exponential and hypergeometric form and expanded the translated expression. Those pre-processed manipulations increased the number of formulae verified from 481 to 660 (approx. 27.4%).

⁵ A *name* in MAPLE is a sequence of one or more characters that uniquely identifies a command, file, variable, or other entity.

⁶ This dataset is available on request to the authors.

The remaining 1,745 test cases were translated but not verified. Note that verification should also fail because either expression would not contain a logical relation and therefore are unverifiable (Sect. 2.2), but these are filtered out in a pre-processing step, or that the CAS is not yet sophisticated enough to verify (such as in the case of asymptotic representations).

We have also applied several numerical tests to these remaining test cases (Sect. 3.3). The results of the numerical tests strongly depend on the tested values. Besides the defined numerical values for general tests, we applied special values for certain constraints. This was realized by our new approach of applying rules for blueprints of common constraints (Sect. 3.2). In automated evaluations, we performed numerical tests by setting variables in test expressions to values contained in the set $\{-0.5, 0.5, 1.5\}$. In the cases where the tested expression is an equation, we tested the difference between the left- and the right-hand sides of the equations. For other relations, we tested if the relation still remains for the calculated numerical values. With this set of the three values and the set of special values depending on additional constraints, 418 of the remaining unverified test cases return a valid output (approx. 24%). Because they exceeded our pre-defined time-limit of 300 seconds, 14 numerical tests stopped. In 892 cases (approx. 51.1%), the calculated values were above our given threshold of 0.001. Note that numerical verifications may fail because of one of four reasons:

1. the numerical engine tests invalid combinations of values;
2. the translation is incorrect;
3. there may be an error in the DLMF source; or
4. there may be an error in MAPLE.

Typical originations for reason 1 are errors produced in the translations of constraints (Sect. 3.2), and may prevent the numerical test engine from using valid numerical values for evaluation. Another typical problem is missing information from context, such as if a variable was pre-defined by substitution, but the test case misses this information. For example, [1, (9.6.1)] defines $\zeta = \frac{2}{3}z^{3/2}$. The following equations are only valid considering the pre-defined numerical value for ζ . Furthermore, this equation is a definition of ζ . If we test this equation by simplifications or numerical evaluations, we get invalid results. We call such failed numerical evaluations by reason 1, *false positives*, since they were marked as particularly interesting cases (for reasons 2–4) but they were caused by missing semantic information originating from the source. It seems that the number of false positive results is very high and plan to reduce this in the future.

Table 2 gives an overview of the symbolic and numerical testing for each DLMF chapter. The overview of translations (T) per chapter reveals the weakness of our translation engine especially in the Chapters 17 (QH) and 34 (TJ). A high ratio of symbolically verified and translated formulae indicates a large success rate for translating macros into MAPLE functions. This is one explanation for the fact that our best result comes from Chapter 4 EF.

Table 2. Overview of symbolic and numerical testing for each DLMF chapter with the corresponding DLMF chapter 2-letter codes (2C), chapters numbers (C#), number of extracted expressions of the second extraction scan (F2), number of translated expressions (T) with approximated percentages, number of translated and symbolically verified cases (TV_s) with approximated percentages, and number of translated cases where the numerical evaluations returned valid outputs for the set of test values $\{-0.5, 0.5, 1.5\}$ (TV_n) with approximated percentages (excluding the number of symbolically verified translations). The best results of each technique are highlighted in bold.

2C	C#	F2	T	TV _s	TV _n	2C	C#	F2	T	TV _s	TV _n
AL	1	85	44 (51.8%)	4 (4.7%)	8 (9.9%)	EL	19	391	151 (38.6%)	31 (7.9%)	31 (8.6%)
AS	2	56	27 (48.2%)	2 (3.6%)	0 (0%)	TH	20	84	56 (66.7%)	6 (7.1%)	0 (0%)
NM	3	41	34 (82.9%)	1 (2.4%)	1 (2.5%)	MT	21	7	2 (28.6%)	0 (0%)	0 (0%)
EF	4	466	406 (87.1%)	182 (39.1%)	61 (21.5%)	JA	22	181	146 (80.7%)	59 (32.6%)	34 (27.9%)
GA	5	56	39 (69.6%)	10 (17.9%)	15 (32.6%)	WE	23	70	28 (40.0%)	0 (0%)	0 (0%)
EX	6	44	13 (29.5%)	0 (0%)	1 (2.3%)	BP	24	60	34 (56.7%)	7 (11.2%)	10 (18.9%)
ER	7	80	60 (75.0%)	24 (30%)	11 (19.6%)	ZE	25	50	27 (54.0%)	12 (24%)	6 (15.8%)
IG	8	120	89 (74.2%)	24 (20%)	13 (13.5%)	CM	26	54	27 (50.0%)	9 (16.7%)	10 (22.2%)
AI	9	160	71 (44.4%)	8 (5%)	19 (12.5%)	NT	27	24	4 (16.7%)	0 (0%)	0 (0%)
BS	10	355	174 (49.0%)	44 (12.4%)	47 (11.9%)	MA	28	186	43 (23.1%)	6 (3.2%)	6 (3.3%)
ST	11	83	71 (85.5%)	20 (24.1%)	11 (17.5%)	LA	29	72	17 (23.6%)	0 (0%)	1 (1.4%)
PC	12	100	71 (71.0%)	14 (14%)	12 (14%)	SW	30	46	11 (23.9%)	0 (0%)	0 (0%)
CH	13	174	168 (96.6%)	61 (35.1%)	24 (21.2%)	HE	31	33	28 (84.8%)	4 (12.1%)	0 (0%)
LE	14	173	163 (94.2%)	31 (17.9%)	37 (26.1%)	PT	32	59	31 (52.5%)	0 (0%)	1 (1.7%)
HY	15	174	170 (97.7%)	39 (22.4%)	34 (25.2%)	CW	33	92	18 (19.6%)	1 (1.1%)	1 (1.1%)
GH	16	32	18 (56.3%)	3 (9.4%)	0 (0%)	TJ	34	30	0 (0%)	—	—
QH	17	96	0 (0%)	—	—	FM	35	1	0 (0%)	—	—
OP	18	253	141 (55.7%)	58 (22.9%)	23 (11.8%)	IC	36	90	23 (25.6%)	0 (0%)	0 (0%)
						Σ		4087	2405 (58.8%)	660 (16.1%)	418 (12.2%)

Originally, the verification rules were developed to identify errors in the translation engine, see reason 2. However, further investigations of the 892 cases reveals a sign error in the DLMF (reason 3), and this corresponded to [1, (14.5.14)], namely

$$Q_{\nu}^{-1/2}(\cos \theta) = \left(\frac{\pi}{2 \sin \theta} \right)^{1/2} \frac{\cos((\nu + \frac{1}{2})\theta)}{\nu + \frac{1}{2}},$$

where Q_{ν}^{μ} is the Ferrers function of the second kind. This error can be found on [14, p. 359], and was reported in DLMF Version 1.0.16 on September 18, 2017. Other cases where formulae seem to be unverified by numerical tests are for DLMF formulas which are valid on Riemann surfaces, but fail because MAPLE uses specific choices of branch values (reason 4). Using our constraint blueprints we were able to identify when a constraint does not match the set of rules. In fact, our constraint handling identified a missing comma after the -2 in the constraint of [1, (10.16.7)], originally printed as $2\nu = -1, -2 - 3, \dots$. This error can be found on [14, p. 228], and was repaired in the DLMF Version 1.0.19 of 2018-06-22.

We are planning to extend the blueprint approach to solve the more general problem of translating constraints to CAS. We continue to update our checking procedure in search of more DLMF and MAPLE errors/inconsistencies, and are constantly improving the translation engine.

Acknowledgements. We are indebted to Bruce Miller and Abdou Youssef for valuable discussions and for the development of the custom macro set of semantic mathematical \LaTeX macros used in the DLMF, and for the development of the POM tagger, respectively. We also thank the DLMF editors for their assistance and support. We also greatly appreciate valuable discussions with Jürgen Gerhard concerning MAPLE.


References

1. Olver, F.W.J., Olde Daalhuis, A.B., Lozier, D.W., Schneider, B.I., Boisvert, R.F., Clark, C.W., Miller, B.R., Saunders, B.V. (eds.) NIST Digital Library of Mathematical Functions. <http://dlmf.nist.gov/>. Release 1.0.19 of 2018-06-22
2. Miller, B.R.: \LaTeX ML: A \LaTeX to XML/HTML/MathML Converter, June 2018. <http://dlmf.nist.gov/LaTeXML/>
3. Miller, B.R., Youssef, A.: Technical aspects of the Digital Library of Mathematical Functions. *Ann. Math. Artif. Intell.* **38**(1–3), 121–136 (2003)
4. Schubotz, M., Greiner-Petter, A., Scharpf, P., Meuschke, N., Cohl, H.S., Gipp, B.: Improving the representation and conversion of mathematical formulae by considering their textual context. In: Joint Conference on Digital Libraries 2018, June 2018, Fort Worth, TX (2018)
5. Youssef, A.: Part-of-math tagging and applications. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) C1CM 2017. LNCS (LNAI), vol. 10383, pp. 356–374. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_25
6. Cohl, H.S., et al.: Semantic preserving bijective mappings of mathematical formulae between document preparation systems and computer algebra systems. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) C1CM 2017. LNCS (LNAI), vol. 10383, pp. 115–131. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_9
7. Schubotz, M., Meuschke, N., Hepp, T., Cohl, H.S., Gipp, B.: VMEXT: a visualization tool for mathematical expression trees. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) C1CM 2017. LNCS (LNAI), vol. 10383, pp. 340–355. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_24
8. Cohl, H.S., et al.: Growing the digital repository of mathematical formulae with generic latex sources. In: Kerber, M., Carette, J., Kaliszky, C., Rabe, F., Sorge, V. (eds.) C1CM 2015. LNCS (LNAI), vol. 9150, pp. 280–287. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20615-8_18
9. Koepf, W.: On two conjectures of M. S. Robertson. *Complex Variables. Theory Appl. Int. J.* **16**(2–3), 127–130 (1991)
10. Ballarin, C., Homann, K., Calmet, J.: Theorems and algorithms: an interface between Isabelle and Maple. In: Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation, ISSAC 1995, pp. 150–157. ACM, New York (1995)
11. Harrison, J., Théry, L.: A skeptic’s approach to combining HOL and Maple. *J. Autom. Reason.* **21**(3), 279–294 (1998)

12. Bailey, D.H., Borwein, J.M., Kaiser, A.D.: Automated simplification of large symbolic expressions. *J. Symbolic Comput.* **60**, 120–136 (2014)
13. Bernardin, L., Chin, P., DeMarco, P., Geddes, K.O., Hare, D.E.G., Heal, K.M., Labahn, G., May, J.P., McCarron, J., Monagan, M.B., Ohashi, D., Vorkoetter, S.M.: *Maple Programming Guide*. Maplesoft, a Division of Waterloo Maple Inc., Waterloo (2016)
14. Olver, F.W.J., Lozier, D.W., Boisvert, R.F., Clark, C.W. (eds.) *NIST Handbook of Mathematical Functions*. Cambridge University Press, Cambridge (2010). U.S. Department of Commerce, National Institute of Standards and Technology, Washington, DC. With 1 CD-ROM (Windows, Macintosh and UNIX)



Concrete Semantics with Coq and CoqHammer

Lukasz Czajka¹(✉), Burak Ekici²(✉), and Cezary Kaliszyk²(✉) 

¹ University of Copenhagen, Copenhagen, Denmark
luta@di.ku.dk

² University of Innsbruck, Innsbruck, Austria
{burak.ekici,cezary.kaliszyk}@uibk.ac.at

Abstract. The “Concrete Semantics” book gives an introduction to imperative programming languages accompanied by an Isabelle/HOL formalization. In this paper we discuss a re-formalization of the book using the Coq proof assistant (version 8.7.2). In order to achieve a similar brevity of the formal text we extensively use CoqHammer, as well as Coq Ltac-level automation. We compare the formalization efficiency, compactness, and the readability of the proof scripts originating from a Coq re-formalization of two chapters from the book.

1 Introduction

Formal proofs allow today most precise descriptions and specifications of computer systems and programs. Such precision is very important both for human learning and for machine knowledge management. Formalization accompanied courses allow students to investigate the topic to an arbitrary level of detail, and naturally offer very precise exercises of the topic [7]. Formalization attached to mathematical knowledge allows algorithms to the knowledge semantically and permits learning machine translation to, from, and between datasets [5]. This becomes even more important with multi-translation, where the availability of the same text in multiple languages improves the computer-understanding and ability to translate between each two [4].

In this short paper we translate parts of the Concrete Semantics book by Nipkow and Klein to Coq. To do so, we improve the CoqHammer¹ [9] automation to be able to handle the more advanced use-cases, improve the legibility of the reconstructed proofs and compare the proof style and other differences in between the two. The project is in some ways similar to the “Certified Programming with Dependent Types” book [2], however we attempt to avoid dependent types and more advanced constructions to build both an easier material for students and a more precise dataset for bootstrapping an automated translation between proof corpora in the style of [5].

¹ Release: <https://github.com/lukaszcz/coqhammer/releases/tag/v1.0.8-coq8.7>

2 Concrete Semantics with Isabelle/HOL

The Concrete Semantics book [7] by Nipkow and Klein is made of two parts. The first part introduces how to write functional programs, inductive definitions and how to reason about their properties in Isabelle/HOL’s structured proof language. While the second part is devoted to formal semantics of programming languages using the “small” imperative IMP² language as the instance. This part more concretely examines several topics in a wide range varying from operational semantics, compiler correctness to Hoare Logic. The proofs presented in this part are not given in Isabelle/HOL’s structured language. However, such a formalization accompanies the paper proofs via the provided links usually given in section beginnings.

In this work we attempt to reformatize in Coq some subset of the Isabelle/HOL theories that accompanies the second part of the book. As illustrated in Sect. 4, we aim at catching the same level of automation in Coq thus approximating the proof texts to the original ones in terms of length. To do so, we use automated reasoning techniques discussed in Sect. 3.

3 Coq and Coq Automation

The Coq proof assistant is based on the Calculus of Inductive Constructions. The main difference from proof assistants based on higher-order logic is the presence of dependent types. Coq also features a rich tactic language Ltac, which allows to write specialised proof automation tactics. Some standard automation tactics already available in Coq are:

- **intuition**: implements a decision procedure for intuitionistic propositional calculus based on the contraction-free sequent calculi LJ^T* of Roy Dyckhoff.
- **firstorder**: extends **intuition** to a proof search tactic for first-order intuitionistic logic.
- **auto** and **eauto**: implement a Prolog-like backward proof search procedure.

The CoqHammer [3,9] plugin extends Coq automation by a number of other useful and generally more powerful tactics similarly to that available in Isabelle [1]. Its main tactic **hammer** combines machine learning and automated reasoning techniques to discharge goals automatically. It works in three phases:

1. **Premise selection** uses machine learning techniques to choose a subset of the accessible lemmas that are likely useful for the goal.
2. **Translation** of the goal and the preselected lemmas to the input formats of first-order automated theorem provers (ATPs) such as Vampire [6] or Eprover [8], and running the ATPs on the translations.

² IMP is a standard Turing complete imperative language involving the mutable global state as a computational side effect. The reason why this language has been selected is just that it has enough expressive power to be Turing complete.

3. **Reconstruction** uses the information obtained from a successful ATP run to re-prove the goal in the logic of Coq. Upon success the **hammer** tactic should be replaced with the reconstruction tactic displayed in the response window. The success of the reconstruction tactic does not depend on any time limits nor external ATPs, therefore it is machine-independent.

The CoqHammer tool provides various reconstruction tactics. Among others, the tactics **hobvious** and **hsimple** perform proof search via the **yelles** tactic (see the last item below) using the information returned from the successful ATP runs after a constant unfolding and hypothesis simplification. Also, CoqHammer comes with tactics written entirely in Ltac. These tactics do not depend on any external tool, and are not informed about available lemmas in the context:

- **sauto** – a “super” version of the standard Coq tactics **auto** and **intuition**. It tries to simplify the goal and possibly solve it without performing much of actual proof search beyond what **intuition** already does. It is designed in such a way as to terminate in a short time in most circumstances. One can customize it by adding rewrite hints to the **yhints** database.
- **scrush** – essentially a combination of **sauto** and **ycrush**. The **ycrush** tactic tries various heuristics and performs some limited proof search. Usually stronger than **sauto**, but may take a long time if it cannot find a proof. In contrast to **sauto**, **ycrush** does not perform rewriting using the hints in the **yhints** database. One commonly uses **ycrush** after **sauto**.
- **yelles n** – performs proof search up to depth n ; slow for n larger than 3–4.

4 Case Studies

In this section, we illustrate a set of goals that are discharged using the Coq automation techniques, presented in Sect. 3, together with a comparison to their original versions, in an Isabelle/HOL formalization, as presented in the Concrete Semantics book. Notice that the examples in this section are given broadly, with no background details. The point to emphasize here is that we can actually achieve a similar brevity of the formal text in terms of proof lengths using proof automation in Coq. The examples are given in code snippets that have Coq text on the left and Isabelle/HOL text on the right side of the minipages.

Note also that we translated the lemma statements into Coq directly from Isabelle/HOL theory files, and proved them using mostly the standard tactics coming with CoqHammer, with only minimal use of more sophisticated custom Ltac tactics, and practically no hints from Coq hint databases. Therefore the translation is not quite automatic but fairly straightforward.

The example given in the below code snippet comes from the Hoare Logic. Leaving the technical details aside, it basically says that a precondition $\{P\}$ of some Hoare triple can be strengthened into $\{P'\}$ if $\{P'\}$ entails $\{P\}$. This is actually one of the corollaries of the consequence (called **conseq** in our Coq formalization) rule of Hoare Logic. Notice that, in this snippet, **hoaret** is the

Coq inductive predicate representing Hoare triples which corresponds to the notation “ \vdash_t ” on Isabelle/HOL side.

```

Lemma strengthen_pre:
   $\forall$  (P P' Q: assn) c, (entails P' P)
   $\rightarrow$  hoaret P c Q  $\rightarrow$  hoaret P' c Q.
Proof. hobvious Empty (@conseq) (@entails) Qed.

lemma strengthen-pre:
   $\llbracket \forall s. P' s \rightarrow P s; \vdash_t \{P\} c \{Q\} \rrbracket$ 
 $\implies \vdash_t \{P'\} c \{Q\}$ 
  by (metis conseq)

```

Upon a call, the CoqHammer tool gets a proof returned by one of the employed ATPs, and discharges the goal using its reconstruction tactic *hobvious* parametrized with the empty set of hypotheses from the goal context, the rule *conseq* and the definition *entails*. Indeed, this is very similar to what happens in Isabelle/HOL proof of the same fact. The proof is simply made of a call to the *metis* tactic with the *conseq* rule as the argument.

Another but slightly more complicated example that stems from the Hoare Logic (using the same notation as the previous one) is given in the below code snippet. This lemma is a version of the partial correctness of the *while* rule enriched with a measure function *f* which is supposed to decrease in each loop iteration so as to guarantee the loop termination.

```

Lemma While_fun:  $\forall$  b P Q c
  (f: state  $\rightarrow$  nat), ( $\forall$  n: nat, hoaret
  (fun s  $\Rightarrow$  P s  $\wedge$  bval s b = true  $\wedge$ 
  n = f s) c (fun s  $\Rightarrow$  P s  $\wedge$  f s < n))
   $\rightarrow$  hoaret P (While b c)
  (fun s  $\Rightarrow$  P s  $\wedge$  bval s b = false).
Proof. pose While; pose conseq;
  unfold entails in *; yelles 3. Qed.

lemma While-fun:
   $\llbracket \bigwedge n::\text{nat}. \vdash_t \{\lambda s. P s \wedge \text{bval } b s \wedge n = f s\} c \{\lambda s. P s \wedge f s < n\} \rrbracket$ 
 $\implies \vdash_t \{P\} \text{WHILE } b \text{ DO } c \{\lambda s. P s \wedge \neg \text{bval } b s\}$ 
  by (rule Hoare-Total.While [where T= $\lambda s n. n = f s$ , simplified])

```

The Coq proof is found by the Ltac implemented tactic *yelles* which performs a proof search until a user specified depth has been reached. In our concrete example, we give it some guidance by using the primitive Coq tactic *pose* with *while* and *conseq* rules as arguments, adding them to the context (or simply generalizing them), together with unfolding the definition of *entails*. This way, the tactic finds a proof at the proof search depth 3. Isabelle/HOL proof of the same statement follows similar lines. It uses a simplification of the *while* rule with the measure function being $\lambda s n. n = f s$. Just notice that our Coq tactic *yelles* is clever enough on this goal to find the measure function automatically.

A third example is about semantics of the IMP language. The lemma shown in the below snippet states that one can deduce the big-step semantics of any terminating IMP program from its small-step semantics. Observe that, in this snippet, the Coq notations “ \implies ” and “ $\rightarrow *$ ” respectively represent the inductive predicates for the (transitive closure of) IMP big-step and small-step semantics. The single difference on Isabelle/HOL side is that we have “ \implies ” standing for big-step semantics.

```

Lemma lem_small_to_big:  $\forall$  p s,
p  $\longrightarrow^*$  (Skip, s)  $\rightarrow$  p  $\Longrightarrow$  s.
Proof. enough ( $\forall$  p p', p  $\longrightarrow^*$  p'  $\rightarrow$ 
 $\forall$  s, p' = (Skip, s)  $\rightarrow$  p  $\Longrightarrow$  s) by scrush.
intros p p' H. induction H; sauto.
hsimple AllHyps (@lem_small1_big_continue)
Empty. Qed.

```

```

lemma small-to-big:
cs  $\rightarrow^*$  (SKIP,t)  $\Longrightarrow$  cs  $\Rightarrow$  t
apply (induction cs (SKIP,t) rule: star.induct)
apply (auto intro: small1-big-continue)
done

```

The Coq proof of this lemma proceeds by an induction on the (transitive closure of) small-step semantics after introducing a helper statement (asserted by the pure Coq tactic `enough` and proven by the Ltac tactic `scrush`) into the goal context. Then, it calls the Ltac tactic `sauto` to do some preprocessing for the CoqHammer call. The base case $p = (\text{Skip}, s)$ is trivially solved by `sauto`. For the inductive case, namely $\forall s, p' = (\text{Skip}, s) \rightarrow p' \Longrightarrow s$, we call CoqHammer and get the goal solved by an application of the reconstruction tactic `hsimple` which uses all hypotheses in the goal context (that's why we introduce a new one at the beginning) and the helper lemma called `lem_small1_big_continue` with no definitions unfolded. This is again very similar to the Isabelle/HOL proof of the fact in hand. The proof uses the induction principle on the transitive closure of the small-step IMP semantics and then applies the helper lemma `lem_small1_big_continue`. Below we give three more examples that we think interesting in the sense that all cases appear on Coq side are discharged fully automatically. And the text size is fairly close to the one of Isabelle/HOL.

```

Lemma exec_n_exec:  $\forall$  n P c1 c2,
exec_n P c1 n c2  $\rightarrow$  exec P c1 c2.
Proof. induction n; intros; destruct H.
- scrush.
- pose @star_step;
hobvious (@H, @IHn)(@Star.star_step)
(@Compiler.exec). Qed.

```

```

Lemma exec_exec_n:  $\forall$  P c1 c2,
exec P c1 c2  $\rightarrow \exists$  n, exec_n P c1 n c2.
Proof. intros; induction H.
-  $\exists$  0; scrush.
- pose exec_Suc; scrush. Qed.

```

```

Lemma exec_eq_exec_n:  $\forall$  P c1 c2,
exec P c1 c2  $\leftrightarrow \exists$  n, exec_n P c1 n c2.
Proof. pose exec_exec_n;
pose exec_n_exec; scrush. Qed.

```

```

lemma exec-n-exec:
P  $\vdash$  c  $\rightarrow^{\wedge} n$  c  $\Longrightarrow$  P  $\vdash$  c  $\rightarrow^*$  c
by (induct n arbitrary: c) (auto intro: star.step)

```

```

lemma exec-exec-n:
P  $\vdash$  c  $\rightarrow^*$  c  $\Longrightarrow \exists$  n. P  $\vdash$  c  $\rightarrow^{\wedge} n$  c
by (induct rule: star.induct) (auto intro:
exec-Suc)

```

```

lemma exec-eq-exec-n:
(P  $\vdash$  c  $\rightarrow^*$  c) = ( $\exists$  n. P  $\vdash$  c  $\rightarrow^{\wedge} n$  c)
by (blast intro: exec-exec-n exec-n-exec)

```

The main lemma `exec_eq_exec_n`, using the other two above, is broadly about the symbolic compilation of IMP programs into a low level language based on a stack machine. It specifically says that one can speak about the n step instruction executions instead of reflexive transitive closure of single step executions. Note that the Coq predicates `exec_n` and `exec` are respectively standing for n step instruction, and transitive closure of single step instruction. These are denoted as “ $\vdash \rightarrow^*$ ” and “ $\vdash \rightarrow^{\wedge} n$ ” in Isabelle/HOL text respectively. The Coq proof from left to right (`exec_n_exec`) of the equivalence is based on an induction over n and the other direction (`exec_exec_n`) is based on an induction over transitive closure of single step executions. The base cases of both induction steps are trivially solved by the tactic `scrush`. The inductive case of the former

proof is a `CoqHammer` call which discharges the goal using the reconstruction tactic `hobvious`. It uses two hypotheses (`H` and the induction hypothesis `IHn`) from the goal context with the lemma called `star_step` (coming from `Star.v`), and the definition `exec`. The inductive case of the latter is just made of an `scrush` application with a guide reminding that the goal is a variant of the lemma `exec_Suc`.

Again, these proofs follow very similar lines with those of Isabelle/HOL of the same facts. The proof of `exec_n_exec` induces on `n` and uses the definition `star_step` to discharge the goal. Similarly, `exec_exec_n` is proven by an induction on the transitive closure of single step executions followed by the application of the `exec_Suc` fact.

5 Conclusion

We have re proven 101 lemmas from the Isabelle/HOL theories `Star`, `AExp`, `BExp`, `ASM`, `Com`, `Big_Step`, `Hoare`, `Small_Step`, `Compiler` and `Compiler2` in Coq; heavily using the automation techniques described in previous sections.

	# of lines	# of words	# of tactics	# of hammer calls	Time(secs)
Isabelle/HOL	2806	11278	544	Not verifiable	31
Coq	3493	19292	1190	468	149

As shown in the above table, the number of Coq tactics we used to get the same lemmas proven is almost twice in number, as opposed to Isabelle/HOL, but about half of which benefits from the automation techniques that `CoqHammer` comes with. This can be seen as an improvement given that the Isabelle/HOL tactics are more compound than the “simple” Coq tactics.

The `coqc 8.7.2` needs 149s to compile the translated source and Isabelle 2017 needs 31s to build the corresponding theories on an Intel Core i7-7600U machine. We attribute the difference mostly to the fact that all used Isabelle tactics are written in ML, while most Coq ones use Ltac.

We plan to build on this work by proving more lemmas coming from different theories of the book and by improving the level of automation, thus decreasing the number of words, in the already proven goals. Please see:

<https://github.com/lukaszcz/COQ-IMP>

for the proofs done so far.

Acknowledgments. This work has been supported by the Austrian Science Fund (FWF) grant P26201, the European Research Council (ERC) grant no. 714034 *SMART* and the Marie Skłodowska-Curie action *InfTy*, program H2020-MSCA-IF-2015, number 704111.

References

1. Blanchette, J.C., Greenaway, D., Kaliszyk, C., Kühlwein, D., Urban, J.: A learning-based fact selector for Isabelle/HOL. *J. Autom. Reason.* **57**(3), 219–244 (2016)
2. Chlipala, A.: *Certified Programming with Dependent Types - A Pragmatic Introduction to the Coq Proof Assistant*. MIT Press, Cambridge (2013)
3. Czajka, L., Kaliszyk, C.: Goal translation for a hammer for Coq (extended abstract). In: Blanchette, J., Kaliszyk, C. (eds.) *International Workshop on Hammers for Type Theories (HaTT 2016)*. EPTCS, vol. 210, pp. 13–20 (2016)
4. Dong, D., Wu, H., He, W., Yu, D., Wang, H.: Multi-task learning for multiple language translation. In: *ACL*, no. 1, pp. 1723–1732. The Association for Computer Linguistics (2015)
5. Kaliszyk, C., Urban, J., Vyskočil, J., Geuvers, H.: Developing corpus-based translation methods between informal and formal mathematics: project description. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) *CICM 2014*. LNCS (LNAI), vol. 8543, pp. 435–439. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08434-3_34
6. Kovács, L., Voronkov, A.: First-order theorem proving and VAMPIRE. In: Sharygina, N., Veith, H. (eds.) *CAV 2013*. LNCS, vol. 8044, pp. 1–35. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_1
7. Nipkow, T., Klein, G.: *Concrete Semantics - With Isabelle/HOL*. Springer, Cham (2014). <https://doi.org/10.1007/978-3-319-10542-0>
8. Schulz, S.: System description: E 1.8. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) *LPAR 2013*. LNCS, vol. 8312, pp. 735–743. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45221-5_49
9. Czajka, L., Kaliszyk, C.: Hammer for Coq: automation for dependent type theory. *J. Autom. Reason.* **61**(1–4), 423–453 (2018)



Automated Determination of Isoptics with Dynamic Geometry

Thierry Dana-Picard¹  and Zoltán Kovács² 

¹ Jerusalem College of Technology, Havaad Haleumi 21 St., 9116011 Jerusalem, Israel
ndp@jct.ac.il

² The Private University College of Education of the Diocese of Linz,
Salesianumweg 3, 4020 Linz, Austria
zoltan@geogebra.org

Abstract. We present two approaches to symbolically obtain isoptic curves in the dynamic geometry software GeoGebra in an automated, interactive process. Both methods are based on computing implicit locus equations, by using algebraization of the geometric setup and elimination of the intermediate variables. These methods can be considered as *automatic discovery*.

Our first approach uses *pure computer algebra support of GeoGebra*, utilizing symbolic differentiation of the input formula. Due to computational challenges we limit here our observations to quartic curves. The second approach hides all details in computer algebra from the user, that is, the input problem can be defined by a *purely geometric way*, considering a conic, a circle being given by its center and radius, and a parabola by the pair focus-directrix, for instance. The results are, however, not new, the novelty being is the way we obtain them, as a handy method for *a new kind of man and machine communication*. Both approaches deliver an algebraic output, namely, a polynomial and its graphical representation. The output is dynamically changed when using a slider bar. In this sense, *dynamic study* of isoptics can be introduced in a new way.

The internal GeoGebra computations, partly *programmed by the authors*, is an on-going work with various challenges in properly formulating systems of equations, in particular, to optimize computations and to avoid unnecessary extra curves in the output. Our paper highlights some of these challenges as well.

1 Introduction

In this paper we study isoptic curves of conics and quartics, using the abilities of GeoGebra, a free dynamic geometry software program (see <http://geogebra.org>). In particular, we demonstrate its capabilities on obtaining isoptic curves with two substantially different approaches. Both methods require advanced computer algebra algorithms, namely, effective computation of an elimination ideal. This feature is implemented by the embedded Computer Algebra System (CAS) Giac [13].

The first method is a standard way to handle geometry problems by using computer algebra. However, we define the problem by using implicit equations. The second method harnesses some novel developments in the GeoGebra tool, namely its implicit locus computation feature, discussed in detail in [1, 2, 10, 12, 16].

The first applied method yields new results in obtaining the isoptic curve of some quartic curves. The second method does not explicitly yield new results, but it can help modeling the basic ideas of isoptic curves of conics by using dynamic geometry in a very simple way, therefore it can introduce the topic to a wider audience than previous works on isoptic curves, including high school students.

In our contribution, both methods exploit *automatic discovery*, a formerly unknown approach in studying isoptics. The implementation of the second method is based on internal improvements of the GeoGebra tool. The programming work, including optimization of geometric equations, was performed by the authors.

We used the following software packages to compute various results:

- GeoGebra with a numerical and graphic approach is used in Subsect. 2.1.
- Maple 2017 was used in Subsubsect. 2.2 for plotting and in Sect. 3 for some algebraic computations.
- Symbolic computations provided by GeoGebra were used in the rest of the paper.

While Maple has no interface with GeoGebra for communication, we imported some formulas from GeoGebra using “copy & paste” to perform more advanced algebraic computations.

Section 2 gives a general overview on isoptic curves in the literature. Sections 3 and 4 demonstrate our approaches to compute isoptic curves with the two methods. Finally, Sect. 5 summarizes our work.

2 Isoptic Curves

Let \mathcal{C} be a plane curve. For a given angle θ such that $0 \leq \theta \leq 180^\circ$, a θ -*isoptic curve* (or simply a θ -isoptic) of \mathcal{C} is the geometric locus of points M through which passes a pair of tangents with an angle of θ between them. If $\theta = 90^\circ$, i.e. if the tangents are perpendicular, then the isoptic curve is called an *orthoptic curve*. Isoptic curves may either exist or not, depending on the given curve and on the angle.

2.1 Orthoptics of Conics

Orthoptic curves of conics are well known since the ancient Greeks. They are displayed in Fig. 1.

- (i) The orthoptic curve of a parabola is its *directrix*¹. If the parabola has equation $y^2 = 2px$ (for p a non-zero real), then its directrix has equation $x = p/2$.
- (ii) The orthoptic curve of an ellipse is its *director circle*². If the ellipse is given by the canonical equation $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$, then the director circle has the equation $x^2 + y^2 = a^2 + b^2$.
- (iii) The existence of an orthoptic curve for a hyperbola depends on the eccentricity, i.e. on the ratio c/a , where $c^2 = a^2 - b^2$, where a and b define the respectively focal axis and the non-focal axis of the hyperbola. If it exists, the orthoptic curve of the hyperbola with canonical equation $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$ (i.e. the focal axis is the x -axis) is the circle whose equation is $x^2 + y^2 = a^2 - b^2$, also called the *director circle*³.

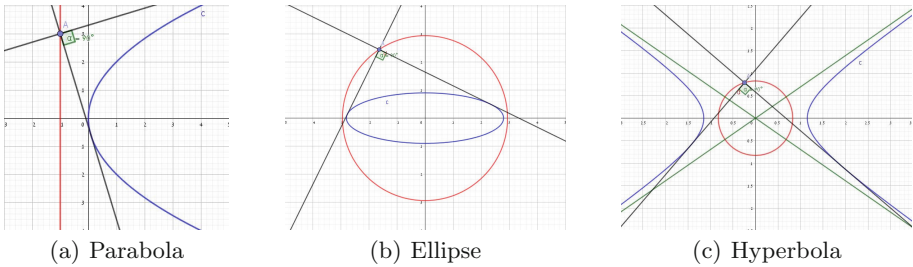


Fig. 1. Orthoptics of conics

This can be verified in a numerical way by utilizing a Dynamic Geometry System (DGS), without computing equations symbolically. Figure 2 is a screen snapshot of an applet using GeoGebra for checking perpendicularity.

Isoptics of ellipses have been studied in [6], yielding a connection between conic sections and toric sections. Isoptics of closed Fermat curves have been studied in [8]. Miernowski, Mozgawa and Szalkowski have studied isoptics for other curves, such as rosettes which are open curves; see [14, 15]. They use support functions to represent the curves.

The study of isoptics of conic sections has been performed using pure algebraic methods. As the equation of a conic is a quadratic equation, the existence of tangents to the conic is also expressed by a quadratic condition. It follows that the existence of tangents is easy to prove, with an analysis of the sign of a discriminant, and explicit equations can be derived for isoptics. Isoptics of parabolas are branches of hyperbolas and isoptics of ellipses and of hyperbolas (when they exist) are quartic curves called Spirics of Perseus. In all the cases, solution to polynomial systems of equations had to be computed. For this purpose, Gröbner packages were very useful.

¹ See the GeoGebra applet at <https://www.geogebra.org/m/pwrWy9dG>.

² See the GeoGebra applet at <https://www.geogebra.org/m/SkQ5qxYr>.

³ See the GeoGebra applet at <https://www.geogebra.org/m/tZcGGrCm>.

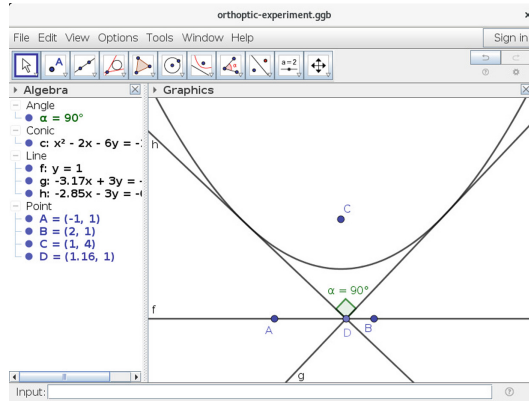
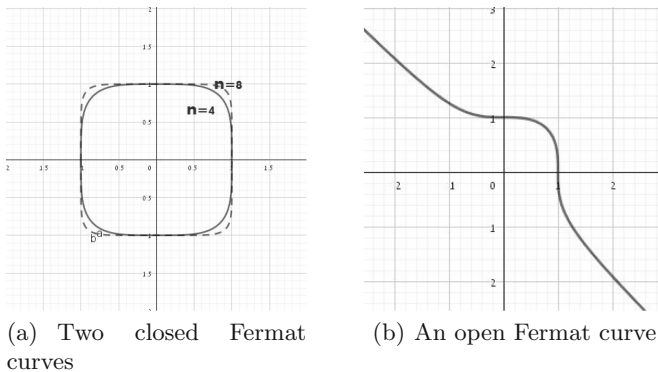


Fig. 2. Checking perpendicularity in GeoGebra

2.2 Generalization to Other Curves

The above-mentioned algebraic methods used for conics cannot generally be applied neither to algebraic curves of higher degree nor to non-algebraic curves. We illustrate this with two examples: Fermat curves as in [8], and an astroid as in [9].

Fermat Curves. We call *Fermat curve* a curve \mathcal{F}_n in the affine plane whose equation is of the form $x^n + y^n = 1$, where n is a non-negative integer. If $n = 2$, the curve is a circle and, by elementary plane geometry, all its isoptics are concentric circles. For even n , the curve \mathcal{F}_n is closed, and for odd n , \mathcal{F}_n is open. Examples are displayed in Fig. 3.



(a) Two closed Fermat curves

(b) An open Fermat curve

Fig. 3. Fermat curves

For even exponents, Fermat curves are smooth and strictly convex loops. Therefore, through every external point passes a pair of tangents. For odd exponents, Fermat curves are open smooth curves, and the number of tangents through a point in the plane can vary. Figure 4 shows three different situations: 0, 2 and 4 tangents.

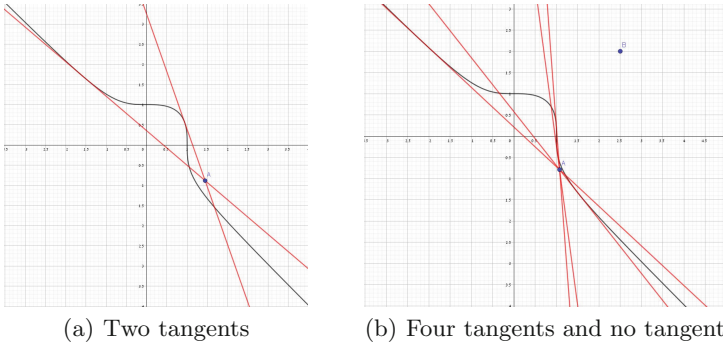


Fig. 4. Multiplicity of tangents to the curve whose equation is $x^3 + y^3 = 1$

Because of Fermat’s Last Theorem, every point on \mathcal{F}_3 but the point of intersection with the axes has at least one irrational coordinate. It follows that writing the same kind of systems of equations as for conics and trying to solve with a Gröbner bases package, looking for Elimination Ideals is useless. These packages do not work over the field \mathbb{R} of real numbers, they work either over the field of rational numbers or over finite fields. Both cases are irrelevant here. In [8], two methods are generally used:

1. experimental work with GeoGebra, in particular the Locus command,
2. numerical methods for θ -isoptics when $\theta \neq 90^\circ$.

This allows one to find a graphical representation of the various θ -isoptics. The orthoptic case is the easiest to deal with. Recall that the curve \mathcal{F}_n is invariant by a rotation of 90° . Now, take a parametric presentation of a tangent L_1 to \mathcal{F}_n , then substitute $t + \pi/2$ instead of t . This gives a parametric presentation of a tangent L_2 , perpendicular to L_1 . The geometric locus of the points of intersection of L_1 and L_2 is the desired orthoptic of \mathcal{F}_n . It has been obtained graphically using the **Locus** command, and a parametric representation has also been computed. Figure 5 shows the Fermat curve \mathcal{F}_{16} and its orthoptic, plotted with Maple 2017, using the parametric presentation.

Astroid. Isoptics of an astroid are studied in [9]. This may be viewed as a generalization of Fermat curves for a non integer power, but the methods have to be different, in particular because of the following remarks:

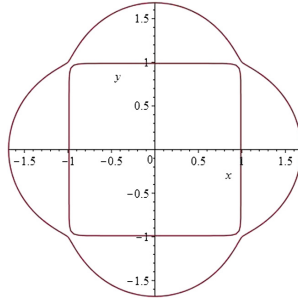


Fig. 5. The orthoptic of a closed Fermat curve

- (i) An astroid is not a Jordan curve: it is non-convex and it has four cusps.
- (ii) There exist points through which more than two tangents pass. Experimentation with GeoGebra shows that these are interior points. It seems that they are all the interior points, but the experimentation is not a proof.
- (iii) For some values of the angle θ , the θ -isoptic is circumscribed by the astroid, and for other values the θ -isoptic has both internal and external points.

The study combined geometric experimentation with GeoGebra and algebraic computations with a Computer Algebra System (CAS). Two examples are shown in Fig. 6.

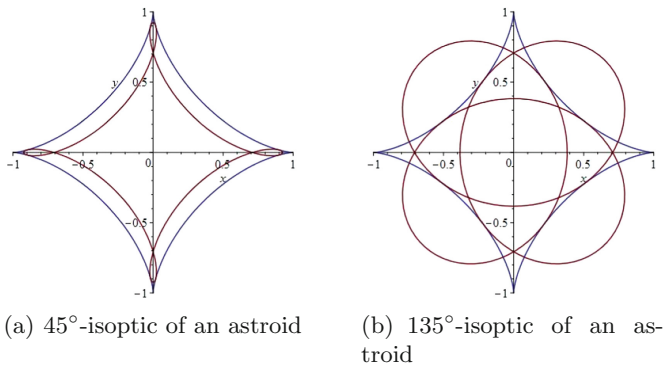


Fig. 6. Isoptics of an astroid

3 The First Approach

Our first approach is as follows. Let \mathcal{C} be an algebraic curve given by an implicit equation $F(x, y) = 0$. Compute the derivatives $d_x = F'_x$ and $d_y = F'_y$. Consider

points $A(x_A, y_A)$ and $B(x_B, y_B)$ that are assumed to be points of the curve, that is,

$$F(x_A, y_A) = 0 \tag{1}$$

and

$$F(x_B, y_B) = 0 \tag{2}$$

hold.

Compute the partial derivatives $p_{x,A} = F'_x(x_A, y_A)$, $p_{x,B} = F'_x(x_B, y_B)$, $p_{y,A} = F'_y(x_A, y_A)$ and $p_{y,B} = F'_y(x_B, y_B)$. Now, when speaking about orthoptic curves, we can assume that

$$p_{x,A} \cdot p_{x,B} + p_{y,A} \cdot p_{y,B} = 0, \tag{3}$$

otherwise, when speaking about θ -isoptics, the following equation holds:

$$(p_{x,A} \cdot p_{x,B} + p_{y,A} \cdot p_{y,B})^2 = \cos^2 \theta \cdot (p_{x,A}^2 + p_{y,A}^2) \cdot (p_{x,B}^2 + p_{y,B}^2). \tag{4}$$

When defining a point $P(x, y)$ that is an element of both tangents t_1 and t_2 to c , it is clear that the points A , $A' = (x_A + p_{y,A}, y_A - p_{x,A})$ and P are collinear; on the other hand, B , $B' = (x_B + p_{y,B}, y_B - p_{x,B})$ and P are collinear as well. Therefore the following equations hold:

$$\begin{vmatrix} x_A & y_A & 1 \\ x_A + p_{y,A} & y_A - p_{x,A} & 1 \\ x & y & 1 \end{vmatrix} = 0, \tag{5} \quad \begin{vmatrix} x_B & y_B & 1 \\ x_B + p_{y,B} & y_B - p_{x,B} & 1 \\ x & y & 1 \end{vmatrix} = 0. \tag{6}$$

At this point we have 5 equations. By eliminating all variables but x and y we are about to obtain an implicit equation whose graphical representation is, at least partly, the θ -isoptic curve. This technique is discussed in detail in [4, Chap. 3, §2] in the frames of elimination theory, and, from practical approach also in [3, Chap. IV, Example 5.8], and from the point of view of the GeoGebra implementation in [1] (where it is identified as automatic discovery). Theoretically, the obtained implicit equation is a multiple of the algebraic closure of the geometrically expected set. In other words, some factors of the obtained implicit equation will contain the expected curve.

Example 1. (See <https://www.geogebra.org/m/JvhNwAzF> in Fig. 7 for an online applet.) Let $F(x, y) = x^4 - y$, that is, we consider the graph of the function $y = x^4$. Let us compute the orthoptic of \mathcal{C} , that is, $\theta = 90^\circ$. Then the following equations hold:

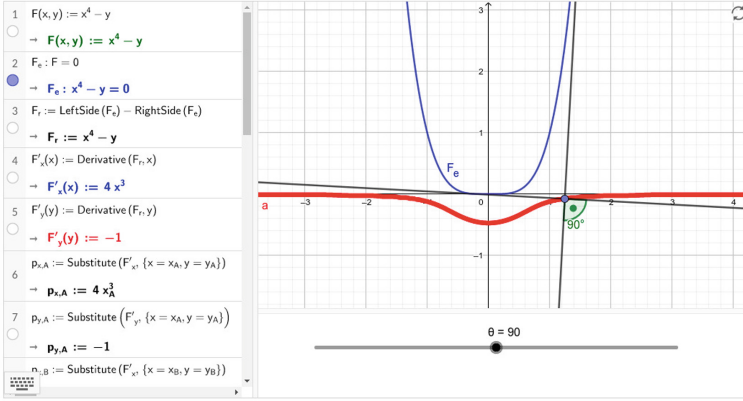


Fig. 7. The orthoptic of the graph given by the quartic equation $y = x^4$

$$x_A^4 - y_A = 0, \quad (7)$$

$$x_B^4 - y_B = 0, \quad (8)$$

$$4x_A^3 \cdot 4x_B^3 + 1 = 0, \quad (9)$$

$$-4x_A^4 + 4x_A^3x + y_A - y = 0, \quad (10)$$

$$-4x_B^4 + 4x_B^3x + y_B - y = 0. \quad (11)$$

After eliminating all variables but x and y from this system by using a CAS, we obtain the equation

$$(65536x^6 + 196608x^4y^2 + 196608x^2y^4 - 41472x^2y + 65536y^6 + 13824y^3 + 729) \cdot \\ (16777216x^6y^3 + 50331648x^4y^5 + 5308416x^4y^2 + 50331648x^2y^7 + \\ 5308416x^2y^4 + 559872x^2y + 16777216y^9 - 1769472y^6 - 186624y^3 + 19683) = 0.$$

Denote the factors of the left-hand side by f_1 and f_2 ; both are irreducible over the integers, but they are reducible over the complex numbers, namely:

$$f_2 = 16777216 \left(x^2y + y^3 - 3/8y^2\sqrt[3]{2} - \frac{9y\sqrt[3]{4}}{64} + \frac{27}{256} \right) \cdot \\ \left(x^2y + y^3 - 3/8y^2 \left(-1/2\sqrt[3]{2} - i/2\sqrt[3]{2}\sqrt{3} \right) - \frac{9y \left(-1/2\sqrt[3]{2} - i/2\sqrt[3]{2}\sqrt{3} \right)^2}{64} + \frac{27}{256} \right) \cdot \\ \left(x^2y + y^3 - 3/8y^2 \left(-1/2\sqrt[3]{2} + i/2\sqrt[3]{2}\sqrt{3} \right) - \frac{9y \left(-1/2\sqrt[3]{2} + i/2\sqrt[3]{2}\sqrt{3} \right)^2}{64} + \frac{27}{256} \right)$$

and

$$\begin{aligned}
 f_1 = & 65536 \cdot \\
 & \cdot (x + ar_1(br_1^4 + cr_1^2 + d)y - er_1(fr_1^4 + gr_1 + h)) \cdot \\
 & \cdot (x - ar_1(br_1^4 + cr_1^2 + d)y - er_1(fr_1^4 + gr_1 + h)) \cdot \\
 & \cdot (x - ar_2(br_2^4 + cr_2^2 + d)y + er_2(fr_2^4 + gr_2 + h)) \cdot \\
 & \cdot (x + ar_2(br_2^4 + cr_2^2 + d)y + er_2(fr_2^4 + gr_2 + h)) \cdot \\
 & \cdot (x + ar_3(br_3^4 + cr_3^2 + d)y + er_3(fr_3^4 + gr_3 + h)) \cdot \\
 & \cdot (x + ar_4(br_4^4 + cr_4^2 + d)y - er_4(fr_4^4 + gr_4 + h)),
 \end{aligned}$$

where $a = 1/4301158990$, $b = 48$, $c = 163867$, $d = 251326464$, $e = 1/17204635960$, $f = 768$, $g = 2621872$, $h = 1870643929$, and r_1 is a root of $z^6 + 3072z_4 + 3135360z^2 + 1077283684$, r_2 is a root of $2150579495z^2 + (2304r_1^5 + 7865616r_1^3 + 9913090777r_1)z - 787728r_1^4 - 2689221337r_1^2 - 4124518600704$, and $r_3 = -j/m \cdot r_1^5 - k/m \cdot r_1^3 - l/m \cdot r_1 - r_2$, where $j = 2304$, $k = 7865616$, $l = 9913090777$, $m = 2150579495$, and $r_4 = -r_3$. This output is provided by Maple.

Now by plotting the result obtained by the elimination and the first factor of f_2 we obtain two overlapping curves, giving an experimental evidence⁴ that all other factors appearing play no role in the geometry. Moreover, if we plot one point on the curve and tangents through this point to the given quartic, moving the point along the curve leaves the tangents perpendicular. That is, the implicit equation of the orthoptic is

$$x^2y + y^3 - 3/8y^2\sqrt[3]{2} - \frac{9y\sqrt[3]{4}}{64} + \frac{27}{256} = 0,$$

namely, a cubic.

In the online applet it is also possible to change θ by dragging the slider. However, for this quartic input polynomial other angles than 90 degrees may request heavy computations.

Example 2. By using the same method we can obtain the orthoptic of a more general open quartic, namely the curve whose equation is $y = x^4 - x$ (see Fig. 8 and <https://www.geogebra.org/m/mfrwfGNc>).

Example 3. Luckily, quadratic input polynomials can be visualized in a quick enough way by using the slider. Figure 9 shows the 35°-isoptic of a hyperbola, for instance, in the web version of GeoGebra. An implicit equation of the curve, computed by GeoGebra, in this case, is

⁴ Valid according to the numerical precision of GeoGebra.

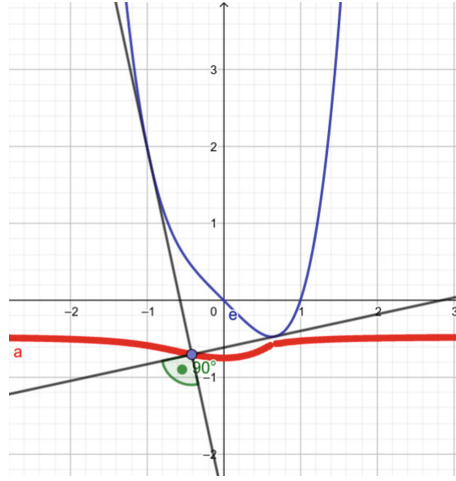


Fig. 8. Orthoptic of the graph given by the quartic equation $y = x^4 - x$

$$\begin{aligned}
 &2x^{14} - 2y^{14} - c^2x^{12} - c^2y^{12} - 10x^2y^{12} - 18x^4y^{10} - 10x^6y^8 \\
 &+ 10x^8y^6 + 18x^{10}y^4 + 10x^{12}y^2 - 6c^2x^2y^{10} - 15c^2x^4y^8 \\
 &- 20c^2x^6y^6 - 15c^2x^8y^4 - 6c^2x^{10}y^2 - 23x^{12} - 23y^{12} + 12c^2x^{10} \\
 &- 12c^2y^{10} - 58x^2y^{10} - 25x^4y^8 + 20x^6y^6 - 25x^8y^4 - 58x^{10}y^2 \\
 &- 36c^2x^2y^8 - 24c^2x^4y^6 + 24c^2x^6y^4 + 36c^2x^8y^2 + 112x^{10} \\
 &- 112y^{10} - 60c^2x^8 - 60c^2y^8 - 80x^2y^8 + 32x^4y^6 - 32x^6y^4 \\
 &+ 80x^8y^2 - 48c^2x^2y^6 + 24c^2x^4y^4 - 48c^2x^6y^2 - 300x^8 - 300y^8 \\
 &+ 160c^2x^6 - 160c^2y^6 + 144x^2y^6 - 136x^4y^4 + 144x^6y^2 + 96c^2x^2y^4 \\
 &- 96c^2x^4y^2 + 480x^6 - 480y^6 - 240c^2x^4 - 240c^2y^4 + 544x^2y^4 \\
 &- 544x^4y^2 + 288c^2x^2y^2 - 464x^4 - 464y^4 + 192c^2x^2 \\
 &- 192c^2y^2 + 608x^2y^2 - 64c^2 + 256x^2 - 256y^2 - 64 = 0,
 \end{aligned}$$

where $c = \cos^2\left(\frac{7}{36}\pi\right)$. This can be simplified (after factorization over \mathbb{C} with Maple, and keeping just the real curve) to

$$cx^4 + 2cx^2y^2 + cy^4 - x^4 - 2x^2y^2 - 4cx^2 - y^4 + 4cy^2 + 4c = 0,$$

that is, the isoptic curve is a quartic. (We note that this curve contains also the set of points for the 145° -isoptic as well, namely, the points which are “close enough” to the hyperbola, deliver the 145° -isoptic, and the others, which are “far enough”, produce the 35° -isoptic. In our theoretical work, these two parts of the curve cannot be separated, because of the squaring of the cosine.)

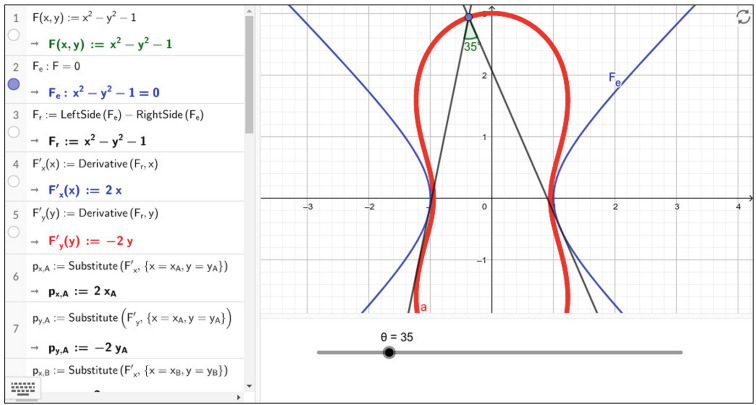


Fig. 9. 35°-isoptic of the function $x^2 - y^2 = 1$

4 The Second Approach

Dynamic geometry systems, including GeoGebra, deal with the notion of free and dependent objects. *Free* points, given in a plane, for example, are objects that are free to change by user interaction, mostly by dragging them with the mouse. On the other hand, *dependent* objects cannot be changed directly—they will change only when their parent objects are changed. In this way the layout of a construction is completely determined by its free points.

Consider a given input point \mathcal{I} , either as a free point, or on a dependent line or circle path \mathcal{P} . Moreover, assume some construction steps, describing the connections between the free and dependent objects, are also given. The user claims a Boolean condition \mathcal{B} holds on some objects of the construction. The task is to determine an equation \mathcal{E} such that for all points \mathcal{I}' of it, if $\mathcal{I} = \mathcal{I}'$, then \mathcal{B} holds. Now \mathcal{E} is called *implicit locus equation*, and its graphical representation is the locus. We emphasize that this task is called automatic discovery in [1].

The GeoGebra syntax of the command is `LocusEquation(<Boolean Expression>, <Point>)`.

In this way, the orthoptic of a given conic can be obtained in the following way:

1. Construct a conic, for example a circle \mathcal{C} , by using the appropriate tool from the Toolbar.
2. Create an arbitrary point P which is outside the circle.
3. Create the tangents f and g from P to \mathcal{C} .
4. Type the command `LocusEquation(f ⊥ g, P)`.

Now the orthoptic of the circle \mathcal{C} will be obtained by an algebraic equation a and its graphical representation is shown in the Graphics View in GeoGebra (Fig. 10, see also <https://www.geogebra.org/m/z2uNpHCU>).

In this way, by having a simple sequence of steps, orthoptics are easy to introduce at high school level.

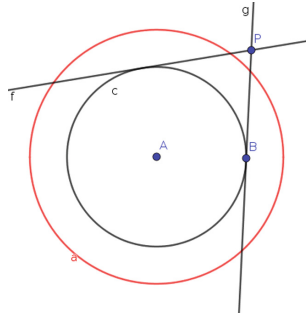


Fig. 10. The orthoptic of a circle

4.1 Implementation Issues

As of April 2018 we successfully implemented tangent equations for circles and parabolas in GeoGebra, and an on-going work is progressing for other conics.

In GeoGebra conics are defined with 6 coordinates: parabolas are, as mentioned above, described with the focus and the directrix (recall that a line is defined by two points on it); and hyperbolas and ellipses are defined by two foci and a point on the curve. Here we give a brief discussion of the implementation of tangent equations for parabolas.

This idea is very natural and has already been introduced by [3,17] in mechanical proofs in planar geometry. However, there are no examples mentioned on conics other than circles. Also, we highlight that in GeoGebra all algebraic translations are made automatically by the computer, that is, all equations should be written in a general but minimal way—we want to avoid computational difficulties.

This system requires handling both the inputs and outputs of the objects to be points. In this sense, the tangent of a parabola has 4 input points: The focus F of the parabola, and two points D_1 and D_2 of the directrix, and some point T_1 on the tangent line. As an output in this case we require two points: T_1 and some other point T_2 on the tangent line. Note that T_1 could be the tangent point T , or it could be some external point P . In the following, we consider the second case, where T_1 is some external point P (see Fig. 11).

Denote the coordinates of a given point A by (x_A, y_A) . We have the following constraints, by using elementary analytic geometry:

$$\begin{vmatrix} x_{D_1} & y_{D_1} & 1 \\ x_{D_2} & y_{D_2} & 1 \\ x_{F'} & y_{F'} & 1 \end{vmatrix} = 0, \quad (12)$$

$$x_M = \frac{x_F + x_{F'}}{2}, \quad (13)$$

$$y_M = \frac{y_F + y_{F'}}{2}, \quad (14)$$

$$(x_P - x_F)^2 + (y_P - y_F)^2 = (x_P - x_{F'})^2 + (y_P - y_{F'})^2, \quad (15)$$

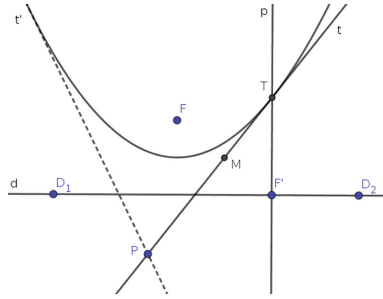


Fig. 11. Tangent to a parabola from an external point P

which describe (12) that $F' \in d$, (13) and (14) that M is the midpoint of FF' , and (15) that $PF = PF'$. Now by setting $T_1 = P$ and $T_2 = M$ as output, we implicitly define the line T_1T_2 , by using the 4 constraints above.

This approach is, however, not completely correct. In the case $M = P$ the output is not correctly defined. This case occurs when P lies on the tangent that is parallel to the directrix. Therefore, we may have a better constraint by considering the following equations as well:

$$(x_{D_1} - x_{F'}) \cdot (x_T - x_{F'}) + (y_{D_1} - y_{F'}) \cdot (y_T - y_{F'}) = 0, \tag{16}$$

$$(x_F - x_{F'}) \cdot (x_T - x_P) + (y_F - y_{F'}) \cdot (y_T - y_P) = 0, \tag{17}$$

which ensure that $F'T \perp d$ (16) and $FF' \perp PT$ (17). Here we no longer use (13) and (14), but the other equations are kept, that is, we still use 4 constraints, but with the output $T_1 = P$ and $T_2 = T$.

Unfortunately, we will still have two problems with the new approach. First, we use two quadratic equations now instead of two linear ones and this will make the computational complexity harder. Second, we still allow $P = T$. This will result in getting the whole parabola as well in the locus, not just the tangent. Therefore we need another constraint to force $P \neq T$ in the following way:

$$z \cdot ((x_P - x_T)^2 + (y_P - y_T)^2) = 1, \tag{18}$$

because we want (x_P, y_P) to be different from (x_T, y_T) , so we require $(x_P - x_T)^2 + (y_P - y_T)^2$ to be nonzero, i.e. invertible. At this stage, the constraints (12), (15), (16), (17) and (18) describe the output tangent line as expected for use in dynamic geometry.

4.2 Isoptics

It would be desirable to generalize the dynamic geometry approach to isoptics as well. Unfortunately, in GeoGebra it is complicated to express the equality of

angles in the condition of an implicit locus equation. The currently supported syntax to check if $\alpha = \beta$ is `AreCongruent(α, β)`.

In this way, it is already possible to study isoptic curves as well, in some basic cases. Figure 12 demonstrates the idea. That is, by sketching up, say, a circle with center A and circumpoint B , we can create external point P with tangents f and g , similarly to Fig. 10. Now we designate two arbitrary points C and D on tangents and construct two segments EF and FG , and define angles $\alpha = \angle EFG$ and $\beta = \angle CPD$. Finally the command `LocusEquation(AreCongruent(α, β), P)` is issued which means that *we search for all points P such that $\alpha = \beta$ is ensured*. In our example this command yields a sextic implicit curve, containing two circles and two lines—these latter ones should be ignored, because, due to technical reasons, they belong to a larger set (including the points C and D). Also, the inner circle is the 135° -isoptic of the circle, since the angles 45° and 135° are not distinguishable in the complex algebraic geometry setup.

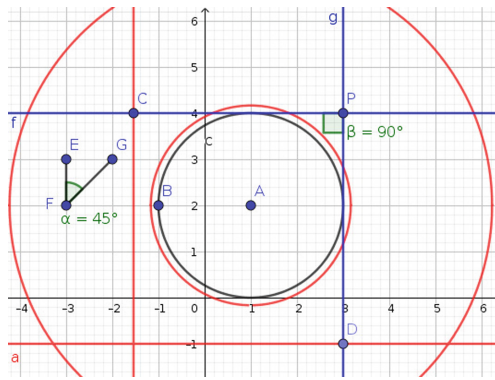


Fig. 12. 45° -isoptic of the circle with dynamic geometry

Anyway, the user can experiment further with this applet, in particular by changing the angle α , in a way similar to Sect. 3, where a slider controlled the animation. Here, however, moves are slower, due to the higher amount of computations, handling every detail. In general, about 1.5 s are required to compute each new frame.

Our final example, shown in Fig. 13, demonstrates the 135° -isoptic of a parabola. In fact, GeoGebra computes 45° - and 135° -isoptics at the same time (the former is the “bottom” branch of the hyperbola, not shown in the Figure)⁵. Again, two extra lines appear, that should be ignored.

This figure can be obtained, however, only in a much bigger amount of time than for the circle. For one curve to be shown, the user has to wait at least 5 s on a modern computer. This makes yet difficult to study a set of θ -isoptics in a quick and convenient way at the moment.

⁵ For the same reason, with another CAS, the name *bisoptic* has been coined in [7].

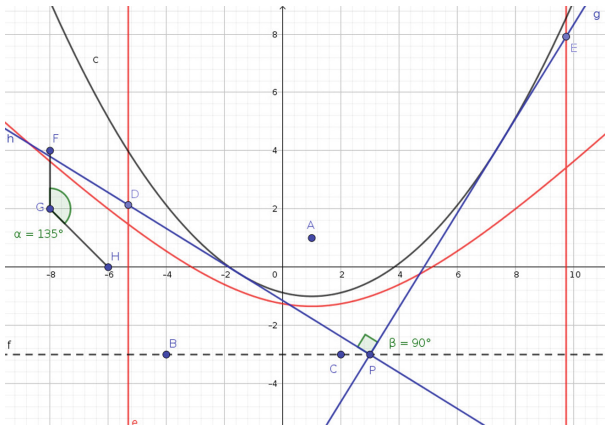


Fig. 13. 135° -isoptic of the parabola with dynamic geometry

5 Conclusion

Isoptic curves are generally considered as a topic for researchers and advanced students having taken at least a first course in Differential Geometry. Previous works, mentioned in particular in Sect. 2, introduced algebraic methods. For conic sections, methods from elementary algebra could be applied, but they cannot be generalized to curves of higher degree.

By using both of our methods for a curve defined by a polynomial implicit equation, we considered ideals generated by the involved polynomials. Computing elimination ideals was efficient, and therefore, it can be preferred to previous works. Our methods, however, cannot work with non-polynomial data.

We highlight that the approaches which we propose may make the topic manageable for students at an earlier stage. We do not intend to have a usage of technology as a black box, but rather to provide tools and incitation to learn more advanced mathematics, as in [5]. The interplay between Computer Algebra and Dynamical Geometry on the one hand, and the switching between different registers of representation (namely algebraic and graphical) on the other hand, makes the study an experimental one. Nevertheless, the user should not forget that behind the scene, the computer uses also a numerical register of representation, in particular, for the graphical representation.

Also, some ambiguous equations may require manual cancelling for the unnecessary curves which can appear within the automated process, as mentioned in Sect. 4. Also, in Sect. 3 we found a curve which delivers solutions for two different problems, and the results are not distinguishable in our theory. (See also [11] for some other, simpler issues in complex algebraic geometry, that are unavoidable in this approach.)

Acknowledgements. Second author partially supported by the grant MTM2017-88796-P from the Spanish MINECO (Ministerio de Economía y Competitividad) and the ERDF (European Regional Development Fund).

References

1. Abánades, M.A., Botana, F., Kovács, Z., Recio, T., Sólyom-Gecse, C.: Development of automatic reasoning tools in GeoGebra. In: Conference ACM Communication in Computer Algebra and Software Demonstration at the ISSAC 2016, vol. 50, no. 3, pp. 85–88 (2016)
2. Abánades, M.A., Botana, F., Montes, A., Recio, T.: An algebraic taxonomy for locus computation in dynamic geometry. *Comput.-Aided Des.* **56**, 22–33 (2014)
3. Chou, S.-C.: *Mechanical Geometry Theorem Proving*. Reidel, Dordrecht (1987)
4. Cox, D., Little, J., O’Shea, D.: *Ideals, Varieties and Algorithms*, Third edn. Springer, Heidelberg (2007). <https://doi.org/10.1007/978-0-387-35651-8>
5. Dana-Picard, T.: Technology as a bypass for a lack of theoretical knowledge. *Int. J. Tech. Math. Educ.* **11**(3), 101–109 (2005)
6. Dana-Picard, T., Mann, G., Zehavi, N.: From conic intersections to toric intersections: the case of the isoptic curves of an ellipse. *Montana Math. Enthusiast* **9**(1), 59–76 (2011)
7. Dana-Picard, T., Zehavi, N., Mann, G.: Bisoptic curves of hyperbolas. *Int. J. Math. Educ. Sci. Technol.* **45**(5), 762–781 (2014)
8. Dana-Picard, T., Naiman, A.: Isoptics of Fermat curves. Preprint, JCT (2018)
9. Dana-Picard, T.: Automated study of isoptic curves of an astroid. Preprint, JCT (2018)
10. Kovács, Z., Recio, T., Vélez, M.P.: GeoGebra Automated Reasoning Tools. A Tutorial (2017). <https://github.com/kovzol/gg-art-doc>
11. Kovács, Z., Recio, T., Sólyom-Gecse, C.: Automatic rewrites of input expressions in complex algebraic geometry provers. In: Narboux, J., Schreck P., Streinu, E. (eds.) *Proceedings of ADG 2016*, pp. 137–143 (2016)
12. Kovács, Z.: Real-time animated dynamic geometry in the classrooms by using fast Gröbner basis computations. *Math. Comput. Sci.* **11**(3–4), 351–361 (2017)
13. Kovács, Z., Parisse, B.: Giac and GeoGebra - improved Gröbner basis computations. In: Gutierrez, J., Schicho, J., Weimann, M. (eds.) *Computer Algebra and Polynomials*. LNCS, vol. 8942, pp. 126–138. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-15081-9_7
14. Miernowski, A., Mosgawa, W.: Isoptics of Pairs of Nested Closed Strictly Convex Curves and Crofton-Type Formulas, *Beiträge zur Algebra und Geometrie*. Contributions to Algebra and Geometry, vol. 42, no. 1, pp. 281–288 (2001)
15. Szalkowski, D.: Isoptics of open rosettes, *Annales Universitatis Mariae Curie-Skłodowska, Lublin - Polonia* LIX, Section A, pp. 119–128 (2005)
16. Vélez, M.P., Recio, T.: Automatic discovery of theorems in elementary geometry. *J. Autom. Reason.* **23**, 63–82 (1999)
17. Wen-Tsün, W.: On the decision problem and the mechanization of theorem-proving in elementary geometry. *Sci. Sinica* **21**, 159–172 (1978)



Biform Theories: Project Description

Jacques Carette, William M. Farmer^(✉), and Yasmine Sharoda

Computing and Software, McMaster University, Hamilton, Canada
wmfarmer@mcmaster.ca

<http://www.cas.mcmaster.ca/~curette>, <http://imps.mcmaster.ca/wmfarmer>

Abstract. A *biform theory* is a combination of an axiomatic theory and an algorithmic theory that supports the integration of reasoning and computation. These are ideal for specifying and reasoning about algorithms that manipulate mathematical expressions. However, formalizing biform theories is challenging as it requires the means to express statements about the interplay of what these algorithms do and what their actions mean mathematically. This paper describes a project to develop a methodology for expressing, manipulating, managing, and generating mathematical knowledge as a network of biform theories. It is a subproject of MathScheme, a long-term project at McMaster University to produce a framework for integrating formal deduction and symbolic computation.

We present the *Biform Theories* project, a subproject of MathScheme [11] (a long-term project to produce a framework integrating formal deduction and symbolic computation).

1 Motivation

Type $2 * 3$ into your favourite computer algebra system, press enter, and you will receive (unsurprisingly) 6. But what if you want to go in the opposite direction? Easy: you ask `ifactors(6)` in Maple or `FactorInteger[6]` in Mathematica.¹ The Maple command `ifactors` returns a 2-element list, with the first element the unit (1 or -1) and the second element a list of pairs (encoded as two-element lists) with (distinct) primes in the first component and the prime's multiplicity in the second. Mathematica's `FactorInteger` is similar, except that it omits the unit (and thus does not document what happens for negative integers).

This simple example illustrates the difference between a simple computation $2 * 3$ and a more complex *symbolic* query, factoring. The reason for using lists-of-lists in both systems is that multiplication and powering are both functions that evaluate immediately in these systems. So that factoring 6 cannot just return $2^1 * 3^1$, as that simply evaluates to 6. Thus it is inevitable that

This research is supported by NSERC.

¹ Other computer algebra systems have similar commands.

both systems must *represent* multiplication and powering in some other manner. Because `ifactors` and `FactorInteger` are so old, they are unable to take advantage of newer developments in both systems, in this case a feature to not immediately evaluate an expression but leave it as a representation of a future computation. Maple calls this feature an *inert form*, while in Mathematica it is a *hold form*. Nevertheless, the need for representing future computations was recognized early on: even in the earliest days of Maple, one could do `5 &^256 mod 379` to compute the answer without ever computing 5^{256} over the integers. In summary, this example shows that in some cases we are interested in `2 * 3` for its value and in other cases we are interested in it for its syntactic structure.

A legitimate question would be: Is this an isolated occurrence, or a more pervasive pattern? It is pervasive. It arises from the dichotomy of being able to *perform* computations and being able to *talk about* (usually to prove the correctness of) computations. For example, we could represent (in Haskell) a tiny language of arithmetic as

```
data Arith =
  Int Integer
  | Plus Arith Arith
  | Times Arith Arith
```

and an evaluator as

```
eval :: Arith -> Integer
eval (Int x) = x
eval (Plus a b) = eval a + eval b
eval (Times a b) = eval a * eval b
```

whose “correctness” seems self-evident. But what if we had instead written

```
data AA = TTT Integer | XXX AA AA | YYY AA AA

eval' :: AA -> Integer
eval' (TTT x) = x
eval' (XXX a b) = eval' a * eval' b
eval' (YYY a b) = eval' a + eval' b
```

how would we know if this implementation of `eval'` is correct or not? The two languages are readily seen to be isomorphic. In fact, there are clearly *two* different isomorphisms. As the symbols used are no longer mnemonic, we have no means to (informally!) decide whether `eval'` is correct. Nevertheless, `Arith` and `AA` both represent (trivial) embedded *domain specific languages* (DSLs), which are pervasively used in computing. Being able to know that a function defined over a DSL is correct is an important problem.

In general, both computer algebra systems (CASs) and theorem proving systems (TPSS) manipulate *syntactic representations* of mathematical knowledge. But they tackle the same problems in different ways. In a CAS, it is a natural question to take a polynomial p (in some representation that the system recognizes as being a polynomial) and ask to factor it into a product of irreducible polynomials [46]. The algorithms to do this have gotten extremely sophisticated

over the years [35]. In a TPS, it is more natural to prove that such a polynomial p is equal to a particular factorization, and perhaps also prove that each such factor is irreducible. Verifying that a given factorization is correct is, of course, easy. Proving that factors are irreducible can be quite hard. And even though CASs obtain information that would be helpful to a TPS towards such a proof, that information is usually not part of the output. Thus while some algorithms for factoring do produce irreducibility *certificates*, which makes proofs straightforward, these are usually not available. And the complexity of the algorithms (from an engineering point of view) is sufficiently daunting that, as far as we know, no TPS has re-implemented them.

Given that both CASs and TPSs “do mathematics”, why are they so different? Basically because a CAS is based around *algorithmic theories*, which are collections of symbolic computation algorithms whose correctness has been established using pen-and-paper mathematics, while a TPS is based around *axiomatic theories*, comprised of signatures and axioms, but nevertheless representing the “same” mathematics. In a TPS, one typically proves theorems, formally. There is some cross-over: some TPSs (notably Agda and Idris) are closer to programming languages, and thus offer the real possibility of mixing computation and deduction. Nevertheless, the problem still exists: how does one verify that a particular function implemented over a representation language carries out the desired computation?

What is needed is a means to *link* together axiomatic theories and algorithmic theories such that one can state that some “symbol manipulation” corresponds to a (semantic) function defined axiomatically? In other words, we want to know that a *symbolic computation* performed on representations performs the same computation as an abstract function defined on the *denotation* of those representations. For example, if we ask to integrate a particular expression e , we would like to know that the system’s response will in fact be an expression representing an integral of e —even if the formal definition of integration uses an infinitary process.

These kinds of problems are pervasive: not just closed-form symbolic manipulations, but also SAT solving, SMT solving, model checking, type-checking of programs, and most manipulations of DSL terms, are all of this sort. They all involve a mixture of computation and deduction that entwine syntactic representations with semantic conditions.

In the next section we will introduce the notion of a *biform theory* that is a combination of an axiomatic theory and an algorithmic theory so that we can define and reason about symbolic computation in the same setting.

2 Background Ideas

A *transformer* is an algorithm that implements a function $\mathcal{E}^n \rightarrow \mathcal{E}$ where \mathcal{E} is a set of expressions. The expressions serve as data that can be manipulated. Different kinds of expressions correspond to different data representations. Transformers can manipulate expressions in various ways. Simple transformers,

for example, build bigger expressions from pieces, select components of expressions, or check whether a given expression satisfies some syntactic property. More sophisticated transformers manipulate expressions in mathematically meaningful ways. We call these kinds of transformers *syntax-based mathematical algorithms (SBMAs)* [27]. Examples include algorithms that apply arithmetic operations to numerals, factor polynomials, transpose matrices, and symbolically differentiate expressions with variables. The *computational behavior* of a transformer is the relationship between its input and output expressions. When the transformer is an SBMA, its *mathematical meaning*² is the relationship between the mathematical meanings of its input and output expressions.

A *biform theory* T is a triple (L, Π, Γ) where L is a language of some underlying logic, Π is a set of transformers that implement functions on expressions of L , and Γ is a set of formulas of L [6, 25, 31]. L includes, for each transformer $\pi \in \Pi$, a name for the function implemented by π that serves as a name for π . The members of Γ are the *axioms* of T . They specify the meaning of the nonlogical symbols in L including the names of the transformers of T . In particular, Γ may contain specifications of the computational behavior of the transformers in Π and of the mathematical meaning of the SBMAs in Π . A formula in Γ that refers to the name of a transformer $\pi \in \Pi$ is called a *meaning formula* for π . The transformers in Π may be written in the underlying logic or in a programming language external to the underlying logic. We say T is an *axiomatic theory* if Π is empty and an *algorithmic theory* if Γ is empty.

Example 1. Let $R_{\text{ax}} = (L, \Gamma)$ be a first-order axiomatic theory of a ring with identity. The language L contains the usual constants (0 and 1), function symbols (+ and *), and predicate symbols (=), and Γ contains the usual axioms. The terms of L , which are built from 0, 1, and variables by applying + and *, have the form of multivariate polynomials. Thus algorithms that manipulate polynomials—that normalize a polynomial, factor a polynomial, find the greatest common divisor of two polynomials, etc.—would naturally be useful for reasoning about the expressions of R_{ax} . Let Π be a set of such transformers on the terms in L , L' be an extension of L that includes vocabulary for naming and specifying the transformers in Π , and Γ' contain meaning formulas for the transformers in Π expressed in L' . Then $R_{\text{bt}} = (L', \Pi, \Gamma \cup \Gamma')$ is a biform theory for rings with identity. It would be very challenging to express R_{bt} in ordinary first-order logic; the meaning formulas in Γ' would be especially difficult to express. Notice that $R_{\text{alg}} = (L', \Pi)$ is algorithmic theory of multivariate polynomials with constants 0 and 1.

Formalizing a biform theory in the underlying logic requires infrastructure for reasoning about the expressions manipulated by the transformers as syntactic entities. This infrastructure provides a basis for *metareasoning with reflection* [29]. There are two main approaches to build such an infrastructure [27]. The *local approach* is to produce a deep embedding of a sublanguage L' of L that

² Computer scientists would call this *denotational semantics* rather than *mathematical meaning*.

includes all the expressions manipulated by the transformers of Π . The *global approach* is to replace the underlying logic of L with a logic such as CTT_{qe} [29] that has an inductive type of *syntactic values* that represent the expressions in L and global quotation and evaluation operators. A third approach, based on “final tagless” embeddings [15], has not yet been attempted as most logics do not have the necessary infrastructure to abstract over type constructors.

A complex body of mathematical knowledge can be represented in accordance with the *little theories method* [30] (or even the *tiny theories method* [18]) as a *theory graph* [36] consisting of axiomatic theories as nodes and theory morphisms as directed edges. A *theory morphism* is a meaning-preserving mapping from the formulas of one axiomatic theory to the formulas of another. The theories—which may have different underlying logics—serve as abstract mathematical models, and the morphisms serve as information conduits that enable theory components such as definitions and theorems to be transported from one theory to another [2]. A theory graph enables mathematical knowledge to be formalized in the most convenient underlying logic at the most convenient level of abstraction using the most convenient vocabulary. The connections made by the theory morphisms in a theory graph then provide the means to find this knowledge and apply it in other contexts.

A *biform theory graph* is a theory graph whose nodes are biform theories. Having the same benefits as theory graphs of axiomatic theories, biform theory graphs are well suited for representing mathematical knowledge that is expressed both axiomatically and algorithmically.

Our previous work on mechanized mathematics systems and on related technologies has taught us that such a graph of biform theories really should be a central component of any future systems for mathematics. We will expand on the objectives of the project and its current state. At the same time, additional pieces of the project beyond what is motivated above (but is motivated by previous and related work) will be weaved in as appropriate.

3 Project Objectives

The primary objective of the Biform Theories project is:

Primary. Develop a methodology for expressing, manipulating, managing and generating mathematical knowledge as a biform theory graph.

Our strategy for achieving this is to break down the problem into the following subprojects:

Logic. Design a logic Log which is a version of simple type theory [26] with an inductive type of syntactic values, a global quotation operator, and a global evaluation operator. In addition to a syntax and semantics, define a proof system for Log and a notion of a theory morphism from one axiomatic theory of Log to another. Demonstrate that SBMAs can be defined in Log and that their mathematical meanings can be stated, proved, and instantiated using Log ’s proof system.

Implementation. Produce an implementation `Impl` of `Log`. Demonstrate that SBMAs can be defined in `Impl` and that their mathematical meanings can be stated and proved in `Impl`.

Transformers. Enable biform theories to be defined in `Impl`. Introduce a mechanism for applying transformers defined outside of `Impl` to expressions of `Log`. Ensure that we know how to write meaning formulas for such transformers. Some transformers can be automatically generated—investigate the scope of this, and implement those which are feasible.

Theory Graphs. Enable theory graphs of biform theories to be defined in `Impl`. Use combinators to ease the construction of large, structured biform theory graphs. Introduce mechanisms for transporting definitions, theorems, and transformers from a biform theory T to an instance T' of T via a theory morphism from T to T' . Some theories (such as theories of homomorphisms and term languages) can be and thus should be automatically generated.

Generic Transformers. Design and implement in `Impl` a scheme for defining generic transformers in a theory graph T that can be specialized, when transported to an instance T' of T , using the properties exhibited in T' .

4 Work Plan Status

The work plan is to pursue the five subprojects described above more or less in the order of their presentation. Here we describe the parts of the work plan that have been completed as well as the parts that remain to be done.

Logic with Quotation and Evaluation

This subproject is largely complete. We have developed `CTTqe` [29], a version of Church’s type theory [22] with global quotation and evaluation operators. (Church’s type theory is a popular form of simple type theory with lambda notation.) The syntax of `CTTqe` has the machinery of \mathcal{Q}_0 [1], Andrews’ version of Church’s type theory plus an inductive type ϵ of syntactic values, a partial quotation operator, and a typed evaluation operator. The semantics of `CTTqe` is based on Henkin-style general models [34]. The proof system for `CTTqe` is an extension of the proof system for \mathcal{Q}_0 .

We show in [29] that `CTTqe` is suitable for defining SBMAs and stating, proving, and instantiating their mathematical meanings. In particular, we prove within the proof system for `CTTqe` the mathematical meaning of a symbolic differentiation algorithm for polynomials.

We have also defined `CTTuqe` [28], a variant of `CTTqe` in which undefinedness is incorporated in `CTTqe` according to the traditional approach to undefinedness [24]. Better suited than `CTTqe` as a logic for interconnecting axiomatic theories, we have defined in `CTTuqe` a notion of a theory morphism [28].

Implementation of the Logic

We have produced an implementation of CTT_{qe} called HOL Light QE [10] by modifying HOL Light [33], an implementation of the HOL proof assistant [32]. HOL Light QE provides a built-in global infrastructure for metareasoning with reflection. Over the next couple years we plan to test this infrastructure by formalizing a variety of SBMAs in HOL Light QE.

Building on the experience we gain in the development of HOL Light QE, we would like to create an implementation of CTT_{qe} in MMT [44] that is well suited for general use and has strong support for building theory graphs. We will transfer to this MMT implementation the most successful of the ideas and mechanisms we develop on the three subprojects that follow using HOL Light QE.

Biform Theories, Transformers, and Generation

Implementation of biform theories in HOL Light QE has not yet started, but we expect that it will be straightforward, as will the application of external transformers. External transformers implemented in OCaml (or in languages reachable via OCaml’s foreign function interface) can be linked in as well.

The most difficult part of this subproject will be adequate renderings of *meaning formulas* that express the mathematical meaning of transformers. We do have some experience [7, 12] creating biform theories. The exploration and implementation of automatic generation of transformers has started.

Biform Theory Graphs

In [7], we developed a case study of a biform theory graph consisting of eight biform theories encoding natural number arithmetic. We produced partial formalizations of this test case [7] in CTT_{uqe} [28] using the global approach for metareasoning with reflection, and in Agda [42, 43] using the local approach. After we have finished with the previous two subprojects, we intend to formalize this in HOL Light QE as well.

In [18], we developed combinators for combining theory presentations. There is no significant difference between axiomatic and biform theories with respect to the semantics of these combinators, and we expect that these will continue to work as well as they did in [8]. There, we also experimented with some small-scale theory generation, which worked well. This subproject will also encompass the implementation of *realms* [9]. We also hope to make some inroads on *high level theories* [6].

Generic, Specializable Transformers

Through substantial previous work [4, 5, 8, 13–17, 19, 38, 39, 41] on code generation and code manipulation, it has become quite clear that quite a lot of mathematical code can be automatically generated. One of the most successful techniques is *instantiation*, whereby a single, generic algorithm exposes a series of *design*

choices that must be explicitly instantiated to produce specialized code. By clever choices of design parameters, and through the use of partial evaluation, one can thus produce highly optimized code without having to hand-write such code.

5 Related Work

Directly related is [37] whose authors also work with biform theory graphs. Michael Kohlhase and Florian Rabe and their students are actively working on related topics. As a natural progression, we (the authors of this paper) have started actively collaborating with them, under the name of the *Tetrapod Project*.

One of the crucial features for supporting the interplay between syntax and semantics is *reflection*, which has a long history and a deep literature. The interested reader should read the thorough related work section in [29] for more details.

There are substantial developments happening in some systems, most notably Agda [42,43], Idris [3] and Lean [40] that we are paying particularly close attention to. This includes quite a lot of work on making reflection practical [20,21,23,47].

On the more theoretical side, *homotopy type theory* [45] is rather promising. However quite a bit of research still needs to be done to make these results practical. Of particular note is the issue that theories that deal directly with syntax seem to clash with the notion of a *univalent universe*, which is central to homotopy type theory.

6 Conclusion

Building mechanized mathematics systems is a rather complex engineering task. It involves creating new science—principally through the creation of logics which can support reasoning about syntax. It also involves significant new engineering—both on the systems side, where *knowledge management* is crucial to reduce the *information duplication* inherent in a naive implementation of mathematics, and on the usability front, where users do not, and should not, care about all the infrastructure that developers need to create their system. Current systems tend to expose this infrastructure, thus creating an additional burden for casual users who may well have a simple task to perform.

The *Biform Theories* project is indeed about infrastructure that we believe is essential to building large-scale mechanized mathematics systems. And yes, we do believe that eventual success would imply that casual users of such a system never hear of “biform theories”.

Acknowledgments. This research was supported by NSERC. The authors would like to thank the referees for their comments and suggestions.

References

1. Andrews, P.B.: *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof*, 2nd edn. Kluwer, Dordrecht (2002)
2. Barwise, J., Seligman, J.: *Information Flow: The Logic of Distributed Systems*, Tracts in Computer Science, vol. 44. Cambridge University Press, Cambridge (1997)
3. Brady, E.: Idris, a general-purpose dependently typed programming language: design and implementation. *J. Funct. Program.* **23**, 552–593 (2013). <https://doi.org/10.1017/S095679681300018X>
4. Carette, J.: Gaussian elimination: a case study in efficient genericity with MetaOCaml. *Sci. Comput. Program.* **62**, 3–24 (2006). Special Issue on the First MetaOCaml Workshop 2004
5. Carette, J., Elsheikh, M., Smith, S.: A generative geometric kernel. In: *Proceedings of the 20th ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2011*, pp. 53–62. ACM, New York (2011). <https://doi.org/10.1145/1929501.1929510>
6. Carette, J., Farmer, W.M.: High-level theories. In: Autexier, S., Campbell, J., Rubio, J., Sorge, V., Suzuki, M., Wiedijk, F. (eds.) *CICM 2008*. LNCS (LNAI), vol. 5144, pp. 232–245. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85110-3_19
7. Carette, J., Farmer, W.M.: Formalizing mathematical knowledge as a biformal theory graph: a case study. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) *CICM 2017*. LNCS (LNAI), vol. 10383, pp. 9–24. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_2
8. Carette, J., Farmer, W.M., Jeremic, F., Maccio, V., O’Connor, R., Tran, Q.M.: The MathScheme library: some preliminary experiments. Technical report, University of Bologna, Italy (2011). uBLCS-2011-04
9. Carette, J., Farmer, W.M., Kohlhase, M.: Realms: a structure for consolidating knowledge about mathematical theories. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) *CICM 2014*. LNCS (LNAI), vol. 8543, pp. 252–266. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08434-3_19
10. Carette, J., Farmer, W.M., Laskowski, P.: HOL light QE. In: Avigad, J., Mahboubi, A. (eds.) *Interactive Theorem Proving*. LNCS. Springer (2018, forthcoming)
11. Carette, J., Farmer, W.M., O’Connor, R.: MathScheme: project description. In: Davenport, J.H., Farmer, W.M., Urban, J., Rabe, F. (eds.) *CICM 2011*. LNCS (LNAI), vol. 6824, pp. 287–288. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22673-1_23
12. Carette, J., Farmer, W.M., Sorge, V.: A rational reconstruction of a system for experimental mathematics. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) *Calculemus/MKM -2007*. LNCS (LNAI), vol. 4573, pp. 13–26. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73086-6_2
13. Carette, J., Kiselyov, O.: Multi-stage programming with functors and monads: eliminating abstraction overhead from generic code. In: Glück, R., Lowry, M. (eds.) *GPCE 2005*. LNCS, vol. 3676, pp. 256–274. Springer, Heidelberg (2005). https://doi.org/10.1007/11561347_18
14. Carette, J., Kiselyov, O.: Multi-stage programming with functors and monads: eliminating abstraction overhead from generic code. *Sci. Comput. Program.* **76**, 349–375 (2011)

15. Carette, J., Kiselyov, O., Shan, C.: Finally tagless, partially evaluated: tagless staged interpreters for simpler typed languages. *J. Funct. Program.* **19**, 509–543 (2009). <https://doi.org/10.1017/S0956796809007205>
16. Carette, J., Kucera, M.: Partial evaluation for maple. In: *ACM SIGPLAN 2007 Workshop on Partial Evaluation and Program Manipulation*, pp. 41–50 (2007)
17. Carette, J., Kucera, M.: Partial evaluation of maple. *Sci. Comput. Program.* **76**, 469–491 (2011)
18. Carette, J., O’Connor, R.: Theory presentation combinators. In: Jeuring, J., et al. (eds.) *CICM 2012. LNCS (LNAI)*, vol. 7362, pp. 202–215. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31374-5_14
19. Carette, J., Shan, C.C.: Simplifying probabilistic programs using computer algebra. In: Gavanelli, M., Reppy, J. (eds.) *PADL 2016*, vol. 9585, pp. 135–152. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-28228-2_9
20. Christiansen, D., Brady, E.: Elaborator reflection extending Idris in Idris. *SIGPLAN Not.* **51**, 284–297 (2016). <https://doi.org/10.1145/3022670.2951932>
21. Christiansen, D.R.: Type-directed elaboration of quasiquotations: a high-level syntax for low-level reflection. In: *Proceedings of the 26nd 2014 International Symposium on Implementation and Application of Functional Languages, IFL 2014*, pp. 1:1–1:9. ACM, New York (2014). <https://doi.org/10.1145/2746325.2746326>
22. Church, A.: A formulation of the simple theory of types. *J. Symb. Logic* **5**, 56–68 (1940)
23. Ebner, G., Ullrich, S., Roesch, J., Avigad, J., de Moura, L.: A metaprogramming framework for formal verification. In: *Proceedings of the ACM on Programming Languages (ICFP)*, vol. 1, p. 34 (2017)
24. Farmer, W.M.: Formalizing undefinedness arising in calculus. In: Basin, D., Rusinowitch, M. (eds.) *IJCAR 2004. LNCS (LNAI)*, vol. 3097, pp. 475–489. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-25984-8_35
25. Farmer, W.M.: Biform theories in chiron. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) *Calculemus/MKM -2007. LNCS (LNAI)*, vol. 4573, pp. 66–79. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73086-6_6
26. Farmer, W.M.: The seven virtues of simple type theory. *J. Appl. Logic* **6**, 267–286 (2008)
27. Farmer, W.M.: The formalization of syntax-based mathematical algorithms using quotation and evaluation. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (eds.) *CICM 2013. LNCS (LNAI)*, vol. 7961, pp. 35–50. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39320-4_3
28. Farmer, W.M.: Theory Morphisms in Church’s type theory with quotation and evaluation. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) *CICM 2017. LNCS (LNAI)*, vol. 10383, pp. 147–162. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_11
29. Farmer, W.M.: Incorporating quotation and evaluation into Church’s type theory. *Inf. Comput.* **260**, 9–50 (2018)
30. Farmer, W.M., Guttman, J.D., Javier Thayer, F.: Little theories. In: Kapur, D. (ed.) *CADE 1992. LNCS*, vol. 607, pp. 567–581. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55602-8_192
31. Farmer, W.M., von Mohrenschildt, M.: An overview of a formal framework for managing mathematics. *Ann. Math. Artif. Intell.* **38**, 165–191 (2003)
32. Gordon, M.J.C., Melham, T.F.: *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, Cambridge (1993)

33. Harrison, J.: HOL light: an overview. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 60–66. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03359-9_4
34. Henkin, L.: Completeness in the theory of types. *J. Symb. Logic* **15**, 81–91 (1950)
35. van Hoeij, M.: Factoring polynomials and the knapsack problem. *J. Number Theory* **95**, 167–189 (2002). <http://www.sciencedirect.com/science/article/article/pii/S0022314X01927635>
36. Kohlhase, M.: Mathematical knowledge management transcending the one-brain-barrier with theory graphs. *Eur. Math. Soc. Newsl.* **92**, 22–27 (2014)
37. Kohlhase, M., Mance, F., Rabe, F.: A universal machine for biformal theory graphs. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (eds.) CICM 2013. LNCS (LNAI), vol. 7961, pp. 82–97. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39320-4_6
38. Kucera, M., Carette, J.: Partial evaluation and residual theorems in computer algebra. In: Ranise, S., Bigatti, A. (eds.) Proceedings of Calculemus 2006. Electronic Notes in Theoretical Computer Science, Elsevier (2006)
39. Larjani, P.: Software specialization as applied to computational algebra. Ph.D. thesis, McMaster University (2013)
40. de Moura, L., Kong, S., Avigad, J., van Doorn, F., von Raumer, J.: The lean theorem prover. In: Automated Deduction - CADE-25, Proceedings of 25th International Conference on Automated Deduction, Berlin, Germany, 1–7 August 2015 (2015)
41. Narayanan, P., Carette, J., Romano, W., Shan, C., Zinkov, R.: Probabilistic inference by program transformation in Hakaru (system description). In: Kiselyov, O., King, A. (eds.) FLOPS 2016. LNCS, vol. 9613, pp. 62–79. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29604-3_5
42. Norell, U.: Towards a practical programming language based on dependent type theory. Ph.D. thesis, Chalmers University of Technology (2007)
43. Norell, U.: Dependently typed programming in Agda. In: Kennedy, A., Ahmed, A. (eds.) Proceedings of TLDI 2009, pp. 1–2. ACM (2009)
44. Rabe, F., Kohlhase, M.: A scalable model system. *Inf. Comput.* **230**, 1–54 (2013)
45. Univalent Foundations Program: Homotopy Type Theory: Univalent Foundations of Mathematics. Institute for Advanced Study (2013). <https://homotopytypetheory.org/book>
46. Von Zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, Cambridge (2003)
47. van der Walt, P.: Reflection in Agda. Master’s thesis, Universiteit Utrecht (2012)



A Coq Formalization of Digital Filters

Diane Gallois-Wong^{1,2(✉)}, Sylvie Boldo^{3,2(✉)}, and Thibault Hilaire^{3,2,4(✉)}

¹ Université Paris-Sud, Orsay, France

² LRI, CNRS & Université Paris-Sud, Université Paris-Saclay,
bâtiment 650, Université Paris-Sud, 91405 Orsay Cedex, France

`Diane.Gallois-Wong@lri.fr`, `sylvie.boldo@inria.fr`

³ Inria, Saclay, France

⁴ Sorbonne Université, 75005 Paris, France

`thibault.hilaire@lip6.fr`

Abstract. Digital filters are small iterative algorithms, used as basic bricks in signal processing (filters) and control theory (controllers). They receive as input a stream of values, and output another stream of values, computed from their internal state and from the previous inputs. These systems can be found in communication, aeronautics, automotive, robotics, etc. As the application domain may be critical, we aim at providing a formal guarantee of the good behavior of these algorithms in time-domain. In particular, we formally proved in Coq some error analysis theorems about digital filters, namely the Worst-Case Peak Gain theorem and the existence of a filter characterizing the difference between the exact filter and the implemented one. Moreover, the digital signal processing literature provides us with many equivalent algorithms, called realizations. We formally defined and proved the equivalence of several realizations (Direct Forms and State-Space).

1 Introduction

Most embedded systems, from planes to MP3 players, rely on numerical signal processing filters. Such a filter takes as inputs an infinite sequence of values, such as measurements of sensors, and returns an infinite sequence of values, such as a sound to be played or a value to control a nuclear power plant (see also Sect. 2.2). Applications of digital filters are therefore numerous, from trivial to life-critical, and formal methods have already been applied to such systems.

Digital filters have previously been formalized in HOL [1]: Akbarpour and Tahar define filters and somehow define the error filter as done in Sect. 5.1. But they do not bound the output error, while we do it using the Worst-Case Peak Gain (WCPG) Theorem of Sect. 5.2. We indeed benefit from the most recent advances for filter error bounding [2]. Another recent work in HOL by Siddique *et al.* focuses on the frequency-domain analysis and the z -Transform [3]. They consider causal filters and difference equations with definitions similar to ours. But their analysis is complementary to ours as we choose a time-domain analysis

D. Gallois-Wong—This work is supported by a grant from the “Fondation CFM pour la Recherche”.

in order to next focus on finite-precision implementations. Akbarpour *et al.* previously compared floating-point and fixed-point implementations of digital filters to ensure their similar behavior [4]. But they require many hypotheses, including the absence of overflow in the computations. Park *et al.* aim at specifying filters and then proving their correctness [5], assuming that all the computations are exact. They then relax this assumption, but bound the floating-point error for one iteration step only, while the difficult part is the propagation of these errors [6]. Last, Feret developed a specific abstract domain for digital filters [7]. Finally, references in digital signal processing can be found in Sect. 2.

This paper presents a formalization of digital filters in the Coq proof assistant [8,9].¹ We rely on the standard library for defining reals and on the Coquelicot library [10] for real analysis and for the Limited Principle of Omniscience (LPO) which is derived from the axiomatization of reals. In addition, we use two axioms. Firstly, Functional Extensionality states that two functions sharing the same value on every input are equal. Secondly, Proof Irrelevance means that two proofs of the same property are equal. These axioms are often used to handle functions and proof objects in a more natural way and are considered safe.

The goals of this paper are a formal definition of a digital filter, its various algorithms and their equivalences, and some formal definitions and proofs about the error analysis to be used for fixed-point and floating-point implementations. This paper is organized as follows. Section 2 gives some background on signal processing and numerical filters. Section 3 presents our formalization choices. Section 4 defines some of the many (mathematically) equivalent algorithms, called realizations, that are used to describe a filter in signal processing. We also prove their equivalences, so that results established for one of them also hold for the others. Section 5 proves the error filter decomposition and the Worst-Case Peak-Gain theorem. Together, these two results allow to bound the final impact of adding a bounded error term at each computation step, having taken into account the propagation of these errors as each of them affects every future step. We were able to formalize this using only real numbers as there is no need to know where the error terms come from, but of course this is intended to be applied to rounding errors in finite-precision arithmetic. Finally, Sect. 6 concludes and gives some perspectives.

Notation: Throughout the article, matrices are in uppercase boldface (e.g. \mathbf{A}), vectors are in lowercase boldface (e.g. \mathbf{a}), scalars are in lowercase (e.g. a). The matrix \mathbf{I}_n is the identity matrix of size n .

2 Digital Filters

2.1 Signals and Operations

A discrete-time signal x is an ordered sequence of numbers denoted $x(k)$, where k is an integer. In a practical setting, such sequences often arise from periodic sampling of continuous time signals $x_c(t)$ where t represents time.

¹ Available at www.lri.fr/~gallois/code/coq-digital-filters-CICM18.tgz.

The signal x can be defined for any integer k or for some finite contiguous set of integers. We here restrict k to be in \mathbb{N} , or more precisely k to be in \mathbb{Z} , but with $x(k) = 0$ for $k < 0$. A signal can be real ($x(k) \in \mathbb{R}$) or vector ($\mathbf{x}(k) \in \mathbb{R}^{p \times 1}$).

The simplest signal is probably the *impulse* signal (also called *Dirac* signal), denoted δ and defined as $\delta(k) = \begin{cases} 1 & \text{if } k = 0 \\ 0 & \text{elsewhere.} \end{cases}$

This signal is central in linear signal processing theory since any signal can be expressed as an infinite sum of impulse signals (see Sect. 2.3).

Three elementary operations on signals can be defined:

- Addition: The sum of two signals x_1 and x_2 is their term-by-term sum, i.e. $y = x_1 + x_2$ means $\forall k, y(k) = x_1(k) + x_2(k)$.
- Scaling: $y = \alpha x$ is the sequence x scaled by $\alpha \in \mathbb{R}$, i.e. $\forall k, y(k) = \alpha x(k)$.
- Time shifting or delay: Let y be the sequence x shifted in time by an integer $K \geq 0$, then $\forall k, y(k) = \begin{cases} x(k - K) & \text{if } k \geq K \\ 0 & \text{elsewhere.} \end{cases}$

2.2 Linear Time Invariant Filters

A filter \mathcal{H} is a mathematical transformation that maps an input signal u into an output signal $y = \mathcal{H}\{u\}$, as shown in Fig. 1. At each time k , the filter produces an output $y(k)$. But, contrary to usual mathematical functions, the output $y(k)$ depends not only on the input $u(k)$, but also on the internal state of the filter (i.e. on the initial condition of the filter and the previous inputs).

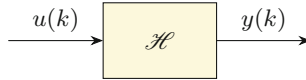


Fig. 1. A discrete-time filter.

A linear time invariant (LTI) filter satisfies the two following properties:

- Linearity: The filter is linear with respect to its input, i.e.

$$\mathcal{H}\{\alpha u_1 + \beta u_2\} = \alpha \mathcal{H}\{u_1\} + \beta \mathcal{H}\{u_2\} \quad (1)$$

- Shift invariance: if the input of the filter is delayed by $K \geq 0$ samples, then the output is also delayed by K samples, i.e. if $\forall k, u_1(k) = u_2(k - K)$, then

$$\forall k, \quad \mathcal{H}\{u_1\}(k) = \begin{cases} \mathcal{H}\{u_2\}(k - K) & \text{if } k \geq K \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

The filter can have as inputs and outputs either scalars (Single-Input Single-Output filter, aka SISO filter), or vectors (Multiple-Input Multiple-Output filter, aka MIMO filter). Due to the linearity of LTI filters, a q -input p -output MIMO filter can be seen as the assembly of $p \times q$ SISO filters, where the ij^{th} element is the filter that captures the effect of the j^{th} input on the i^{th} output.

LTI filters are compositions of the three elementary operations on signals (addition/subtraction, multiplication by constant and delay, the last one being classically denoted z^{-1} as in the z -transform [11, 12]). Data-flow graphs using these operations as blocks and signal as streams are widely used in signal and control theory to describe the different realizations, as shown in Figs. 3a, b, and 4a.

2.3 Impulse Response

The impulse response of a SISO filter \mathcal{H} , denoted h , is the answer (output) of the filter to an impulse input δ (i.e. $h = \mathcal{H}\{\delta\}$).

Since any input u can be written as an infinite sum of weighted shifted impulses $u(k) = \sum_{l \in \mathbb{Z}} u(l)\delta(k-l)$, then, by linearity of the considered filter, the output y of u through \mathcal{H} can be obtained by

$$y(k) = \mathcal{H}\{u\}(k) = \sum_{l \in \mathbb{Z}} u(l)h(k-l) \quad (3)$$

The output y is obtained as a convolution of signals u and h . For that reason, the impulse response of a filter fully defines it. From the impulse response characterization, the LTI filters can be divided in two types: the Finite Impulse Response (FIR) filters, where h is null above a certain time, and the Infinite Impulse Response (IIR) filters, where h has infinite support [11].

2.4 Constant-Coefficient Difference Equation

An important subclass of LTI filters consists of those for which a n -order constant-coefficient difference equation exists between inputs and outputs, i.e. the last n inputs and outputs are linked by $\forall k, \sum_{i=0}^n a_i y(k-i) = \sum_{i=0}^n b_i u(k-i)$, where the $\{a_i\}_{0 \leq i \leq n}$ and $\{b_i\}_{0 \leq i \leq n}$ are constant coefficients.

We also assume that $a_0 = 1$, so that the equation is rearranged as

$$y(k) = \sum_{i=0}^n b_i u(k-i) - \sum_{i=1}^n a_i y(k-i). \quad (4)$$

This relationship describes an IIR filter as soon as the a_i 's coefficients are not all null, otherwise it describes an FIR filter. The value n is said to be the *order* of the filter. These coefficients are also the coefficients of the *transfer function* of the filter: this mathematical object describes its input-output relationship in frequency domain (whereas (4) describes it in time domain). It has also been formalized in Coq, but with few properties, and thus not presented here.

3 Formalization

3.1 Signals and Filters

The first components required to work with digital filters are signals, presented in Sect. 2.1. For now, we consider real signals (sequences of real numbers). We define them as functions from \mathbb{Z} to \mathbb{R} that take the value 0 for every $k < 0$.

Definition `causal` $(x : \mathbb{Z} \rightarrow \mathbb{R}) := (\text{forall } k : \mathbb{Z}, (k < 0)\%Z \rightarrow x\ k = 0\%R)$.
Record `signal` := {`signal_val` :> $\mathbb{Z} \rightarrow \mathbb{R}$; `signal_prop` : `causal signal_val`}.

Axioms `FunctionalExtensionality` and `ProofIrrelevance`, discussed in the introduction, ensure that signals are fully characterized by their values for $k \geq 0$.

This definition of signals makes them easy to build recursively. Notably, order- n recursion (when $x(k)$ is built from $x(k-1), x(k-2), \dots, x(k-n)$) is very common when working with filters: for example, (4) is an order- n recursive relation on y . For signals, the negative values are known (since they are zero), so we may use an order- n recursion. See Sect. 3.2 for more on recursive constructions.

We could have defined signals as functions from \mathbb{N} to \mathbb{R} . Ironically, working with \mathbb{N} instead of \mathbb{Z} would bring back problems of initialization and saturation. We have considered and discarded several solutions to work with \mathbb{N} : they either complicated proofs by requiring to handle additional non-trivial cases, or used alternative notations that made theorems more difficult to read and compare to their usual signal processing formulation. For example, in order to understand some statements, one is required to know that $0-2$ is equal to 0 when working with \mathbb{N} in Coq. Readability is very important to us as we want to spread formal methods among the digital processing community. Ultimately, we felt that the readability and the more intuitive subtraction offered by \mathbb{Z} were worth adapting a few libraries and recursive constructions.

We define the three elementary operations on signals described in Sect. 2.1, namely addition, scaling by a real and time shift. We prove that the results are still signals (they take the value 0 for $k < 0$). The only interesting point is in the time shift by an integer K , which usually requires that $K \geq 0$: we arbitrarily choose that it returns the null signal when $K < 0$, so that the function is total.

Once we have signals, filters are simply defined as functions from signals to signals: **Definition** `filter` := `signal -> signal`. From the three elementary operations on signals defined above (addition, scaling by a real and time shifting), we define compatibility of a filter with addition, compatibility with scalar multiplication and shift invariance. Finally, we define `LTI_filter` : `filter -> Prop` (linear time invariant filter) as the conjunction of these properties.

As explained in Sect. 2.2, a filter can be SISO (Single-Input Single-Output) when it handles real signals as in the definition above, or MIMO (Multiple-Input Multiple-Output) with vector signals. A vector signal is defined as a function from \mathbb{Z} to \mathbb{R}^p that returns the null vector for every $k < 0$. A MIMO filter is then:

Definition `MIMO_filter` $\{N_in\ N_out : \mathbb{Z}\} :=$
`@vect_signal N_in -> @vect_signal N_out`.

As in textbooks and for the sake of readability, most of our theorems are dedicated to SISO filters. Nevertheless, we define the State-Space characterization of a filter, presented in Sect. 4.3, for MIMO filters, as we explicitly need this to study error propagation in Sect. 5.1.

3.2 Recursion over \mathbb{Z}

To define a filter given by its constant-coefficient difference Eq. (4), we need to build the output signal y by order- n recursion. In practice, we find it easier to use strong recursion which is more general. The standard library has a lemma `Coq.ZArith.Wf_Z.Z1t_0_rec` that shows that a construction by strong recursion is possible for positive indexes, whereas what we want is the actual function resulting from this construction. Therefore, we define `Z_strong_rec` which builds this function. It takes an initialization function `f_lt_0` that will only be used for $k < 0$, and a step function `f_rec` that computes the image of k for $k \geq 0$, depending on the image of j for any $j < k$. If we note $f|_{<k}$ the restriction of f to $(-\infty, k)$, `Z_strong_rec` builds the function $f : \mathbb{Z} \rightarrow T$ such that

$$\begin{cases} \forall k < 0, f(k) = \mathbf{f_lt_0}(k) \\ \forall k \geq 0, f(k) = \mathbf{f_rec}(k)(f|_{<k}) \end{cases} \quad (5)$$

Definition `Z_strong_rec` (`f_lt_0` : $\mathbb{Z} \rightarrow T$)
(`f_rec` : $\mathbb{Z} \rightarrow (\mathbb{Z} \rightarrow T) \rightarrow T$) (`k` : \mathbb{Z}) : $T := (\dots)$

But, as we want to work with total functions, the second argument of `f_rec` is total. When we write $f|_{<k}$, we actually mean that it is equal to previously computed values of f for $j < k$, and to default values for $j \geq k$. As `f_rec` is just an argument of `Z_strong_rec`, it could evaluate $f|_{<k}$ for $j \geq k$, which would make no sense from a recursion perspective. To have a proper recursive construction, the argument `f_rec` needs to verify the property `f_rec_well_formed`, which means that when we call `f_rec` (`k` : \mathbb{Z}) (`g` : $\mathbb{Z} \rightarrow T$) with $k \geq 0$, the result does not depend on the values of `g` for $j \geq k$. Lemmas `Z_strong_rec_lt_0` and `Z_strong_rec_ge_0` express (5), the second one needing the hypothesis that the argument `f_rec` is well-formed. They serve as an interface so that outside of their own proofs, the 13-line definition of `Z_strong_rec` is never expanded.

Definition `f_rec_well_formed` (`f_rec` : $\mathbb{Z} \rightarrow (\mathbb{Z} \rightarrow T) \rightarrow T$) : **Prop** :=
`forall` (`k` : \mathbb{Z}) (`g1 g2` : $\mathbb{Z} \rightarrow T$), (`k` \geq 0)% $\mathbb{Z} \rightarrow$
(`forall` `j` : \mathbb{Z} , (`j` $<$ `k`)% $\mathbb{Z} \rightarrow$ `g1 j` = `g2 j`) \rightarrow `f_rec k g1` = `f_rec k g2`.

Lemma `Z_strong_rec_ge_0 f_lt_0 f_rec k` :
(`k` \geq 0)% $\mathbb{Z} \rightarrow$ `f_rec_well_formed f_rec` \rightarrow
`Z_strong_rec f_lt_0 f_rec k` = `f_rec k (Z_strong_rec f_lt_0 f_rec)`.

For example, consider a signal x defined by the recursive relation $x(k) = x(k-1) + x(k-2)$. We define the function representing this relation:
`f_rec` := `fun` (`k` : \mathbb{Z}) (`g` : $\mathbb{Z} \rightarrow \mathbb{R}$) \Rightarrow (`g`(`k-1`)% \mathbb{Z} + `g`(`k-2`)% \mathbb{Z})% \mathbb{R} , and we can prove that it is well-formed. As a signal should be zero for $k < 0$, the initialization function is `f_lt_0` := `fun` (`k` : \mathbb{Z}) \Rightarrow 0% \mathbb{R} . Then, (`Z_strong_rec f_lt_0 f_rec`) : $\mathbb{Z} \rightarrow \mathbb{R}$ is the function that corresponds to our signal x .

3.3 Adapting Existing Libraries to Relative Indexes

We build upon the Coquelicot library to obtain sums of consecutive terms of a sequence and matrices that are compatible with relative indexes. Matrices with relative indexes look strange, but there is no problem in practice, as relevant indexes for a given matrix of size $\mathbf{h} \times \mathbf{w}$ are only subsets in any case: $1 \leq i \leq \mathbf{h}$ and $1 \leq j \leq \mathbf{w}$. We were careful to handle matrices without relying on the particular Coquelicot definition, so that we can easily switch to other existing libraries. In particular, we may use the Mathcomp library [13] and rely on its linear algebra theorems to evaluate the WCPG defined in Sect. 5.2.

4 Filter Realizations

To implement a filter, one needs an explicit algorithm, which produces at each k an output $y(k)$ from the input $u(k)$ and its internal state. In the literature, a lot of algorithms exist to implement a linear filter [11]: Direct Forms, State-Space, Second-Order Sections, cascade or parallel decomposition, Lattice Wave Digital filters [14], δ - or ρ -operator based filters [15, 16], etc. Each of them presents some advantages with respect to the number of coefficients, the software or hardware adequacy, the finite-precision behavior, etc. and the choice of the realization is itself a research domain [17].

We present here three classical realizations: Direct Form I (that comes from the constant-coefficients difference Eq. (4)), Direct Form II (that uses the same coefficients in another algorithm) and State-Space. In practice, these realizations are explained by a data-flow graph, that describes how data (signals) are processed by a filter in terms of inputs and outputs.

4.1 Direct Form I

Direct Form I (DFI) comes directly from the constant-coefficient difference equation presented in Sect. 2.4: $y(k) = \sum_{i=0}^n b_i u(k-i) - \sum_{i=1}^n a_i y(k-i)$. It depends on the $2n + 1$ transfer-function coefficients $a_1, a_2, \dots, a_n, b_0, b_1, \dots, b_n$.

<pre> foreach k do $y(k) \leftarrow \sum_{i=0}^n b_i u(k-i) - \sum_{i=1}^n a_i y(k-i)$ end </pre> <p style="text-align: center;">(a) Direct Form I</p>	<pre> foreach k do $e(k) \leftarrow u(k) - \sum_{i=1}^n a_i e(k-i)$ $y(k) \leftarrow \sum_{i=1}^n b_i e(k-i)$ end </pre> <p style="text-align: center;">(b) Direct Form II</p>
---	--

Fig. 2. Direct Form I and II algorithms.

The corresponding algorithm is presented in Fig. 2a and the data flow graph in Fig. 3a. The real program is slightly more complex: the n previous values of u and y are stored in memory (the z^{-1} squares in the data flow graph in Fig. 3a represent *delay*, each one is a memory element).

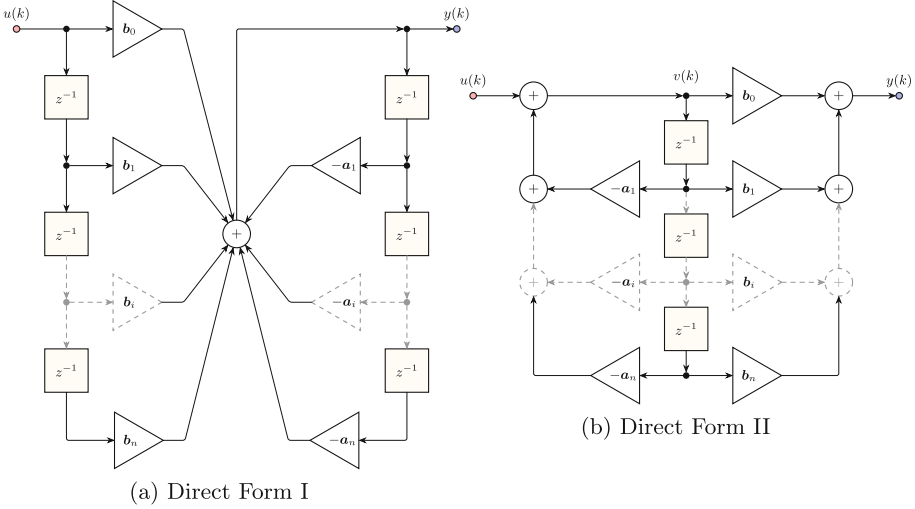


Fig. 3. Direct Form I and II data-flow graphs.

In Coq, we define a type `TFCoeffs` for the transfer function coefficients. They are given as an order n and two sequences $a, b : \mathbb{Z} \rightarrow \mathbb{R}$. In practice, n should be positive, and we will only use the values $a(i)$ for $1 \leq i \leq n$ and the values $b(i)$ for $0 \leq i \leq n$. So far, we have not needed to enforce this in the definition.

Record `TFCoeffs := {TFC_n : Z ; TFC_a : Z -> R ; TFC_b : Z -> R}`.

The main component needed for our recursive construction of a signal x is a function, noted `f_rec` in our definition of strong recursion (Sect. 3.2), that builds $x(k)$ for $k \geq 0$ given all previous values of x . `(DFI_f_rec tfc u) : Z -> (Z -> R) -> R` is such a function: it expresses the recursive relation characterizing the output signal for input u of the filter defined by Direct Form I with the coefficients `tfc`.

From this, we define the function that builds a filter from its transfer function coefficients using Direct Form I, called `filter_from_TFC` because Direct Form I is the canonical way to build a filter from these coefficients.

Definition `DFI_f_rec (tfc : TFCoeffs) (u : signal) :=`
`(fun (n : Z) (y_before_n : Z -> R) =>`
`(sum 0 (TFC_n tfc) (fun i => (TFC_b tfc i) * u (n-i)%Z)%R`
`- (sum 1 (TFC_n tfc) (fun i => (TFC_a tfc i) * y_before_n (n-i)%Z)%R)).`

Definition `filter_from_TFC (tfc : TFCoeffs) : filter :=`
`fun (u : signal) => build_signal_rec (DFI_f_rec tfc u).`

Finally, we prove that a filter built this way is LTI. This proof presents no real difficulty once we have adequate lemmas about sum of consecutive value of a sequence and recursive building of a signal.

4.2 Direct Form II

Direct Form II is quite similar to Direct Form I. It uses the same coefficients, those of the transfer function. The main difference is that it only requires n delays (instead of $2n$), so it is more efficient to implement. It can be described by the data flow graph of Fig. 3b, or by the algorithm of Fig. 2b.

Where Direct Form I builds the output y from previous values of both y and the input u , Direct Form II builds an intermediary signal e from previous values of itself and only the current value of u , then it builds y from previous values of e . The Coq definitions reflect this. As the construction of e is recursive, we define `DFII_u2e_f_rec tfc u`, which is the main function allowing to build e given the transfer function coefficients and an input signal.

Combining the constructions of the intermediary signal e and of the output signal y , we define `filter_by_DFII` which builds a filter from transfer function coefficients using Direct Form II.

```

Definition DFII_u2e_f_rec (tfc : TFCoeffs) (u : signal) :=
  fun (n : Z) (e_before_n : Z -> R) =>
    u n - sum 1 (TFC_n tfc) (fun i => (TFC_a tfc i) * e_before_n (n-i)%Z).
Definition DFII_e2y (tfc : TFCoeffs) (e : signal) := Build_signal
  (fun n => sum 0 (TFC_n tfc) (fun i => (TFC_b tfc i) * e (n-i)%Z)%R)
  (DFII_e2y_prop tfc e).
Definition filter_by_DFII (tfc : TFCoeffs) : filter :=
  fun (u : signal) => DFII_e2y tfc (build_signal_rec (DFII_u2e_f_rec tfc u)).

```

Finally, as Direct Form I and Direct Form II use the same coefficients, which are also the coefficients associated to the transfer function, implementing either of these algorithms with the same set of coefficients should produce the same filter, meaning the same output even if the algorithms are different.

```

Theorem DFI_DFII_same_filter (tfc : TFCoeffs) :
  filter_from_TFC tfc = filter_by_DFII tfc.

```

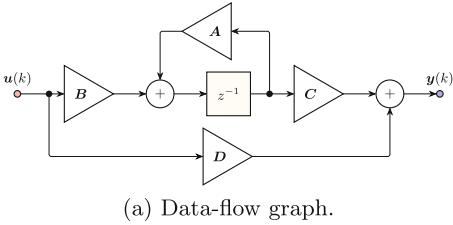
Since a filter built using `filter_from_TFC` (which is by definition built using Direct Form I) is LTI, so is a filter built using Direct Form II.

4.3 State-Space

We now consider MIMO filters, which are needed for the error analysis in Sect. 5.1. For a q -input p -output MIMO filter \mathcal{H} , the State-Space representation is described by four matrices (A, B, C, D) . The corresponding algorithm is:

$$\begin{cases} \mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \\ \mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) + \mathbf{D}\mathbf{u}(k) \end{cases} \quad (6)$$

where $\mathbf{x}(k) \in \mathbb{R}^{n \times 1}$, $\mathbf{u}(k) \in \mathbb{R}^{q \times 1}$ and $\mathbf{y}(k) \in \mathbb{R}^{p \times 1}$ are the state vector, input vector and output vector, respectively. The matrices $\mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times q}$, $\mathbf{C} \in \mathbb{R}^{p \times n}$ and $\mathbf{D} \in \mathbb{R}^{p \times q}$ characterize the filter \mathcal{H} . Figure 4a exhibits its data-flow graph, and Algorithm 4b its algorithm (scalar rewriting of (6)).



```

foreach k do
  for i ∈ {1, ..., n} do
    |  $x_i(k+1) \leftarrow \sum_{j=1}^n A_{ij} x_j(k) + \sum_{j=1}^q B_{ij} u_j(k)$ 
  end
  for i ∈ {1, ..., p} do
    |  $y_i(k) \leftarrow \sum_{j=1}^n C_{ij} x_j(k) + \sum_{j=1}^q D_{ij} u_j(k)$ 
  end
end

```

(b) Algorithm

Fig. 4. State-Space data-flow graph and algorithm.

In Coq, we define a State-Space as a record containing the size of the state vector and the four matrices. Here `@mtx R_Ring h w` is the type of matrices of size $h \times w$ with coefficients in the ring \mathbb{R} .

Context {N_in N_out : Z}.

Record StateSpace := {StSp_n : Z ; (* size of the state vector *)
 StSp_A : @mtx R_Ring StSp_n StSp_n ;
 StSp_B : @mtx R_Ring StSp_n N_in ;
 StSp_C : @mtx R_Ring N_out StSp_n ;
 StSp_D : @mtx R_Ring N_out N_in}.

For a given State-Space and a given input vector signal u , we first define the vector signal of state vectors x by recursion, then the input vector signal y from u and x , following closely the relationship in (6). We obtain a function that builds the MIMO filter corresponding to a State-Space. We also define a SISO State-Space as a State-Space where the context variables N_{in} and N_{out} are both 1. Then we define `filter_from_StSp` that associates a SISO filter to such a State-Space. We also prove that a filter built from a State-Space is always LTI.

The impulse response h (introduced in Sect. 2.3) of a filter defined by a State-Space is proved to be computable from the matrices of the State-Space with:

$$h(k) = \begin{cases} \mathbf{D} & \text{if } k = 0 \\ \mathbf{C}\mathbf{A}^{k-1}\mathbf{B} & \text{if } k > 0 \end{cases} \quad (7)$$

Moreover, in SISO, a State-Space can also be built from the transfer function coefficients. Below is one of many ways to build such a State-Space, requiring

$(n+1)^2$ coefficients for the four matrices combined (note that \mathbf{C} is a single-row matrix, \mathbf{B} single-column, and \mathbf{D} single-row single-column):

$$\begin{aligned} \mathbf{A} &= \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & 1 \\ -a_n & \dots & \dots & -a_2 & -a_1 \end{pmatrix}, & \mathbf{B} &= \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ 1 \end{pmatrix} \\ \mathbf{C} &= (b_n - a_n b_0 \ \dots \ b_1 - a_1 b_0), & \mathbf{D} &= (b_0) \end{aligned} \quad (8)$$

Notation `n := (TFC_n tfc)`.

Definition `SISO_StSp_from_TFC : SISO_StateSpace := @Build_StateSpace 1 1`

```
(*StSp_n*) n
(*StSp_A*) (make_mtx n n (fun i j =>
  if Z_eq_dec i n then (- (TFC_a tfc (n+1-j)))
  else if Z_eq_dec (i+1) j then 1 else 0))
(*StSp_B*) (make_mtx n 1 (fun i j => if Z_eq_dec i n then 1 else 0))
(*StSp_C*) (make_mtx 1 n (fun i j =>
  TFC_b tfc (n+1-j) - TFC_a tfc (n+1-j) * TFC_b tfc 0))
(*StSp_D*) (mtx_of_K (TFC_b tfc 0)).
```

Theorem `StSp_TFC_same_filter (tfc : TFCoeffs) : (TFC_n tfc >= 0)%Z -> filter_from_StSp (SISO_StSp_from_TFC tfc) = filter_from_TFC tfc.`

We define a Coq function that associates a (Single Input Single Output) State-Space to transfer function coefficients, by simply constructing the matrices as in (8). More importantly, we prove that the filter built from such a State-Space is the same as the filter obtained directly from the transfer function coefficients.

As we have seen previously, there are at least two ways to build a filter directly from these coefficients: Direct Form I and Direct Form II. Although Direct Form I is the canonical one, it is much easier to use Direct Form II in this proof. The main step is to prove that for any k , the state vector $\mathbf{x}(k)$ contains $e(k-n)$ as its first coefficient, $e(k-(n-1))$ as its second one, etc. up to $e(k-1)$ as its n -th coefficient, where e is the auxiliary signal that appears in Direct Form II (Fig. 2b).

This is done by strong induction, with trivial initialization for $k < 0$ since everything is zero. For $k \geq 0$, the *ones* just above the diagonal in \mathbf{A} and the *zeros* in \mathbf{B} mean that the coefficients 1 to $n-1$ of $\mathbf{x}(k)$ are the coefficients 2 to n of $\mathbf{x}(k-1)$, so the only point left to prove is that $\mathbf{x}(k)_n = e(k-1)$. This is ensured by the last line of \mathbf{A} and the last coefficient of \mathbf{B} , which make the matrix operations in (6) unfold into exactly the computation of $e(k)$ in Fig. 2b.

Note that the matrices $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ are not uniquely defined for a given filter: distinct State-Spaces describe distinct algorithms, but they may build the same filter. So it is possible to search for the *optimal* State-Space when considering the effect of the finite precision degradations [18].

5 Error Analysis Tools

We now aim at giving tools for a future full error analysis of implemented filters. Using finite precision arithmetic (floating- or fixed-point arithmetic), some arithmetic operations may introduce errors (mainly because the number of bits to represent the values is finite, and may be not enough to represent the exact result). These errors are then to be taken into account in the following computations as they may accumulate over time. To bound this accumulation, Sect. 5.1 shows that these errors may be extracted from the main computations. Their values are then modified over time by to another filter. To bound this error, we now only need to bound the maximal value of a filter (this may also help us prevent overflows). This is done in Sect. 5.2 by the Worst-Case Peak Gain Theorem.

5.1 Error Filter

Let us focus on the errors due to finite precision arithmetic, without more details on this arithmetic. The corresponding quantization can be modeled as an extra term, called *roundoff error*. We consider a State-Space $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$, as this is the most general of the realizations that we have presented. Indeed, we have proven that from any other of these realizations, we can build a State-Space that defines the same algorithm and thus the same filter.

At each step, the evaluation of the states and outputs (see Algorithm 4b) is composed of sum-of-products (SoP), one per state and output. Since they are not exact, each may produce an error, compared to the exact SoP. So (6) becomes:

$$\begin{aligned} \mathbf{x}^*(k+1) &\leftarrow \mathbf{A}\mathbf{x}^*(k) + \mathbf{B}\mathbf{u}(k) + \boldsymbol{\varepsilon}_x(k) \\ \mathbf{y}^*(k) &\leftarrow \mathbf{C}\mathbf{x}^*(k) + \mathbf{D}\mathbf{u}(k) + \boldsymbol{\varepsilon}_y(k) \end{aligned} \quad (9)$$

where $\mathbf{x}^*(k)$ and $\mathbf{y}^*(k)$ are the computed values for the state vector and output vector, and $\boldsymbol{\varepsilon}_x(k)$ and $\boldsymbol{\varepsilon}_y(k)$ are the vectors of roundoff errors due to the sum-of-products evaluation. Denote $\boldsymbol{\varepsilon}(k)$ the column vector that aggregates those errors: $\boldsymbol{\varepsilon}(k) = \begin{pmatrix} \boldsymbol{\varepsilon}_x(k) \\ \boldsymbol{\varepsilon}_y(k) \end{pmatrix} \in \mathbb{R}^{n+p}$, where n is the size of $\mathbf{x}(k)$ and p is the size of $\mathbf{y}(k)$.

In order to capture the effects of the finite precision implementation we must take into account the propagation of the roundoff errors through the data-flow. The output error $\boldsymbol{\Delta}\mathbf{y}(k)$ is defined as the difference between the outputs of the implemented and the exact filters: $\boldsymbol{\Delta}\mathbf{y}(k) = \mathbf{y}^*(k) - \mathbf{y}(k)$. Subtracting (6) to (9), it follows that $\boldsymbol{\Delta}\mathbf{y}(k)$ can be seen as the output of the vector signal of errors $\boldsymbol{\varepsilon}(k)$ through the filter \mathcal{H}_ε defined by the State-Space $(\mathbf{A}, \mathbf{B}_\varepsilon, \mathbf{C}, \mathbf{D}_\varepsilon)$, where $\mathbf{B}_\varepsilon = (\mathbf{I}_n \mathbf{0})$ and $\mathbf{D}_\varepsilon = (\mathbf{0} \mathbf{I}_p)$. Equivalently (thanks to the linearity of the considered filter), the implemented filter can be seen as the sum of the exact filter \mathcal{H} and the error filter \mathcal{H}_ε with $\boldsymbol{\varepsilon}(k)$ as input, as shown on Fig. 5.

The filter \mathcal{H}_ε expresses how the errors propagate through the filter, and knowing some properties on the roundoff errors $\boldsymbol{\varepsilon}(k)$ will lead to properties

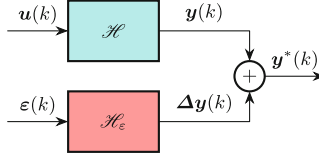


Fig. 5. Equivalent filter, with errors separated.

on the output errors $\Delta \mathbf{y}(k)$, and hence on the accuracy of the implemented algorithm.

In Coq, we assume we are given a State-Space `stsp`, a vector input signal `u` and vector error signals `err_x` and `err_y`. We also assume the obviously reasonable hypotheses that the implicit size of output vectors for `stsp` and the order of `stsp` are non-negative (`(N_out >= 0)%Z` and `(StSp_n stsp >= 0)%Z`). We define `x'` (recursively) then `y'` (using `x'`) the vector signals corresponding to \mathbf{x}^* and \mathbf{y}^* in (9). We define `B_err` and `D_err` the matrices corresponding to B_ε and D_ε and `errors` as the vertical concatenation of `err_x` and `err_y`. We prove that `y'` is indeed the sum of the outputs of the exact filter described by `stsp` for input `u`, and of the error filter described by a new State-Space `stsp_err` for input `errors`.

5.2 Worst-Case Peak Gain Theorem

The Worst-Case Peak Gain Theorem provides the maximum possible value for the outputs of a state-space filter. Applied on the filter \mathcal{H}_ε , it gives a bound on the output error due to the finite precision computations.

We consider \mathcal{H} a SISO LTI filter. Its Worst-Case Peak Gain (WCPG) [19, 20], noted $\langle\langle \mathcal{H} \rangle\rangle$, is an element of \mathbb{R} defined as:

$$\langle\langle \mathcal{H} \rangle\rangle = \sum_{k=0}^{\infty} |h(k)| \quad (10)$$

where h is the impulse response of \mathcal{H} (see Sect. 2.3). If an input signal u is bounded by M ($\forall k, |u(k)| \leq M$), then the corresponding output signal y is bounded by the value $\langle\langle \mathcal{H} \rangle\rangle M \in \mathbb{R}$. We can also write this as an inequality over \mathbb{R} :

$$\forall u, \sup_{k \in \mathbb{Z}} (\mathcal{H}\{u\}(k)) \leq \langle\langle \mathcal{H} \rangle\rangle \sup_{k \in \mathbb{Z}} (u(k)) \quad (11)$$

Moreover, the WCPG is optimal: it is the smallest number that verifies (11). In Coq, we define the WCPG exactly as in (10) using `Lim_seq` from Coquelicot [10].

Definition `dirac` : `signal` := (... (* 0 -> 1, k <> 0 -> 0 *)

Definition `impulse_response` (H : `filter`) : `signal` := H `dirac`.

Definition `sum_abs_IR` (H : `filter`) (n : `nat`) :=

`sum 0 (Z.of_nat n) (fun k : Z => Rabs ((impulse_response H) k)).`

Definition `wcpg` (H : `filter`) : `Rbar` := `Lim_seq` (`sum_abs_IR` H).

To prove (11), we rely on the fact that the image by a LTI filter of a signal u can be obtained as the convolution of u and the impulse response of the filter: $\mathcal{H}\{u\}(k) = \sum_{l \in \mathbb{Z}} u(l)h(k-l)$ (3). We also prove the optimality of the WCPG, with two theorems depending on whether the WCPG is finite. For both of them, from the definition of $\langle\langle \mathcal{H} \rangle\rangle$ as an infinite sum, we can get an index N such that $\sum_{k=0}^N |h(k)|$ is sufficiently close to $\langle\langle \mathcal{H} \rangle\rangle$. Then, we define a signal u such that for $0 \leq k \leq N$, $u(k)$ is in $\{1, -1\}$ and has the same sign as $h(N-k)$. We obtain $\mathcal{H}\{u\}(N) = \sum_{l \in \mathbb{Z}} u(l)h(N-l) = \sum_{0 \leq l \leq N} |h(l)|$.

Theorem `wcpg_theorem` (H : filter) (u : signal) (M : R) :

LTI_filter H \rightarrow (forall k : Z, Rabs (u k) <= M) \rightarrow
 (forall k : Z, Rbar_le (Rabs (H u k)) (Rbar_mult wcpg M)).

Theorem `wcpg_optimal` (H : filter) : LTI_filter H \rightarrow is_finite (wcpg H) \rightarrow

forall epsilon : R, epsilon > 0 \rightarrow exists (u : signal) (N : Z),
 ((forall k : Z, Rabs (u k) <= 1) \wedge (H u) N > wcpg H - epsilon).

Theorem `infinite_wcpg_optimal` (H : filter) : LTI_filter H \rightarrow

wcpg H = p_infty \rightarrow forall M : R, exists (u : signal) (N : Z),
 ((forall k : Z, Rabs (u k) <= 1) \wedge H u N > M).

Another important property of a filter is that, for any bounded input ($\exists M, \forall k, |u(k)| \leq M$), the output is bounded as well (by a number M'). This property is known as the Bounded Input Bounded Output (BIBO) stability [21], and is shared by most filters that are of practical interest. We easily defined the bounded property for signals, and the BIBO property for filters. An important theorem is that the WCPG of a LTI filter verifying this property is always finite.

Theorem `BIBO_wcpg_finite` (H : filter) :

LTI_filter H \rightarrow BIBO H \rightarrow is_finite (wcpg H).

The principle is to prove the contrapositive: if the WCPG is infinite then we will build an input u bounded by 1 such that the output is unbounded. As seen in the proof of optimality of the WCPG, for any bound M , we can construct an input u bounded by 1 and an index N such that $|\mathcal{H}\{u\}(N)| > M$. Two facts allow us to get from this construction, where u can be chosen after M , to a single unbounded signal. Firstly, the property $|\mathcal{H}\{u\}(N)| > M$ only depends on values of u for $0 \leq k \leq N$. Secondly, we are able to adapt this construction to leave an arbitrary number of preset input values unchanged: for any $M \in \mathbb{R}$, $K \in \mathbb{Z}$ and imposed values $u(0), \dots, u(K)$ that are bounded by 1, we can extend them into a signal u still bounded by 1 such that $|\mathcal{H}\{u\}(N)| > M$ for some index N . Iterating this construction, we build a signal such that its image has at least a term exceeding 0, a term exceeding 1, and so on at least a term exceeding M for any $M \in \mathbb{Z}$. These repeated constructions are tricky to handle in Coq, and we used the $\{\dots | \dots\}$ strong existential construction rather than the existential quantifier. We relied on the Limited Principle of Omniscience (LPO) to be able to construct indices N as described above.

6 Conclusions and Perspectives

We have formalized digital filters in the Coq proof assistant and provided several realizations. For one of these realizations, namely the State-Space, we have proved theorems about error analysis, that will be useful when finite-precision arithmetic will come into play. A surprising difficulty was the induction on \mathbb{Z} as described in Sect. 3.2: it was to decide that the standard library results were not exactly what we needed and to state the corresponding theorems and total functions.

A part of the digital processing results we did not focus on is the transfer function. We defined it but we have not yet linked it to the rest of the development. The z -Transform has been formalized in HOL [3, 12] and it will be interesting to see if similar theorems may be proved with our Coq formalization.

The Worst-Case Peak Gain Theorem has been proved for the SISO filters, including State-Space. The general formula $\langle\langle \mathcal{H} \rangle\rangle = |\mathbf{D}| + \sum_{k=0}^{\infty} \left| \mathbf{C} \mathbf{A}^k \mathbf{B} \right|$ has been proved with \mathbf{D} and $\mathbf{C} \mathbf{A}^k \mathbf{B}$ being 1×1 matrices, implicitly converted to real numbers. To be applied on the error filter, the proof needs to be generalized to MIMO filters, which is not difficult but cumbersome due to matrix manipulation.

To handle more realizations and develop proofs (such as error analysis proofs) only once, we may use another realization called the Specialized Implicit Framework (SIF) [22]. It was designed as a unifying tool to describe and encompass all the possible realizations of a given transfer function (like the direct forms, State-Spaces, cascade or parallel decomposition, etc.). SIF is an extension of the State-Space realization, modified to allow chained Sum-of-Products operations.

A natural perspective is to handle floating-point and fixed-point computations. Indeed, digital filters are run on embedded software that cannot compute with real numbers. As far as floating-point arithmetic is concerned, the Flocq library [23] will suit our needs, but fixed-point will be more complicated. Even if Flocq has a fixed-point format and the corresponding theorems, we want to take overflow into account and this is hardly done within Flocq: only the IEEE-754 formalization of binary floating-point numbers assumes an upper bound on the exponent. Moreover, we may want to have several handling of overflow as done in [4]. We want at least three modes: (i) ensuring that no overflow happens; (ii) two's complement arithmetic, where a modulo operation is used when overflow happens; (iii) saturation arithmetic, where the maximal value is used when overflow happens. Adding two's complement and overflow modes to Flocq will be a necessary step towards the formal proof of the behaviors of embedded digital filters.

The final use of this work is to handle industrial applications, like the filters and controllers used in telecommunication, automotive or aeronautic with the following flow. Some filter specifications (like a transfer function) first gives an algorithm to realize the filter (like the Direct Forms or the State-Space). Then it is transformed in finite-precision code to be executed on a specific target. The bound of the output error (due to the finite-precision arithmetic) will be then deduced and proved.

References

1. Akbarpour, B., Tahar, S.: Error analysis of digital filters using HOL theorem proving. *J. Appl. Logic* **5**(4), 651–666 (2007). 4th International Workshop on Computational Models of Scientific Reasoning and Applications
2. Hilaire, T., Lopez, B.: Reliable implementation of linear filters with fixed-point arithmetic. In: *Proceedings of IEEE Workshop on Signal Processing Systems (SiPS)* (2013)
3. Siddique, U., Mahmoud, M.Y., Tahar, S.: Formal analysis of discrete-time systems using z-transform. *J. Appl. Logic*, 1–32 (2018, accepted). Elsevier
4. Akbarpour, B., Tahar, S., Dekdouk, A.: Formalization of fixed-point arithmetic in HOL. *Formal Methods Syst. Des.* **27**(1), 173–200 (2005)
5. Park, J., Pajic, M., Lee, I., Sokolsky, O.: Scalable verification of linear controller software. In: Chechik, M., Raskin, J.-F. (eds.) *TACAS 2016*. LNCS, vol. 9636, pp. 662–679. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_43
6. Park, J., Pajic, M., Sokolsky, O., Lee, I.: Automatic verification of finite precision implementations of linear controllers. In: Legay, A., Margaria, T. (eds.) *TACAS 2017 Part I*. LNCS, vol. 10205, pp. 153–169. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_9
7. Feret, J.: Static analysis of digital filters. In: Schmidt, D. (ed.) *ESOP 2004*. LNCS, vol. 2986, pp. 33–48. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24725-8_4
8. Bertot, Y., Castéran, P.: *Interactive Theorem Proving and Program Development*. TTCS. Springer, Heidelberg (2004). <https://doi.org/10.1007/978-3-662-07964-5>
9. The Coq Development Team: *The Coq Proof Assistant Reference Manual* (2017)
10. Boldo, S., Lelay, C., Melquiond, G.: Coquelicot: a user-friendly library of real analysis for Coq. *Math. Comput. Sci.* **9**(1), 41–62 (2015)
11. Oppenheim, A.V., Schaffer, R.W., Buck, J.R.: *Discrete-Time Signal Processing*, 2nd edn. Prentice-Hall Inc., Upper Saddle River (1999)
12. Siddique, U., Mahmoud, M.Y., Tahar, S.: On the formalization of Z-transform in HOL. In: Klein, G., Gamboa, R. (eds.) *ITP 2014*. LNCS, vol. 8558, pp. 483–498. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08970-6_31
13. Gonthier, G., et al.: A machine-checked proof of the odd order theorem. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) *ITP 2013*. LNCS, vol. 7998, pp. 163–179. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39634-2_14
14. Fettweiss, A.: Wave digital filters: theory and practice. *Proc. IEEE* **74**(2), 270–327 (1986)
15. Middleton, R., Goodwin, G.: *Digital Control and Estimation, a Unified Approach*. Prentice-Hall International Editions, Upper Saddle River (1990)
16. Li, G., Wan, C., Bi, G.: An improved ρ -DFIIt structure for digital filters with minimum roundoff noise. *IEEE Trans. Circ. Syst.* **52**(4), 199–203 (2005)
17. Hanselmann, H.: Implementation of digital controllers - a survey. *Automatica* **23**(1), 7–32 (1987)
18. Gevers, M., Li, G.: *Parametrizations in Control, Estimation and Filtering Problems*. Springer, Heidelberg (1993). <https://doi.org/10.1007/978-1-4471-2039-1>
19. Balakrishnan, V., Boyd, S.: On computing the worst-case peak gain of linear systems. *Syst. Control Lett.* **19**, 265–269 (1992)
20. Boyd, S.P., Doyle, J.: Comparison of peak and RMS gains for discrete-time systems. *Syst. Control Lett.* **9**(1), 1–6 (1987)

21. Kailath, T.: Linear Systems. Prentice-Hall, Upper Saddle River (1980)
22. Hilaire, T., Chevrel, P., Whidborne, J.: A unifying framework for finite wordlength realizations. *IEEE Trans. Circ. Syst.* **8**(54), 1765–1774 (2007)
23. Boldo, S., Melquiond, G.: Computer Arithmetic and Formal Proofs. ISTE Press - Elsevier, London (2017)



MathTools: An Open API for Convenient MATHML Handling

André Greiner-Petter¹(✉), Moritz Schubotz¹, Howard S. Cohl², and Bela Gipp¹

¹ Department of Computer and Information Science, University of Konstanz,
Box 76, 78464 Konstanz, Germany

{andre.greiner-petter,moritz.schubotz,bela.gipp}@uni-konstanz.de

² Applied and Computational Mathematics Division,
National Institute of Standards and Technology, Mission Viejo, CA 92694, USA
howard.cohl@nist.gov

Abstract. Mathematical formulae carry complex and essential semantic information in a variety of formats. Accessing this information with different systems requires a standardized machine-readable format that is capable of encoding presentational and semantic information. Even though MATHML is an official recommendation by W3C and an ISO standard for representing mathematical expressions, we could identify only very few systems which use the full descriptiveness of MATHML. MATHML's high complexity results in a steep learning curve for novice users. We hypothesize that this complexity is the reason why many community-driven projects refrain from using MATHML, and instead develop problem-specific data formats for their purposes. We provide a user-friendly, open-source application programming interface for controlling MATHML data. Our API allows one to create, manipulate, and efficiently access commonly needed information in presentation and content MATHML. Our interface also provides tools for calculating differences and similarities between MATHML expressions. The API also allows one to determine the distance between MATHML expressions using different similarity measures. In addition, we provide adapters for numerous conversion tools and the canonicalization project. Our toolkit facilitates processing of mathematics for digital libraries without the need to obtain XML expertise.

Keywords: MATHML · API · Toolkit · Java

1 Introduction

MATHML has the ability to represent presentational and content information. Several optional features and markup options make MATHML a highly versatile, but complex format. MATHML's ability to mix presentation and content markup for an expression (hereafter called fully descriptive MATHML) makes MATHML increasingly important for mathematical digital libraries. For example, Listing 1 shows a simple example of a cross-referenced parallel markup MATHML document generated from the \LaTeX string $\frac{a}{b}$.

Listing 1. Parallel markup MathML with examples of cross-references.

```
<math><semantics>
  <mfrac id="p.2" xref="c.1" >
    <mi id="p.1" xref="c.2">a</mi>
    <mi id="p.3" xref="c.3">b</mi></mfrac>
<annotation-xml encoding="MathML-Content"><apply>
  <divide id="c.1" xref="p.2" />
  <ci id="c.2" xref="p.1">a</ci>
  <ci id="c.3" xref="p.3">b</ci></apply></annotation-xml>
<annotation encoding="application/x-tex">\frac{a}{b}</annotation>
</semantics></math>
```

Although MATHML is an official recommendation of the [World Wide Web Consortium](#) since 1998, has been an ISO standard (ISO/IEC 40314) since 2015, and is also part of HTML5, it is still a rarely used format. For example, the prominent browser Microsoft Internet Explorer does not support MathML. Google Chrome supported MathML only in version 24 and dropped it again in newer versions in favor of MathJax¹. Furthermore, we were only able to identify few databases that use fully descriptive MATHML. Most databases use basic MATHML instead, such as the DLMF [4] which only provides presentational MATHML. Numerous tools that process math allow for MATHML as an input or export format. Most of these tools avoid MATHML as an internal data representation. Instead, many systems create their own problem-specific data representation and implement custom tools to parse MATHML for internal processing. We hypothesize that the complexity of MATHML and the steep learning curve required to master MATHML are the main reasons why community-driven projects usually avoid the use of content MATHML in their mathematical databases. This workaround causes problems in regard to reusability, restricts applicability, and decreases efficiency for processing mathematics.

During our research of a MATHML benchmark [17], we realized that parsing MATHML data is an error-prone endeavor. Even small changes, such as missing default namespaces, can cause the parsing process to fail.

With MathTools, we provide a collection of tools that overcome the issues of complexity and simplifies access to MATHML data. MathTools also contains an open API for easy access to useful services that we used in several of our previous projects, such as for similarity calculations or for \LaTeX to MATHML conversion tools [3, 9, 10, 12, 15–19]. The tools we have developed are able to parse fully descriptive MATHML and to a certain degree, invalid MATHML. Furthermore, we provide easy access to several useful features implemented in MATHML related projects. These features include similarity metrics, distance calculations, and Java imports for state-of-the-art conversion tools.

2 Related Work

Many digital libraries avoid using MATHML and process mathematics using custom internal formats instead. The conversion tool \LaTeX XML [11] is able to

¹ <https://bugs.chromium.org/p/chromium/issues/detail?id=152430#c43>.

generate presentation and content MATHML using \LaTeX input. Instead of processing the input directly to MATHML, \LaTeXML uses a customized XML format, and creates MATHML in a post-processing step. The Speech Rule Engine (SRE) implemented in MathJax [2] is able to translate presentation MATHML expressions to speech. Translations of the SRE use a set of rules that are applied to a nonstandard tree representation of the original MATHML input. Computer Algebra Systems (CAS) allow for complex computations of mathematical expressions. These systems usually create their own internal data format, but typically allow for MATHML export. For example, the CAS MAPLE² parses inputs to a directed acyclic graph (DAG) presentation [1, Chap. 2], whereas Mathematica uses a custom tree representation [20, Chap. 33].

There are other tools that internally use MATHML in combination with customized implementations for parsing MATHML. Examples such tools include MATHMLCAN [5], a canonicalization tool for MATHML data; the Visualization of Mathematical Expression Trees (VMEXT) [19] for MATHML data; and dependency graphs [7] for analyzing relationships between math expressions.

3 MathTools

We provide MathTools, a collection of tools for convenient handling of MathML data and an open Java API to access useful services related to MathML data. MathTools allows one to parse, manipulate, and analyze fully descriptive MATHML data conveniently. We have chosen Java to realize MathTools, because the majority of related work is also implemented in Java. Java is also the most frequently used programming language. The source of this project is publicly available on GitHub³ and on maven-central under an Apache 2 license.

The project is organized in multiple modules, of which the **core** module provides the main functionality. Figure 1 illustrates possible workflows for processing the input and shows the interaction of different modules. The **core** module is designed for parsing, loading, storing, and manipulating presentation and content MATHML data. The **core** also provides helper functions for easy access to specific information. Examples of supported operations include splitting presentation and content trees; accessing original \TeX data if available; and extracting all of the identifiers, e.g., **mi** or **ci** elements from the data. The **core** module also allows one to ‘clean’ MATHML of unwanted features. Such features include cross-references or entire content subtrees. This functionality is particularly helpful for obtaining simple MATHML for testing and learning purposes.

² The mention of specific products, trademarks, or brand names is for purposes of identification only. Such mention is not to be interpreted in any way as an endorsement or certification of such products or brands by the National Institute of Standards and Technology, nor does it imply that the products so identified are necessarily the best available for the purpose. All trademarks mentioned herein belong to their respective owners.

³ <https://github.com/ag-gipp/MathMLTools>.

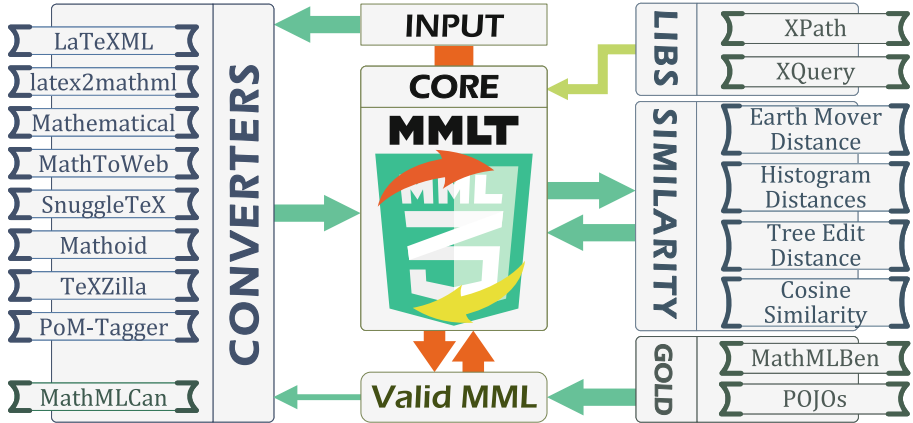


Fig. 1. The pipeline schema of MathTools and the modules. The orange arrow indicates the general workflow for processing a MATHML input to a valid MATHML output. The converters also allows for mathematical L^AT_EX input, while the MATHML feature requires valid MATHML input. The gold module provides valid MATHML without the core module. Distances and similarities can be calculated using the similarity module. Single elements or subtrees from valid MATHML can be accessed through the core module. (Color figure online)

We equipped MathTools with several extra modules, which we consider useful for typical use cases related to processing MATHML data. MathTools contains the following modules.

Gold: is part of the MATHMLBEN project [17], whose purpose is to provide a comfortable interface to MATHML gold standards within Java. The module uses Plain Old Java Objects (POJOs) for representing entries of the gold standard. The **gold** module can be used to implement benchmarks that use MATHML.

Converters: is an outgrowth of the MATHMLBEN project and provides fast access to state-of-the-art conversion tools. We implemented a Java interface for each supported tool and the MATHMLCAN [5] that allows canonicalization of the tool’s outputs. The module allows one to: (1) embed uniform MATHML data into translation workflows; (2) conveniently change the translation engines; and (3) add new conversion tools to the API.

Libraries: is a collection of reliable XPath and XQuery expressions for accessing elements in MATHML data. Due to the XML structure of MATHML documents, MathTools uses data query languages to access specific elements. The **library** module can be used to reliably query data from MATHML documents independently of programming languages.

Similarity: implements several distance and similarity measures for MATHML documents. Comparing MATHML data is a common task for many mathematical information retrieval systems, e.g., for mathematical search engines and for plagiarism detection systems that analyze mathematical expressions, as well as for

many evaluation and benchmark scenarios. All measures included in the module, except the tree edit distance, produce histograms for MATHML elements using the names of the elements and their accumulated frequency of occurrence. Frequency histograms of MATHML elements have been successfully applied as part of plagiarism detection methods to determine the similarity of mathematical expressions, see [9, 10] for a more detailed explanation. We implemented the following similarity measures.

- **Histogram Distance:** calculates the absolute and relative differences of histograms formed from MATHML data. A small histogram distance indicates semantic similarity between MATHML expressions.
- **Tree Edit Distance:** calculates the number of changes that must be performed to transform one MATHML tree into another. The user can control the weights for insertions, deletions, and renaming of elements. We use an implementation of RTED [13] to perform these calculations. Tree edit distances are helpful for detecting structural changes of MATHML trees.
- **Earth Mover Distance (EMD):** is originally a distance measure for comparing probability distributions. The measure models a cost function for changing one distribution to another. In our case, the probability distribution is the histogram of the MATHML data. Our calculations are performed using a Java implementation of [14]. EMD is widely used in multimedia information retrieval and for pattern recognition algorithms.
- **Cosine Similarity:** is a distance measure between two non-zero vectors. In our case, the vectors are represented by the histogram of the MATHML data.

Note that EMD and Cosine Similarity can be used for entire documents. In this case, the histograms are an accumulation of all MATHML expressions in the document. In this scenario, EMD and Cosine Similarity metrics obtain semantic similarities between documents.

4 Conclusion and Future Work

The MathTools project is a unified toolkit to facilitate working with MATHML. By using MathTools, other researchers will no longer need to develop their own data representations, nor will they have to deal with the full complexity of XML. The developed tools are actively being used in our own projects, such as MATHMLBEN [17], VMEXT [19], the Mathematical Language Processing (MLP) Project [12], and HyPlag [6, 8, 10] for mathematical plagiarism detection. Our hope is that the tools presented in this paper will help others to realize MATHML related projects in the future.

Our goal is to actively maintain and extend MathTools. Plans for current and future work include enlarging the libraries of XPath and XQuery expressions and providing transformation tools for MATHML trees. Furthermore, we are currently working on semantic enhancements of MATHML through the use of WikiData (as proposed in the MATHMLBEN project). This semantic enhanced MATHML data

can be used to calculate semantic distances through the calculation of distances between WikiData links. We plan to maintain the current API for the foreseeable future.

Acknowledgements. We would like to thank Felix Hamborg, Vincent Stange, Jimmy Li, Telmo Menezes, and Michael Kramer for contributing to the MathTools project. We are also indebted to Akiko Aizawa for her advice and for hosting the first two authors as visiting researchers at the National Institute of Informatics (NII) in Tokyo. This work was supported by the FITWeltweit program of the German Academic Exchange Service (DAAD) as well as by the German Research Foundation (DFG) through grant no. GI 1259/1.

References

1. Bernardin, L., et al.: Maple 2016 Programming Guide. Maplesoft, a division of Waterloo Maple Inc. (2016). ISBN 978-1-926902-46-3
2. Cervone, D., Krautzberger, P., Sorge, V.: Towards universal rendering in MathJax. In: Proceedings of the W4A 2016. ACM Press (2016). <https://doi.org/10.1145/2899475.2899494>
3. Cohl, H.S., et al.: Growing the digital repository of mathematical formulae with generic LaTeX sources. In: Kerber, M., Carette, J., Kaliszky, C., Rabe, F., Sorge, V. (eds.) CICM 2015. LNCS (LNAI), vol. 9150, pp. 280–287. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20615-8_18
4. Olver, F.W.J., et al. (eds.): NIST Digital Library of Mathematical Functions. <http://dlmf.nist.gov/>. Release 1.0.19 of 22 June 2018
5. Formánek, D., et al.: Normalization of digital mathematics library content. In: Davenport, J., et al. (eds.) Proceeding of OpenMath/MathUI/CICM-WiP, vol. 921, Bremen, 9–13 July 2012
6. Gipp, B., et al.: Web-based demonstration of semantic similarity detection using citation pattern visualization for a cross language plagiarism case. In: ICEIS 2014 - Proceedings of the 16th International Conference on Enterprise Information Systems, vol. 2, Lisbon, 27–30 April 2014. <https://doi.org/10.5220/0004985406770683>
7. Kristianto, G.Y., Topic, G., Aizawa, A.: Utilizing dependency relationships between math expressions in math IR. *Inf. Retr. J.* **20**(2) (2017). <https://doi.org/10.1007/s10791-017-9296-8>
8. Meuschke, N., Gipp, B., Breiting, C.: CitePlag: a citation-based plagiarism detection system prototype. In: Proceedings of the 5th International Plagiarism Conference, Newcastle upon Tyne (2012)
9. Meuschke, N., et al.: Analyzing mathematical content to detect academic plagiarism. In: Lim, E., et al. (eds.) Proceedings of the ACM CIKM. ACM (2017). <https://doi.org/10.1145/3132847.3133144>
10. Meuschke, N., et al.: HyPlag: a hybrid approach to academic plagiarism detection. In: Proceedings of the SIGIR, Ann Arbor (2018)
11. Miller, B.R.: LaTeXML: A L^AT_EX to XML/HTML/MathML Converter. <http://dlmf.nist.gov/LaTeXML/>. Accessed June 2018
12. Pagel, R., Schubotz, M.: Mathematical language processing project. In: England, M., et al. (eds.) Proceedings of the MathUI/OpenMath/ThEdu/CICM-WiP, vol. 1186 (2014)

13. Pawlik, M., Augsten, N.: RTED: a robust algorithm for the tree edit distance. In: CoRR abs/1201.0230 (2012). [arXiv:1201.0230](https://arxiv.org/abs/1201.0230)
14. Pele, O., Werman, M.: Fast and robust earth mover's distances. In: IEEE 12th ICCV, Kyoto. IEEE Computer Society, 27 September–4 October 2009. <https://doi.org/10.1109/ICCV.2009.5459199>
15. Schubotz, M.: Augmenting mathematical formulae for more effective querying & efficient presentation. Ph.D. thesis, TU, Berlin (2017). <https://doi.org/10.14279/depositonce-6034>. ISBN 978-3-7450-6208-3
16. Schubotz, M., Krämer, L., Meuschke, N., Hamborg, F., Gipp, B.: Evaluating and improving the extraction of mathematical identifier definitions. In: Jones, G., et al. (eds.) CLEF 2017. LNCS, vol. 10456, pp. 82–94. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65813-1_7
17. Schubotz, M., et al.: Improving the representation and conversion of mathematical formulae by considering their textual context. In: Proceedings of the ACM/IEEE-CS JCDL, Fort Worth, June 2018. <https://doi.org/10.1145/3197026.3197058>
18. Schubotz, M., et al.: Semantification of identifiers in mathematics for better math information retrieval. In: Proceedings of the 39th International ACM SIGIR, Pisa. ACM (2016). <https://doi.org/10.1145/2911451.2911503>. ISBN 978-1-4503-4069-4
19. Schubotz, M., Meuschke, N., Hepp, T., Cohl, H.S., Gipp, B.: VMEXT: a visualization tool for mathematical expression trees. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) CICM 2017. LNCS (LNAI), vol. 10383, pp. 340–355. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_24
20. Wolfram, S.: An Elementary Introduction to the Wolfram Language, 2nd edn. Wolfram Media (2017). ISBN 978-1944183059



Aligator.jl – A Julia Package for Loop Invariant Generation

Andreas Humenberger^{1(✉)}, Maximilian Jaroschek^{1,2}, and Laura Kovács^{1,3}

¹ TU Wien, Vienna, Austria

ahumenbe@forsyte.at

² JKU Linz, Linz, Austria

³ Chalmers, Gothenburg, Sweden

Abstract. We describe the `Aligator.jl` software package for automatically generating all polynomial invariants of the rich class of extended P-solvable loops with nested conditionals. `Aligator.jl` is written in the programming language Julia and is open-source. `Aligator.jl` transforms program loops into a system of algebraic recurrences and implements techniques from symbolic computation to solve recurrences, derive closed form solutions of loop variables and infer the ideal of polynomial invariants by variable elimination based on Gröbner basis computation.

1 Introduction

In [2] we described an automated approach for generating loop invariants as a conjunction of polynomial equalities for a family of loops, called extended P-solvable loops. For doing so, we abstract loops to a system of algebraic recurrences over the loop counter and program variables and compute polynomial equalities among loop variables from the closed form solutions of the recurrences.

Why Julia? Our work was previously implemented in the `Aligator` software package [4], within the Mathematica system [8]. While Mathematica provides high-speed implementations of symbolic computation techniques, it is a proprietary software which is an obstacle for using `Aligator` in applications of invariant generation. The fact that Mathematica provides no possibility to parse and modify program code was also a reason to move to another environment. To make `Aligator` better suited for program analysis, we decided to redesign `Aligator` in the Julia programming language [3]. Julia provides a simple and efficient interface for calling C/C++ and Python code. This allows us to resort to already existing computer algebra libraries, such as Singular [1] and SymPy [5]. Julia also provides a built-in package manager that eases the use of other packages and enables

All authors are supported by the ERC Starting Grant 2014 SYMCAR 639270. We also acknowledge funding from the Wallenberg Academy Fellowship 2014 TheProSE, the Swedish VR grant GenPro D0497701, and the Austrian FWF research projects RiSE S11409-N23 and W1255-N23. Maximilian Jaroschek is also supported by the FWF project Y464-N18.

others to use Julia packages, including our `Aligator.jl` tool. Before committing to Julia, we also considered the computer algebra system SageMath [7] and an implementation directly in C/C++ as options for redesigning `Aligator`. The former hosts its own Python version which makes the installation of other Python packages (e.g. for parsing source code) tedious and error-prone. While C/C++ is very efficient and provides a large ecosystem on existing libraries, developing C/C++ projects requires more effort than Julia packages. We therefore believe that Julia provides the perfect mix between efficiency, extensibility and convenience in terms of programming and symbolic computations.

`Aligator.jl`. This paper overviews `Aligator.jl` and details its main components. The code of `Aligator.jl` is available open-source at:

<https://github.com/ahumenberger/Aligator.jl>.

All together, `Aligator.jl` consists of about 1250 lines of Julia code. We evaluated `Aligator.jl` on challenging benchmarks on invariant generation. Our experimental results are available at the above mentioned link and demonstrate the efficiency of `Aligator.jl`.

Contributions. Our new tool `Aligator.jl` significantly extends and improves the existing software package `Aligator` as follows:

- Unlike `Aligator`, `Aligator.jl` is open-source and easy to integrate into other software packages.
- `Aligator.jl` implements symbolic computation techniques directly in Julia for extracting and solving recurrences and generates polynomial dependencies among exponential sequences.
- Contrarily to `Aligator`, `Aligator.jl` handles not only linear recurrences with constant coefficients, called C-finite recurrences. Rather, `Aligator.jl` also supports hypergeometric sequences and sums and term-wise products of C-finite and hypergeometric recurrences [2].
- `Aligator.jl` is complete. That is, a finite basis of the polynomial invariant ideal is always computed.

2 Background and Notation

`Aligator.jl` computes polynomial invariants of so-called extended P-solvable loops [2]. Loop guards and test conditions are ignored in such loops and denoted by `...` or `true`, yielding non-deterministic loops with sequencing and conditionals. Program variables $V = \{v_1, \dots, v_m\}$ of extended P-solvable loops have numeric values, abstracted to be rational numbers. The assignments of extended P-solvable loops are of the form $v_i := \sum_{j=0}^m c_j v_j + c_{m+1}$ with constants c_0, \dots, c_{m+1} , or $v_i := r(n)v_i$, where $r(n)$ is a rational function in the loop counter n . We give an example of an extended P-solvable loops in Fig. 1.

In correspondence to V , the initial values of the variables are given by the set $V_0 := \{v_1(0), \dots, v_m(0)\}$; that is, $v_i(0)$ is the initial value of v_i . In what follows, we consider V and V_0 fixed and state all definitions relative to them. Given an extended P-solvable loop as input, `Aligator.jl` generates all its polynomial equality invariants. By a polynomial equality invariant, in the sequel simply polynomial invariant, we mean the equality:

$$p(v_1, \dots, v_m, v_1(0), \dots, v_m(0)) = 0, \tag{1}$$

where p is a polynomial in $V \cup V_0$ with rational number coefficients. In what follows, we also refer to the polynomial p in (1) as a polynomial invariant. For $n \in \mathbb{N} \setminus \{0\}$ and a loop variable v_i , we write $v_i(n)$ to denote the value of v_i after the n th loop iteration. As (1) is a loop invariant, we have:

$$p(v_1(n), \dots, v_m(n), v_1(0), \dots, v_m(0)) = 0 \text{ for } n > 0.$$

As shown in [2,6], the set of polynomial invariants in V , w.r.t. the initial values V_0 , forms a polynomial ideal, called the polynomial invariant ideal. Given an extended P-solvable loop, `Aligator.jl` computes *all* its polynomial invariants as it computes a basis of the polynomial invariant ideal, a finite set of polynomials $\{b_1, \dots, b_k\}$. Any polynomial invariant can be written as a linear combination $p_1 b_1 + \dots + p_k b_k$ for some polynomials p_1, \dots, p_k .

```

while ... do
  if ... then
    r := r - v; v := v + 2
  else
    r := r + u; u := u + 2
  end if
end while

```

Fig. 1. An extended P-solvable loop.

3 System Description of `Aligator.jl`

Inputs to `Aligator.jl` are extended P-solvable loops and are fed to `Aligator.jl` as `String` in the Julia syntax. We illustrate the use of `Aligator.jl` on our example from Fig. 1:

Example 1. Fig. 1 is specified as a Julia string as follows:

```

julia> loopstr = """
    while true
      if true
        r = r - v; v = v + 2
      else
        r = r + u; u = u + 2
      end
    end
    """

```

Polynomial loop invariants are inferred using `Aligator.jl` by calling the function `aligator(str::String)` with a string input containing the loop as its argument.

```

julia> aligator(loopstr)
Singular Ideal over Singular Polynomial Ring (QQ), (r_0,v_0,u_0,r,v,u)
with generators (v_0^2-u_0^2-v^2+u^2+4*r_0-2*v_0+2*u_0-4*r+2*v-2*u)

```

The result of `Aligator.jl` is a Gröbner basis of the polynomial invariant ideal. It is represented as an object of type `Singular.sideal` that is defined in the `Singular` package. For Fig. 1, `Aligator.jl` reports that the polynomial invariant ideal is generated by the polynomial invariant $\{v_0^2 - u_0^2 - v^2 + u^2 + 4r_0 - 2v_0 + 2u_0 - 4r + 2v - 2u = 0\}$ in variables r_0, v_0, u_0, r, v, u , where r_0, v_0, u_0 denote respectively the initial values of r, v, u .

We now overview the main parts of `Aligator.jl`: (i) extraction of recurrence equations, (ii) recurrence solving and (iii) computing the polynomial invariant ideal.

Extraction of Recurrences. Given an extended P-solvable loop as a Julia string, `Aligator.jl` creates the abstract syntax tree of this loop. This tree is then traversed in order to extract loop paths (in case of a multi-path loop) and the corresponding loop assignments. The resulting structure is then flattened in order to get a loop with just one layer of nested loops. Within `Aligator.jl` this is obtained via the method `extract_loop(str::String)`. As a result, the extracted recurrences are represented in `Aligator` by an object of type `Aligator.MultiLoop`, in case the input is a multi-path loop; otherwise, the returned object is of type `Aligator.SingleLoop`.

Example 2. Using Example 1, `Aligator.jl` derives the loop and its corresponding systems of recurrences:

```

julia> loop = extract_loop(loopstr)
2-element Aligator.MultiLoop:
 [r(n1+1) = r(n1) - v(n1), v(n1+1) = v(n1) + 2, u(n1+1) = u(n1)]
 [r(n2+1) = r(n2) + u(n2), u(n2+1) = u(n2) + 2, v(n2+1) = v(n2)]

```

As loop paths are translated into single-path loops, `Aligator.jl` introduces a loop counter for each path and computes the recurrence equations of the loop variables r, v, u with respect to the loop counters n_1 and n_2 .

Recurrence Solving. For each single-path loop, its system of recurrences is solved. `Aligator.jl` performs various simplifications on the extracted recurrences, for example by eliminating cyclic dependencies introduced by auxiliary variables and uncoupling mutually dependent recurrences. The resulting, simplified recurrences represent sums and term-wise products of C-finite or hypergeometric sequences. `Aligator.jl` computes closed forms solutions of such recurrences by calling the method `closed_forms` and using the symbolic manipulation capabilities of `SymPy.jl`:

Example 3. For Example 2, we get the following systems of closed forms:

```

julia> cforms = closed_forms(loop)
2-element Array{Aligator.ClosedFormSystem,1}:
 [v(n1) = 2*n1+v(0), u(n1) = u(0), r(n1) = -n1^2-n1*(v(0)-1)+r(0)]
 [u(n2) = 2*n2+u(0), v(n2) = v(0), r(n2) = n2^2+n2*(u(0)-1)+r(0)]

```

The returned value is an array of type `Aligator.ClosedFormSystem`.

Invariant Ideal Computation. Using the closed form solutions for (each) single-path loop, `Aligator.jl` next derives a basis of the polynomial invariant ideal of the (multi-path) extended P-solvable loop. To this end, `Aligator.jl` uses the `Singular.jl` package for Gröbner basis computations in order to eliminate variables in the loop counter(s) from the system of closed forms. For multi-path loops, `Aligator.jl` relies on iterative Gröbner basis computations until a fixed point is derived representing a Gröbner basis of the polynomial invariant ideal – see [2] for theoretical details.

Computing polynomial invariants within `Aligator.jl` is performed by the function `invariants(cforms::Array{ClosedFormSystem,1})`. The result is an object of type `Singular.sideal` and represents a Gröbner basis of the polynomial invariant ideal in the loop variables.

Example 4. For Example 3, `Aligator.jl` generates the following Gröbner basis, as already described on page 4:

```

julia> ideal = invariants(cforms)
Singular Ideal over Singular Polynomial Ring (QQ), (r_0,v_0,u_0,r,v,u)
with generators (v_0^2-u_0^2-v^2+u^2+4*r_0-2*v_0+2*u_0-4*r+2*v-2*u)

```

4 Experimental Evaluation

Our approach to invariant generation was shown to outperform state-of-the-art tools on invariant generation for multi-path loops with polynomial arithmetic [2]. In this section we focus on the performance of our new implementation in `Aligator.jl` and compare results to `Aligator` [4]. In our experiments, we used benchmarks from [2]. Our experiments were performed on a machine with a 2.9 GHz Intel Core i5 and 16 GB LPDDR3 RAM. When using `Aligator.jl`, the invariant ideal computed by `Aligator.jl` was non-empty for each example; that is, for each example we were able to find non-trivial invariants.

Tables 1(a) and (b) show the results for a set of single- and multi-path loops respectively. In both tables the first column shows the name of the instance, whereas columns two and three depict the running times (in seconds) of `Aligator` and `Aligator.jl`, respectively.

By design, `Aligator.jl` is at least as strong as `Aligator` concerning the quality of the output. When it comes to efficiency though, we note that `Aligator.jl` is slower than `Aligator`. We expected this result as `Aligator` uses the highly optimized algorithms of Mathematica. When taking a closer look at how much time is spent in the different parts of `Aligator.jl`, we observed that the most time in `Aligator.jl` is consumed by symbolic manipulations. Experiments indicate that

Table 1. Experimental evaluation of `Aligator.jl`.

(a)			(b)		
<i>Single-path</i>	<code>Aligator</code>	<code>Aligator.jl</code>	<i>Multi-path</i>	<code>Aligator</code>	<code>Aligator.jl</code>
<code>cohencu</code>	0.072	2.879	<code>divbin</code>	0.134	1.760
<code>freire1</code>	0.016	1.159	<code>euclidex</code>	0.433	3.272
<code>freire2</code>	0.062	2.540	<code>fermat</code>	0.045	2.159
<code>petter1</code>	0.015	0.876	<code>knuth</code>	55.791	12.661
<code>petter2</code>	0.026	1.500	<code>lcm</code>	0.051	2.089
<code>petter3</code>	0.035	2.080	<code>mannadiv</code>	0.022	1.251
<code>petter4</code>	0.042	3.620	<code>wensley</code>	0.124	1.969

we can improve the performance of `Aligator.jl` considerably by using the Julia package `SymEngine.jl` instead of `SymPy.jl`. We believe that our initial experiments with `Aligator.jl` are promising and demonstrate the use of our efforts in making our invariant generation open-source.

5 Conclusion

We introduced the new package `Aligator.jl` for loop invariant generation in the programming language Julia. Our `Aligator.jl` tool is an open-source software package for invariant generation using symbolic computation and can easily be integrated with other libraries and tools.

References

1. Decker, W., Greuel, G.M., Pfister, G., Schönemann, H.: SINGULAR 4-1-0—a computer algebra system for polynomial computations (2016). <http://www.singular.uni-kl.de>
2. Humenberger, A., Jaroschek, M., Kovács, L.: Invariant generation for multi-path loops with polynomial assignments. In: Dillig, I., Palsberg, J. (eds.) VMCAI. LNCS, vol. 10747, pp. 226–246. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73721-8_11
3. Julia. <https://julialang.org/>
4. Kovács, L.: Aligator: a mathematica package for invariant generation (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 275–282. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71070-7_22
5. Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, S., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., Scopatz, A.: SymPy: symbolic computing in Python. *PeerJ Comput. Sci.* **3**, e103 (2017). <https://doi.org/10.7717/peerj-cs.103>

6. Rodríguez-Carbonell, E., Kapur, D.: Generating all polynomial invariants in simple loops. *J. Symb. Comput.* **42**(4), 443–476 (2007). <https://doi.org/10.1016/j.jsc.2007.01.002>
7. SageMath. <http://www.sagemath.org/>
8. Wolfram, S.: *An Elementary Introduction to the Wolfram Language*. Wolfram Media Inc. (2017). <https://www.wolfram.com/language/elementary-introduction/2nd-ed/>



Enhancing ENIGMA Given Clause Guidance

Jan Jakubův^(✉) and Josef Urban

Czech Technical University in Prague, Prague, Czech Republic
jakubuv@gmail.com

Abstract. ENIGMA is an efficient implementation of learning-based guidance for given clause selection in saturation-based automated theorem provers. In this work, we describe several additions to this method. This includes better clause features, adding conjecture features as the proof state characterization, better data pre-processing, and repeated model learning. The enhanced ENIGMA is evaluated on the MPTP2078 dataset, showing significant improvements.

1 ENIGMA: Efficient Learning of Given Clause Guidance

State-of-the-art saturation-based automated theorem provers (ATPs) for first-order logic (FOL), such as E [5], are today’s most advanced tools for general reasoning across a variety of mathematical and scientific domains. Many ATPs employ the *given clause algorithm*, translating the input FOL problem $T \cup \{\neg C\}$ into a refutationally equivalent set of clauses. The search for a contradiction is performed maintaining sets of *processed* (P) and *unprocessed* (U) clauses. The algorithm repeatedly selects a *given clause* g from U , extends U with all clauses inferred with g and P , and moves g to P . This process continues until a contradiction is found, U becomes empty, or a resource limit is reached. The search space of this loop grows quickly and it is a well-known fact that the selection of the right given clause is crucial for success.

ENIGMA [4] stands for *Efficient learning-based Inference Guiding Machine* that steers clause selection in saturation-based ATPs like E. ENIGMA is based on a simple but fast *logistic regression* algorithm effectively implemented by the LIBLINEAR open source library [2]. In order to employ logistic regression, first-order clauses need to be translated to fixed-length numeric *feature vectors*. ENIGMA uses (top-down-)oriented term-tree walks of length 3 as *features*. For example, a unit clause “ $P(f(a, b))$ ” contains only features “ (P, f, a) ” and “ (P, f, b) ” (see [4, Sect. 3.2] for details). Features are enumerated and a clause C is translated to the feature vector φ_C whose i -th member counts the number of occurrences of the i -th feature in clause C . We also count top-level literal symbols (positive or negative) and we unify variables and Skolem symbols.

J. Jakubův and J. Urban—This work was supported by the ERC Consolidator grant no. 649043 *AI4REASON*, and by the Czech project *AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466* and the European Regional Development Fund.

In order to train an ENIGMA *predictor* \mathcal{E} , all the given clauses \mathcal{C} from a set of previous successful proof searches are collected. The given clauses used in the proofs are classified as positive ($\mathcal{C}^+ \subseteq \mathcal{C}$) and the remaining given clauses as negative ($\mathcal{C}^- \subseteq \mathcal{C}$). The clause sets ($\mathcal{C}^+, \mathcal{C}^-$) are turned into feature vector sets (Φ^+, Φ^-) using a fixed *feature enumeration* π . Then a LIBLINEAR *classifier* w (a *weight vector*) is trained on the classification (Φ^+, Φ^-), classifying each clause as *useful* or *un-useful* for the proof search. The classifier w and enumeration π give us a predictor $\mathcal{E} = (w, \pi)$ which is used to guide next proof searches in combination with other E heuristics. Thanks to the fast feature extraction mechanism and the fast (linear) evaluation of the features in a particular learned model, there is no slowdown of the given clause loop. In fact, ENIGMA is faster than some of the more advanced hand-programmed E evaluation heuristics [3]. The training speed allows fast MaLAREa-style [8] iterative loop between ATP proving and re-learning [4, Sect. 5.1].

2 Enhanced ENIGMA Features and Classifiers

This section briefly describes improvements from the previous ENIGMA version [4].

Sparse Features Encoding. Previously, ENIGMA was tested only on the CASC 2016 AIM benchmark [7] which contains only about 10 different symbols. Using a straightforward dense encoding yielded feature vectors of size 10^3 , because ENIGMA features are triples of symbols. Such exhaustive enumeration of feature vectors was too big for larger symbol signatures. The term-tree walks features are, however, relatively sparse: symbols are typically applied only to a small number of other symbols. Hence we switched to a sparse encoding, using only the features which actually appear in the training data. This significantly reduced the feature vector sizes while preserving the predicting accuracy. This alone allows us to test ENIGMA on Interactive Theorem Proving (ITP) benchmarks which tend to have much larger signatures.

Conjecture Features. Second, for AIM we initially did not consider conjectures to be a part of the model. Hence the same clauses were being recommended in every possible AIM proof search. This can work (similarly to the *hint* method [9]) on benchmarks of very related problems (such as AIM), but hardly on large ITP-based formal libraries, which are much more heterogeneous. To overcome this, we embed the *conjecture features* in the feature vectors. For a clause C , instead of using the vector φ_C of length n (where n is the number of different features appearing in the training data), we use a vector (φ_C, φ_G) of length $2n$ where φ_G contains the features of the conjecture G . For a training clause C , G corresponds to the conjecture of the proof search where C was selected as a given clause. When classifying a clause C during a proof search, G corresponds to the conjecture currently being proved.

Horizontal Features. The previously described term-tree walk features of length 3, can be seen as *vertical* chains in the term-tree. In order to provide

a more accurate representation of clauses by feature vectors, we additionally introduce *horizontal* features. For every term $f(t_1, \dots, t_n)$, we introduce the feature $f(s_1, \dots, s_n)$ where s_i is the top-level symbol of the subterm t_i . Our feature enumeration π is extended with the horizontal features from the training data, and the feature vectors again record the number of occurrences of each such feature in a clause. For example, the unit clause “ $P(f(g(a), g(a)))$ ” is characterized by one occurrence of horizontal features “ $P(f)$ ” and “ $f(g, g)$ ”, and by two occurrences of “ $g(a)$ ”.

Static Features. We have also introduced *static* clause features, based on existing E’s *feature vectors* used for subsumption indexing [6]. The first three features are signature independent and consist of (1) the clause length, and (2–3) the counts of positive/negative literals. For each signature symbol f , we also use (4–5) the number of occurrences of f in positive/negative literals, (6–7) the maximal depth of any occurrence of f in positive/negative literals.

E uses *clause evaluation functions*, which assign a numeric *weight* to every clause, to select the next given clause. The weight is computed for every generated clause and the clause with the smallest weight is selected. The ENIGMA weight function for predictor \mathcal{E} assigns a smaller weight to clauses positively classified by \mathcal{E} . Previously, we had to combine this weight with the clause length to prevent E from generating very long positively classified clauses. With the new static features this is no longer necessary.

Accuracy-Balancing Boosting. The training data produced by previous proof searches are typically unbalanced in the number of positive and negative samples. Usually there are many more negatives and predictors trained directly on such data usually mis-classify positive examples much more than the negative ones. Previously, we used trivial boosting methods, like repeating positive samples ten times, to correct this behavior. Now we use the following iterative boosting method. Given training data $(\mathcal{C}_0^+, \mathcal{C}_0^-)$ we create an ENIGMA predictor \mathcal{E}_0 , test \mathcal{E}_0 on the training data, and collect the positives mis-classified by \mathcal{E}_0 . We then repeat (*boost*) the mis-classified positives in the training data, yielding updated $(\mathcal{C}_1^+, \mathcal{C}_0^-)$ and an updated predictor \mathcal{E}_1 . We iterate this process, and with every iteration, the accuracy on the positive samples is increased, while the accuracy on the negatives is typically decreased. We finish the boosting when the positive accuracy exceeds the negative one.

Directed Looping. An E *strategy* S is a collection of E’s options that influence the proof search. An important part of every strategy S is the collection of clause evaluation functions used for given clause selection. Given an ENIGMA predictor \mathcal{E} trained on proofs done with S , we can either (1) use \mathcal{E} alone to select the given clause, or we can (2) combine \mathcal{E} with other evaluation functions from S . In (2) we by default select half of the given clauses by \mathcal{E} , and the other half by the original selection functions from S . This gives rise to two new E strategies, denoted (1) $S \odot \mathcal{E}$, and (2) $S \oplus \mathcal{E}$. In practice, there is usually no clear winner between the two as there are problems solved by the first one but not the other, and the other way round.

Given a set of training problems P and a strategy S , we can run S for each problem from P , extract training samples $\mathcal{C}_0 = (\mathcal{C}_0^+, \mathcal{C}_0^-)$, and create the predictor \mathcal{E}_0 . Then we can evaluate both ENIGMA strategies $S \odot \mathcal{E}_0$ and $S \oplus \mathcal{E}_0$ on P and obtain additional training samples. This yields new training samples $\mathcal{C}_1 \supseteq \mathcal{C}_0$ and a new predictor \mathcal{E}_1 . The predictor \mathcal{E}_1 usually combines the strengths of both ENIGMA strategies $S \odot \mathcal{E}_0$ and $S \oplus \mathcal{E}_0$. This looping process can be iterated and after several iterations, strategies $S \odot \mathcal{E}_i$ and $S \oplus \mathcal{E}_i$ start performing almost equally, typically outperforming the initial strategy S on P . This way we produce an enhanced single strategy. Currently we just iterate this process three times. Future work includes adaptive termination criteria based, for instance, on a respective performance of $S \odot \mathcal{E}_i$ and $S \oplus \mathcal{E}_i$.

3 Experimental Evaluation

The previous version of ENIGMA was evaluated with a single E strategy on the CASC 2016 AIM benchmark [7]. Here we additionally evaluate on the Mizar MPTP2078 [1] dataset (MZR) as an example ITP benchmark. This section presents three experiments designed to (1) evaluate the effect of conjecture features on prediction accuracies, (2) evaluate other enhancements on the training data, and to (3) test the ability of ENIGMA to generalize to similar problems.

We use both the AIM and MZR benchmarks to evaluate the effect of conjecture features on predictor accuracy. We first evaluate 10 E strategies on the benchmarks and construct 10 different ENIGMA predictors. We measure the *10-fold cross-validation* accuracy, where the data are divided into 10 subsets (*folds*) with 9 folds used for training and one fold left aside for evaluation. As expected, adding the conjecture features helps on the MZR data, improving an averaged 10-fold cross-validation accuracy from 88.8% to 91.5%. On AIM this improves from 76.3% to 77.8%. We explain this by the AIM problems having more similar conjectures. The feature vector sizes vary from 60 to 130 on AIM and from 2300 to 22000 on MZR.

Next we evaluate the effect of the other enhancements on MZR. The MZR problems are naturally clustered into 26 categories, based on the original Mizar article which was the source for the Mizar-to-FOL translation. Hence the problems within each category are typically related and use similar symbols. The E auto mode contains more than 60 strategies and we select the ten best-performing on MZR problems. This gives us a portfolio of 10 well-performing strategies, denoted *autos*, and we attempt to further improve this portfolio using the ENIGMA predictors.

We train a separate predictor $\mathcal{E}_{S,A}$ for every strategy S from *autos*, and for every article (category) A , using all the enhancements from Sect. 2. This gives us 10 ENIGMA strategies per article (currently, for predictor $\mathcal{E}_{S,A}$ we simply take $S \oplus \mathcal{E}'_{S,A}$ after 3 iterations of looping). Now, on each article A , we can compare the portfolio of the ten *autos* with the portfolio of the ten ENIGMA strategies trained on A . This tells us how ENIGMA strategies perform on the training data. Table 1 summarizes the results for several articles. All the E runs were performed

Table 1. Best-performing ENIGMA models for selected articles.

article	total	autos	ENIG	ENIG+	article	total	autos	ENIG	ENIG+
compts	23	7	7	+0.0%	filter	65	6	7	+16.7%
enumset1	96	86	86	+0.0%	orders	61	28	36	+28.6%
pre	37	21	22	+4.8%	wellord1	59	27	35	+29.6%
relset	32	20	22	+10.0%	yellow19	38	12	18	+50.0%
funct	235	160	185	+15.6%	waybel	174	42	74	+76.2%

with 10s time limit per problem (precisely, each portfolio strategy runs for 1s on a problem). The *total* column shows the number of problems in the article A , *autos* is the number of problems solved by *autos*, the *ENIG* column presents the number of problems solved by the ENIGMA strategies, and finally *ENIG+* summarizes the gain in percentage of the ENIGMA strategies on *autos*. On the training data, ENIGMA strategies were able to solve all the problems solved by *autos* and also additional problems. The gain in percentage varies from 0% to a surprisingly high value of 76.2%, with 15.7% on average.

Table 2. Overall portfolio improvement.

portfolio	Solved	E%	autos%	autos+	autos−
E (auto-schedule)	1343	+0%	−3.8%	+25	−79
autos (10)	1397	+4.0%	+0%	+0	−0
ENIGMA (62)	1450	+7.9%	+3.7%	+103	−50

The next experiment is designed to test the ability of ENIGMA predictors to generalize, and to enrich an already well-performing portfolio. From the previous experiment, we have altogether $26 \cdot 10$ ENIGMA strategies. The strategies trained on article A have been already evaluated on A , and thus we can compute their greedy cover on A . To reduce the number of strategies, we take only the strategies in the greedy cover of some article A . This reduction gives us a portfolio of 62 ENIGMA strategies, which can now be compared with the performance of the ten *autos* on the whole MZR benchmark. We use the time limit of 300s per problem, equally divided among the strategies in a portfolio.

The results are presented in Table 2. We additionally evaluate E in the *auto-schedule* mode, which is a state-of-the-art general-purpose native strategy scheduler of E. The numbers in parentheses indicate the number of problems in a given portfolio. The *E%* column measures the gain on *auto-schedule* in percents, *autos%* measures the gain on *autos*, and *autos+* and *autos−* state the difference between the problems solved by the respective portfolio and by *autos*. The *autos* portfolio outperforms *auto-schedule* even though *auto-schedule* contains all the strategies from *autos* (and more). This is because we have constructed *autos* from the ten strategies which perform (greedily) best on the MZR benchmark.

We can see that the ENIGMA strategies solve 3.7% more problems than the initial *autos* strategies, and 7.9% more than *auto-schedule*. This means that the ENIGMA strategies are capable of generalizing to problems similar to the training problems with a measurable improvement. Without the enhancements from Sect. 2, we had been able to improve single strategies, but we were much less successful in enriching a well-performing portfolio of diverse strategies. The new ENIGMA enhancements are instrumental for achieving this.

4 Software Distribution

We added support for ENIGMA to E Prover¹ and we developed a simple Python module `atpy` for construction of ENIGMA models and evaluation of E strategies. The module is available at our web page² and it contains an example with instructions (see `examples/README.ENIGMA.md`) on how to use `atpy` for ENIGMA experiments. The required binaries for LIBLINEAR and the extended E Prover are provided (for x86) together with example benchmark problems and E strategies.

The `atpy` module is capable of (1) evaluation of E strategies on custom benchmark problems, (2) extraction of training samples from E outputs, (3) construction of ENIGMA models with boosting and looping, and of (4) construction of E strategies enhanced with ENIGMA predictors. The ENIGMA models are stored in the directory `Enigma` in separate subdirectories. The model directory contains files (1) `model.lin` with LIBLINEAR weight vector w and (2) `enigma.map` with feature enumeration π . Horizontal features are represented by strings of the form “ $f.s_1 \dots s_n$ ”, and vertical features by “ $s_1 : s_2 : s_3$ ”. The extended E supports our clause evaluation function `Enigma` which takes a relative model directory as an argument. Our E extension additionally includes a feature extraction tool `enigma-features` required by `atpy`.

References

1. Alama, J., Heskes, T., Kühlwein, D., Tsvitsvadze, E., Urban, J.: Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reason.* **52**(2), 191–213 (2014). <https://doi.org/10.1007/s10817-013-9286-5>
2. Fan, R., Chang, K., Hsieh, C., Wang, X., Lin, C.: LIBLINEAR: a library for large linear classification. *J. Mach. Learn. Res.* **9**, 1871–1874 (2008)
3. Jakubův, J., Urban, J.: Extending E prover with similarity based clause selection strategies. In: Kohlhase, M., Johansson, M., Miller, B., de Moura, L., Tompa, F. (eds.) *CICM 2016*. LNCS (LNAI), vol. 9791, pp. 151–156. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42547-4_11
4. Jakubův, J., Urban, J.: ENIGMA: efficient learning-based inference guiding machine. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) *CICM 2017*. LNCS (LNAI), vol. 10383, pp. 292–302. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_20

¹ <https://github.com/ai4reason/eprover/tree/ENIGMA>.

² <https://github.com/ai4reason/atpy>.

5. Schulz, S.: E - a brainiac theorem prover. *AI Commun.* **15**(2–3), 111–126 (2002)
6. Schulz, S.: Simple and efficient clause subsumption with feature vector indexing. In: Bonacina, M.P., Stickel, M.E. (eds.) *Automated Reasoning and Mathematics*. LNCS (LNAI), vol. 7788, pp. 45–67. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36675-8_3
7. Sutcliffe, G.: The 8th IJCAR automated theorem proving system competition - CASC-J8. *AI Commun.* **29**(5), 607–619 (2016). <https://doi.org/10.3233/AIC-160709>
8. Urban, J., Sutcliffe, G., Pudlák, P., Vyskočil, J.: MaLAREa SG1 - machine learner for automated reasoning with semantic guidance. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) *IJCAR 2008*. LNCS (LNAI), vol. 5195, pp. 441–456. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71070-7_37
9. Veroff, R.: Using hints to increase the effectiveness of an automated reasoning program: case studies. *J. Autom. Reason.* **16**(3), 223–239 (1996). <https://doi.org/10.1007/BF00252178>



Formalized Mathematical Content in Lecture Notes on Modelling and Analysis

Michael Junk^(✉), Stefan Hölle, and Sebastian Sahli

Universität Konstanz, Fach D194, 78457 Konstanz, Germany
{michael.junk,stefan.hoelle,sebastian.sahli}@uni-konstanz.de

Abstract. We introduce an approach to include formalized mathematics in regular undergraduate mathematics courses. In order to enable automatic parsing and checking, the \LaTeX code of our lecture notes systematically uses strict syntax conventions for all definitions and theorems, including in particular an unambiguous grammar for all formulas. Resulting from several years of experience, we present a body of about 1000 pages of notes for a course on modelling and a sequence of three courses on analysis and calculus which are a regular part in the undergraduate mathematics curriculum at the University of Constance. We explain the basic structure of the formalization and its syntax and discuss the practicability of the approach.

Keywords: Annotated lecture notes · Formalized mathematics
Mathematical meta language

1 Introduction

The project \MATH as acronym for *meta-language for models, algorithms and theories* [1] originated in an attempt to reduce the difficulties of first-year-students in mathematics. These difficulties which are mainly related to the higher level of rigor compared to school-mathematics, are normally alleviated by exemplary explanations of logic and proof concepts alongside the development of the actual theory. This *implicit* learning-by-doing approach is also adopted in the tutorials, where students learn the rules of mathematics in a trial and error fashion while concentrating on the mathematical content.

In order to make the underlying mechanisms of mathematics more transparent, we tried to formulate and *explicitly* teach the rules in a way that can easily be grasped by beginners.

Our considerations have been used in the redesign of a four-week preparatory course (~ 160 students) which is offered at Constance before the start of the regular lectures. Subsequently, the approach has been adopted in the series of analysis lectures from first to third semester (starting with ~ 300 students). Since 2014, the formalized approach is regularly used in the mandatory lecture

on mathematical modelling which focuses on the formulation and definition of models for real-world-problems (~ 40 students per year). Altogether, the lecture notes of these courses form a collection of about 1000 pdf-pages where the formal parts are structured with specific environments and macros in the corresponding \LaTeX -files (the analysis-series [2] has $\sim 30,000$ lines of \LaTeX).

The practical usage in lectures also led to continuous improvements of the formalization due to the direct and indirect feedback from students in tutorials, examinations or discussions and due to the need to formulate the quite general lecture material concisely and consistently.

2 Examples of Formalizations

An important concept in \MATH are so called *frames* which serve multiple purposes and are organized hierarchically. In general, a frame consists of a list of *names* which point to abstract mathematical objects. Their interrelation is formulated in the frame's *condition*-section by specifying a list of mathematical statements referring to the introduced names and to those of the parent frames. Logical consequences resulting from the assumptions can be formulated and proved in the optional *conclusion*-section of the frame. Defining abbreviations within a frame enriches its name-space and allows to construct associated mappings and records (lists with named components).

In the following subsections, we present examples from the lecture series [2] which covers real analysis in its first part, starting from an axiomatic introduction of the real numbers, followed by multivariate calculus in the second part, including concepts like normed-, metric- and topological spaces. The third part is devoted to ordinary differential equations and to abstract measure theory.

2.1 Conditions

About 31% of the 450 names defined in the analysis course refer to frames without conclusion section which serve as notions like *sequence*, *convergent*, or *differentiable*. To fix ideas we consider the simple notion

$$\text{nonNegative} := x \mathbf{with} x \geq 0 \square$$

Here the keyword **with** separates the name x from the frame-condition which ends at the \square symbol. The infix-expression $x \geq 0$ is used as abbreviation for $\text{greaterOrEqual}(x, 0)$ which requires its arguments to be elements of \mathbb{R} . By type-inference, the condition $x \in \mathbb{R}$ is therefore implicitly contained in $x \geq 0$ and need not be listed in the frame condition. Within the \LaTeX -code the definition would look like this

```
\begin{mtext}
nonNegative $:= x$ \mwith $x\geq0$ \sq
\end{mtext}
```

To express the statement that the object $1 \in \mathbb{R}$ satisfies the frame-condition, we write $1 : \text{nonNegative}$ which is casually pronounced as *1 is non-negative*. It means that all frame conditions are satisfied once the name x is replaced by 1 at all occurrences.

2.2 Mappings

The largest portion (39%) of names is allotted to number-, set-, or function valued mappings like *limit* for the limit of a sequence, *union* for the union of two sets, or *derivative* for the derivative of a function. As example we consider the function valued mapping

$$\text{monomial}(n \text{ with } n \in \mathbb{N}_0) := x \text{ with } x \in \mathbb{R} \mapsto x^n$$

which shows two possible forms of mapping definitions at once: In the nameless form, the parameter condition and the resulting expression are separated by the \mapsto symbol while in its named form, the parameter condition is provided in a bracket behind the name and the resulting expression after the $:=$ symbol. Evaluations are denoted with the usual bracket notation, i.e. $\text{monomial}(2)(5) = 25$ would be true. The L^AT_EX-code is

```
\begin{mtext}
monomial($n$ \mwith $n\in\mathbb{N}_0$) $:= x$ \mwith $x\in\mathbb{R}$ \mapsto $x^n$
\end{mtext}
```

If an abbreviation is introduced for the evaluation of a mapping, the declaration is currently given in normal text mode right after the definition of the mapping. Also the parsing of more intricate abbreviations would be explained here (e.g. $\int_{-1}^1 a \cdot x^2 dx$ is transformed into $\text{definiteIntegral}(x \mapsto a \cdot x^2, -1, 1)$).

2.3 Parametric Conditions

Notions depending on parameters, like *zero of a function* or *neighborhood of a point* make up 27% of the defined names. They are realized as frame-valued mappings. As example, we consider

$$\text{root}(x \text{ with } x \in \mathbb{R}) := r \text{ with } r \cdot r = x \quad \square$$

Here, $(-2) : \text{root}(4)$ holds which would be pronounced as *-2 is a root of 4*.

2.4 Theorems

In the lecture notes, all lemmas and theorems (~770) are denoted with the forall-quantifier followed by the premise (a condition) which is separated from the conclusions by the keyword **hold** and \square as end mark. For example,

$$\forall M \text{ with } M \subset X \text{ hold } \text{int}(M) : \text{open}; (\partial M) : \text{closed} \quad \square$$

Since the majority of theorems has a single statement as conclusion, also the following syntax is used: $\forall x \text{ with } x \in \mathbb{R} \text{ holds } x^2 : \text{nonNegative}$

```

\begin{mtext}
$\forall x$ \mwith $x \in \mathbb{R}$ \mholds $x^2$ \nonNegative
\end{mtext}

```

2.5 Proofs

Proofs in the lecture notes (~530) are not presented in a completely formal style because reading such texts would be quite straining. Nevertheless, they are carried out with a later formalization in mind. In particular, for each statement type, proof steps are introduced for derivation and usage, respectively. Especially in the proofs of the preparatory course and the initial parts of Analysis 1, extensive referencing to these steps is typical. In the L^AT_EX-code, proofs are always embedded in the `shaded`-environment of the `framed`-package, for example

$$\forall \mathcal{M}, M \text{ with } \mathcal{M} : \text{setFamily}; M \in \mathcal{M} \text{ holds } M \subset \bigcup \mathcal{M};$$

In a direct proof we assume that \mathcal{M}, M are given and that $\mathcal{M} : \text{setFamily}$ and $M \in \mathcal{M}$ hold. In a direct proof of $M \subset \bigcup \mathcal{M}$ we further assume that x is given with $x \in M$. Hence $M : (U \text{ with } U \in \mathcal{M}; x \in U \square)$ can be compressed, so that $\exists U \text{ with } U \in \mathcal{M}; x \in U \square$ holds. Application of (S36) to (\mathcal{M}, x) yields an equality which allows us to show $x \in \bigcup \mathcal{M}$ by replacement ■ Application of (S24) to $(M, \bigcup \mathcal{M})$ now gives the claimed result ■

2.6 Spaces, Structures and Models

Conditions that give rise to several results and additional notions are formulated as full-fledged frames. This covers usual algebraic structures like *group* or *field*, analytic structures like *Banach*- or *Hilbert*-space, all models in the modelling lecture but also the whole Analysis 1 content when viewed from Analysis 2. In the following example, we can see how the keyword **conclusions** separates the condition block from the results.

```

metricSpace := (X, d) with
  X : set; d : X × X → ℝ;
  ∀x, y, z with x ∈ X; y ∈ X; z ∈ X hold
    d(x, y) = 0 ⇔ x = y; d(x, y) = d(y, x); d(x, y) ≤ d(x, z) + d(z, y) □
conclusions
  B(x, r with x ∈ X; r > 0) := {y with d(y, x) < r};
  innerPoint(M with M ⊂ X) := x with ∃r with B(x, r) ⊂ M □ □;
  open := M with ∀x with x ∈ M holds x : innerPoint(M) □;
  ...
□

```

If d is defined as $(x, y) \mapsto |x - y|$, the pair (\mathbb{R}, d) satisfies the frame-condition so that $M := \text{metricSpace}(\mathbb{R}, d)$ denotes a record with field names corresponding to the local names of the frame. In particular, $K := M.B(0, 1)$ would be the unit ball of the metric centered at $0 \in \mathbb{R}$ and $K : M.\text{open}$ would be true.

In the lecture notes, the conclusion section of a frame typically coincides with a subsection of the L^AT_EX-document because introducing local concepts and their relationships requires substantial motivation and explanation which interrupts the formal definition of the frame.

As far as the L^AT_EX-code is concerned, indenting and formatting is limited to the `mtab`-macro and double slash for new-line in order to alleviate automatic parsing. For example, the above definition would start like

```
\begin{mtext}
metricSpace $:= (X,d)$ \mwith\\
\mtab $X:$ set; $d:X\times X\to\R$;\\
...
```

3 Conclusions

Using the M^AT_H-syntax for definitions and theorems consistently throughout the lecture notes, we obtain human readable documents that deliberately stress the formal mathematical structure and which allow automatic parsing of the mathematical content. Since emphasising the formal interrelation of mathematical structures, notions and theorems was our goal, there was no need to hide formal structure from the reader so that hardly any extra cost during writing appeared. However, restructuring the classical material in a formally consistent way was not without effort.

Although the primary goal in the development of M^AT_H was *not* the design of a specific digital representation of mathematical content or a new proof assistant, we needed ad hoc solutions for presenting the formal content and for checking the consistency of our language rules which led to the L^AT_EX-approach and to the implementation of a prototypical M^AT_H-interpreter which is currently being tested (see [2] for the ANTLR-grammar files and some code snippets). Ultimately, the idea is to obtain interpretable code from the L^AT_EX-source by focusing on the `mtext` blocks and stripping the `$` and other L^AT_EX-formatting symbols.

To fully accomplish this task, the textual descriptions of abbreviations for prefix forms of mappings should be replaced by more formal ones (e.g. using specialized L^AT_EX-macros as in s_LE_X [3]). This would increase the workload during writing only slightly.

Certainly a more drastic increase would result from providing formal proofs invisibly (or hideably in a hypertext version) alongside the current vernacular proofs. While we do introduce proof steps similar to those in proof assistants (like Mizar [4]) we do not ask students to write down proofs completely formal. In particular, we are satisfied when students indicate in their proofs that they use

specific steps and that the conditions for applying the steps are satisfied. Nevertheless, the extra information contained in formal proofs could be of interest when the presented vernacular form is too sketchy for the students to appraise all the details. In this most elaborate form, the extraction of $\mathbb{M}\text{ATH}$ -code with subsequent checking would be possible in the sense of literate programming [5].

Alternatively, a conversion of the formalized material to content-based representation formats like OMDoc [6] would generate an interface to a variety of other applications (the obvious connection between OMDoc -theories and $\mathbb{M}\text{ATH}$ -frames would be helpful here). This conversion could be accomplished, for example, with an intermediate formulation in terms of $\text{s}\text{T}\text{E}\text{X}$ [3].

Our practical experience with $\mathbb{M}\text{ATH}$ -grammar in undergraduate lectures is generally positive. In the preparatory course, time is abundant and the material is easily conveyed. Nevertheless, students do have problems with observing the presented rules of mathematics (which was the reason to start the project). Here, a definite advantage is that difficulties with the rules can be much better addressed once they are made explicit. During the semester, the standard curriculum leads to much higher pressure and students tend to favor purely intuitive understanding without fallback to formal procedures (although this is a poor strategy when concepts are becoming more abstract). However, an attempt to switch to the usual vernacular form of definitions and theorems at the end of the first semester was not approved by the majority of the students. By that time, they appreciated the formal way of writing as being unambiguous and easy to use.

References

1. $\mathbb{M}\text{ATH}$ -Homepage: Currently in German: <http://www.math.uni-konstanz.de/mmath>
2. LaTeX-Sources: LATEX and PDF versions of the analysis lecture notes (in German). <http://www.math.uni-konstanz.de/mmath/download>
3. Kohlhase, M.: Using LaTeX as a semantic markup format. *Math. Comput. Sci.* **2**, 279–304 (2008)
4. Mizar-Homepage. <http://www.mizar.org/>
5. Knuth, D.E.: Literate Programming. *Comput. J.* **27**(2), 97–111 (1984)
6. Kohlhase, M.: OMDoc - An Open Markup Format for Mathematical Documents [version 1.2]. Springer, Heidelberg (2006). <https://doi.org/10.1007/11826095>



Isabelle Import Infrastructure for the Mizar Mathematical Library

Cezary Kaliszyk¹  and Karol Pąk² 

¹ Universität Innsbruck, Innsbruck, Austria
cezary.kaliszyk@uibk.ac.at

² Uniwersytet w Białymstoku, Białystok, Poland
pakkarol@uwb.edu.pl

Abstract. We present an infrastructure that allows importing an initial part of the Mizar Mathematical Library into the Isabelle/Mizar object logic. For this, we first combine the syntactic information provided by the Mizar parser with the syntactic one originating from the Mizar verifier. The proof outlines are then imported by an Isabelle package, that translates particular Mizar directives to appropriate Isabelle meta-logic constructions. This includes processing of definitions, notations, typing information, and the actual theorem statements, so far without proofs. To show that the imported 100 articles give rise to a usable Isabelle environment, we use the environment to formalize proofs in the Isabelle/Mizar environment using the imported types and their properties.

1 Introduction

The Mizar project [10] has developed a language allowing users to write formal mathematics similar to the way it is done in informal mathematical practice [3]. This convenience for the users however means that the Mizar system is the only tool able to parse the language. Many exports of the *Mizar Mathematical Library* (MML) [1] to various formats and systems have been developed, including variants of XML [21, 24], TPTP [6], and OMDoc [12], however they are all static. They allow inspecting the information contained in the Mizar library, presenting it [26], searching, and even give proof hints [20], but do not allow any further development of the proof scripts.

One of the reasons for this state-of-art is the architecture of the Mizar verifier. The verification of a Mizar proof script, called an *article* is performed by a series of independent programs, that each check particular aspects of syntactic and semantic correctness. Modules include derived information in the intermediate representation and drop the data which the Checker will not need to certify the proof. This final format, which is used by most of the exports, no longer contains the original user input and it is usually not possible to completely restore it.

The paper has been supported by the resources of the Polish National Science Center granted by decision n^oDEC-2015/19/D/ST6/01473.

Such lost information together with the processed script is necessary in our project [17] aiming to certify the Mizar proofs in Isabelle and create an environment to further develop the Mizar Library in the Isabelle logical framework [29]. For this, we develop an application able to combine the syntactic and semantic information available at the various stages of the Mizar processing. Such processed information is suitable for importing it in Isabelle or other frameworks, it could for example serve as a basis for a more complete import into OMDoc, that would include user steps and notations.

We develop an Isabelle infrastructure able to import the processed Mizar information into the Isabelle/Mizar object logic. Definitions of Mizar functions, types, predicates, etc. are automatically processed giving rise to Isabelle/Mizar constants together with their respective properties. Type inference information, contained in Mizar registrations and clusters is transformed to Isabelle theorems, that are furthermore put into special theorem lists that can be used by Isabelle/Mizar type inference. Actual theorem (as well as lemma and theorem scheme) statements give rise to corresponding statements in Isabelle with Mizar notations transformed into similar Isabelle MixFix notations [28].

Contributions. This paper introduces an infrastructure for importing an initial part of the Mizar Mathematical Library into the Isabelle/Mizar object logic. The particular contributions are:

- We create a combined syntactic-semantic export of the Mizar proof data. As part of this we propose ways to match the processed semantic information with the original proof script.
- We develop an Isabelle infrastructure able to process and import the first 100 MML articles (out of 1318) into Isabelle/Mizar, so far mostly without proofs. The transferred values are loaded into the LCF environment in the spirit of Isabelle/Import [14].
- The imported environment allows users to work with Mizar data inside Isabelle, with features including notations and type inference. We demonstrate this by formalizing proofs depending on the imported parts of the library, namely the beginning of the Mizar article NEWTON which defines factorials and exponents and shows their basic properties.

Contents. In Sect. 2 we discuss existing Mizar exports. Section 3 introduces the Isabelle/Mizar object logic. In Sect. 4 we discuss the Isabelle import infrastructure. Section 6 discusses combining the syntactic information. As this is mostly technical and not necessary to understand the previous sections, we chose to present it at this point. Finally in Sect. 5 we show an example formalization that uses the imported definitions, theorems, and Mizar type inference rules.

2 Existing Mizar Exports

This section introduces the process how Mizar processes a formalization, and discusses the existing exports from Mizar which use various parts of this process.

The Mizar system processes each formalization in multiple stages. The stages are independent processes, which communicate using external files. There are more than 20 types of such files. This large number allows different stages and data exports to import only minimal information, which was crucial given the memory limits 40 years ago, when Mizar was conceived. In 2005, Urban [24] has modified the representation used in the intermediate stages to a slightly more accessible XML format. This has been done relatively late, as the larger representation does slow down the verification of the MML 1.67 times, but at the same time it allowed external access of some of the Mizar information.

```

----- tarski.xml -----
<Proposition line="27" col="35">
  <For pid="0" vid="2">
    <Typ kind="M" nr="1" pid="1"><Cluster/><Cluster/></Typ>
    <ls>
      <Var nr="1"/>
      <Typ kind="M" nr="2" pid="2"><Cluster/><Cluster/></Typ>
    </ls>
  </For>
</Proposition>
----- tarski.idx -----
<Symbol kind="I" nr="2" name="x"/>
----- tarski.eno -----
<Pattern kind="M" nr="1" aid="HIDDEN" formatnr="2" constrkind="M"
  constrnr="1" relnr="1">
----- tarski.frm -----
<Format kind="M" nr="2" symbolnr="2" argnr="0"/>
----- tarski.dcx -----
<Symbol kind="M" nr="2" name="object"/>

```

Fig. 1. The statement of the theorem “Everything is a set” in Mizar XML together with the information encoded in four other tarski XML files. These are necessary to decode the information. In the statement, the `<For pid="0" vid="2">` binds the second occurrence of the symbol `x`, assigning it the type `object`, which corresponds to `pid="1"`. Similarly the `<Is>` node encodes the information that the variable bound by the first `x` quantifier is of the type set `pid="2"`.

The information processed by the first stages of Mizar is encoded in the Mizar XML (an example statement presented in Fig. 1), with some of the original syntactic information lost. All formulas are transformed to the Mizar normal form (MNF), which uses only selected logical connectives (\wedge , \neg , \top , and \forall) [4]. Furthermore, Mizar XML also uniquely identifies the constructors corresponding to particular notations, fixes their arities, and expands some abbreviations. For example the Mizar type `Function of A,B` is exported as `quasi_total Function-like Relation-like Element of bool [:A,B:]` in the XML representation.

Later versions of Mizar XML included `pid` attributes containing information about the notation and normalization, which allowed the generation of HTML and a system for semantic browsing of the Mizar Library [4]. This information is passed by the ANALYZER to the CHECKER, however the latter ignores it.

The notations are re-constructed based on this information heuristically, however in some cases it is not possible to guess the original representation.¹ Such small inconsistencies do not have a significant influence on the rendered HTML.

Urban has also developed the *Mizar Problems for Theorem Proving* (MPTP) export [25]. It is also based on the semantic XML representation. It aims to provide problems for automated reasoning, cross-verification [27] of Mizar, and more recently exploited for machine learning for theorem proving [13], and lemma extraction [19]. The export uses an XSLT stylesheet to make all problems independent from the Mizar environment, which allows consistent naming of symbols across the whole corpus. The core of Mizar is based on first-order logic, however the theorem and axiom schemes can also be exported to higher-order TPTP problems based on the same information [6].

The MPTP translation closely represents the first-order top-level statements of each article. Mizar types are represented as predicates. Definition blocks are unfolded to separate axioms stating the return type, the parent type, and the property of the introduced concepts. Similarly Mizar registrations (user-defined type inference rules) become axioms about the types. Second-order objects cannot be directly represented in the first-order format, therefore a fresh constant, together with an axiomatization, is introduced for each instance. This is theoretically justified by the set theoretic axioms of replacement and comprehension. This has been sufficient to translate the MML.

The MPTP representation was also used to import Mizar into OMDoc [12]. This means that the representation is semantically correct, but cannot reliably represent the original syntax. In particular many notations and variable names have been lost. Still this is enough to browse and search the Mizar library together with other theorem proving libraries.

Rather than extracting the information from the XML representation processed by the CHECKER, it is also possible to use the intermediate output of the parser. This approach has been used by *Weakly Strict Mizar* (WSX) [21] and *More Strict Mizar* (MSX) [7] which export subsequent syntactic layers. The aim of WSX (presented in Fig. 2) is to provide a standalone parser for Mizar articles, that can be used by external tools. MSX is an extension of WSX, where the representation is additionally augmented by variable information: variables are categorized as reserved, free, bound, or local constant together with unique identifiers and implicit quantifiers are made explicit. Both syntactic exports have complete information about the user input, but do not include any derived information. In particular the disambiguation and hidden arguments are missing, and are very hard to reconstruct for any external tool.

¹ These rarely affect the HTMLization of the current Mizar library, see for example http://mizar.uwb.edu.pl/version/current/html/funct_2.html#FC4 where the expression $K3(\underline{g}, f)$ includes $K3$ rather than $f * \underline{g}$. Note the reverse order of arguments.

```

<Proposition>
<Label idnr="0" spelling="" line="27" col="5"/>
<Universal-Quantifier-Formula line="27" col="5">
  <Explicitly-Qualified-Segment line="27" col="5">
    <Variables>
      <Variable idnr="2" spelling="x" line="27" col="7"/>
    </Variables>
    <Standard-Type nr="2" spelling="object" line="27" col="20"/>
  </Explicitly-Qualified-Segment>
  <Qualifying-Formula line="27" col="35">
    <Simple-Term idnr="2" spelling="x" line="27" col="28"/>
    <Standard-Type nr="1" spelling="set" line="27" col="35"/>
  </Qualifying-Formula>
</Universal-Quantifier-Formula>
</Proposition>

```

Fig. 2. The statement of the theorem “Everything is a set” in Weakly Strict Mizar. WSX includes all user input, exactly as it was typed by the user and parsed in a tree form. The proposition without a label states that the explicitly universally bound variable x is of type set.

3 Isabelle and the Mizar Object Logic

Isabelle [29] is a logical framework, that is a proof assistant designed for the purpose of defining logics and working in these logics. The foundations of Isabelle are a variant of simple type theory with ML-style polymorphism. Isabelle’s implementation is based on a relatively small kernel implemented in Standard ML. It includes functionality that makes it convenient to define constants present in logics, basic inference rules, notations, and the procedures specific to particular logics.

In our previous work we formally defined the semantics of Mizar as an Isabelle object logic [18]. This included mechanisms that allowed for the definition and use of Mizar types, introduction of meta-level constants and predicates, and the use of Mizar quantifiers. The notations of the resulting Isabelle/Mizar object logic have been optimized [16] allowing a combination of Mizar definitions with Isabelle-style ones, some hidden arguments, re-definitions, as well as extended constructions present in the Mizar library such as set comprehensions and structures. We finally defined a type inference mechanism for Isabelle/Mizar [15]. This Isabelle/Mizar object logic, serves as the basis for the imported Mizar library.

4 Isabelle Import Infrastructure

In this section we discuss the import of an already pre-processed Mizar library article. An outline of the procedure preparing an article for import will be discussed in Sect. 6.

Isabelle already includes a package that allows importing proofs originating from HOL Light and HOL4 [14, 22] as well as one for importing the OpenTheory library articles [11]. Both of these are restricted to higher-order logic and only include higher-order logic kernel specific basic inference steps. The Mizar articles do not include any basic inference steps, but rather specify the different kind of Mizar language items: Mizar style definitions, actual proved statements or user proved type inference rules. The existing package are therefore insufficient for our purposes, in fact none of the Isabelle HOL/Import rules could be re-used.

We therefore propose a new format to represent Mizar article to import, which will consist of (possibly nested) sequences of commands. In practice, we will represent the article in XML with different tags corresponding to the possible Mizar proof script items.

The architecture of the Mizar import will be influenced by the Isabelle community's experience with HOL/Import. The first version of HOL/Import [22] attempted to syntactically create Isabelle theory files corresponding to the imported articles. It was soon realized, that the input syntax of Isabelle changes significantly with successive versions and it was too much work to keep the output of HOL/Import up to date with it. Therefore, the newer version of HOL/Import does not attempt to generate an Isabelle theory file source, but rather processes the values directly, giving rise to an environment with the definitions and theorems pre-loaded, together with documentation that a user can use instead of a theory file, to further develop the formalizations. This approach came to be a viable one, and did not need significant updates since its creation. In our current work we imitate the latter approach.

A common part shared by the import of the various Mizar item types are Mizar propositions, terms, and types all of which will be imported as Isabelle terms. To simplify this process, the term representation exported from Mizar will include all the necessary information to represent the knowledge in a logical framework. All variables are guaranteed to be quantified and include their meta-types (i.e., a Mizar type variable or a term variable or a propositional variable), as well as the meta-types of their arguments in case of higher-order variables (scheme variables). Applications and abstractions directly correspond to the Isabelle application and abstraction (abstraction is present in Mizar in case of constructions such as quantifiers, set comprehensions, or the choice operator). The seven constants corresponding to the basic predicate logic and five constant corresponding to the foundations of Mizar are mapped explicitly to their Isabelle counterparts. Finally, we also map manually the five constants corresponding to the axiomatic Mizar article `HIDDEN`. All subsequent constants will be defined by the imported Mizar articles.

Mizar definitions need to be presented in a format that can be accepted by Isabelle. This means equations, where left-hand side consists of the newly introduced concept possibly applied to some arguments. To fit the various kinds of Mizar definitions (including definitions of types, functions, predicates, and structures, conditional definitions, definitions by means, synonyms, and antonyms) in this format, we create a local theory context with assumptions, in that context

define the constant, and apply definitional theorems. Such theorems give rise to definition correctness obligations (such as for example existence and uniqueness for functions defined by means). The proof obligations are currently assumed – this will be discussed in future work. Finally, suitable derived theorems are declared and exported to the global theory and attributes (such as its use in the type inference mechanism) are applied.

Actual theorems and lemmas are straightforward to process, as we do not import the proofs yet. The statement is fully transformed to an Isabelle corresponding statement. There are two special cases of theorems. Mizar type inference rules (referred to as clusters in Mizar) need to be further processed and added to specific type inference lists. Mizar schemes give rise to higher-order theorems, therefore Mizar variable declarations need to be transformed to Isabelle assumptions.

The general methods for translating definitions will be discussed in Sect. 6.3. Here we will only give an example of a definition that provides two types of information, the meaning of the defined object `ordinal2_def_10` and the Mizar type inference rule `ordinal2_def_ty`.

```

definition
  let fi be Ordinal-Sequence;
  given A such that
    A is_limes_of fi;
  func lim fi → Ordinal means
    it is_limes_of fi;
end;

thm ordinal2_def_10
  fi is Ordinal-Sequence $_{ORDINAL2M1}$   $\implies$ 
   $\exists A : \text{Ordinal}_{ORDINAL1M3}. A \text{ is\_limes\_of } fi$ 
   $\implies \text{lim } fi \text{ is\_limes\_of } fi$ 
thm ordinal2_def_10_ty
  fi is Ordinal-Sequence $_{ORDINAL2M1}$   $\implies$ 
  lim fi is Ordinal $_{ORDINAL1M3}$ 

```

5 Mizar-Style Proof Development in Isabelle

In this section we present a case study, which shows the usability of the imported part of the Mizar library in Isabelle. For this, we manually re-formalize selected parts (1 reduction, 3 clusters, 6 theorems, and the definition proof obligations) of the Mizar article `NEWTON`, which defines powers and factorials.

The article is not among the first 100 imported articles, and our export infrastructure is not yet able to interpret the Mizar environments in order to process it automatically. The `NEWTON` article depends (directly or indirectly) on 80 Mizar articles. All these are imported automatically, therefore the current formalization relies both on Mizar and on Isabelle when it comes to proof checking. Importing the proofs will allow reducing the trust base to a single system.

In the paper we only present the statements, all the theorems have proofs in the formalization, and so does the definition, as one needs to prove that the result of the function is of the declared type. For correspondence, Isabelle theorem and constant names include their absolute MML addresses [25], we however consider using more canonical Isabelle-like names [2]. We first define the binary power function, Mizar version presented on the left for comparison:

<pre> definition let x be Complex, n be natural Number; func x ^ n → number equals Product (n → x); </pre>	<pre> mdef newton_def.1 (_ - [90,0]91) where mlet x is Complex, n is natural Number func xⁿ → number equals Π (n → x) </pre>
---	---

where $n \mapsto x$ denotes a constant sequence of length n , equal x everywhere.

Next, we prove a number of properties of the defined power operator. The proofs make use of various concepts (including natural numbers, complex addition, finite products) from the imported articles. Furthermore, the declared user type inference rules imported from previous articles are automatically used by the Isabelle/Mizar type inference, which automatically handles a number of reasonings about the soft types.

<pre> mtheorem newton_th.4: z⁰ = 1 mtheorem newton_th.6: z^{s + N 1} = z^s *_C z mtheorem newton_th.8: x^{s + C t} = x^s *_C x^t </pre>	<pre> mtheorem newton.th.5: z¹ = z mtheorem newton.th.7: (x *_C y)^s = x^s *_C y^s mtheorem newton.th.9: (x^s)^t = x^(s *_C t) </pre>
--	--

An example of the use of an imported statement is the induction on natural numbers. It is used to show the user-level typing rule, that the power of a natural number is also a natural number.

<pre> registration let x, n be natural Number; cluster x ^ n → natural; coherence proof A0: n is Nat by TARSKI:1; defpred P[Nat] means x ^ \$1 is natural; A1: for a being Nat st P[a] holds P[a+1] proof let a be Nat; assume P[a]; then reconsider b = x ^ a as Nat; hence [ty]:x^a is natural Number by mauto x ^ (a+1) = b*x by Th6; hence thesis; end; A2: P[0] by RVSUM_1:94; for a being Nat holds P[a] from NAT_1:sch 2 (A2, A1); hence thesis by A0; end; end; </pre>	<pre> mtheorem mlet x is natural Number, n is natural Number cluster xⁿ → natural proof have A0:n is Nat using tarski.th.1 by simp let ?P = λit. x^{it} is natural have A1: for a be Nat st ?P(a) holds ?P(a + N 1) proof(rule ballI,rule impl) fix a assume [ty]:a be Nat and ?P(a) have x^{a + N 1} = x^a *_C x using newton.th.6[of a] by mauto thus x^{a + N 1} is natural by mauto qed mauto have A2:?P(0) using newton.th.4 by mauto have for a be Nat holds ?P(a) using nat_1.sch.2[of ?P] A2 A1 by simp thus xⁿ is natural using A0 by simp qed </pre>
--	--

The Isabelle/Isar environment with Mizar proofs imported is already usable, albeit the level of automation is still weaker than that of Mizar. Some of the Mizar abbreviations have been introduced as definitions, and are therefore not automatically unfolded. By specifying our own notations and additional registrations, the environment becomes well usable, where even some of the complicated proofs are of similar length to those of Mizar. It is however currently slower, especially when it comes to selecting the background information: in the above proof at some reasoning steps 139 typing judgements are derived automatically and stored in an Isabelle theorem list. This is possibly due to the highly optimized Mizar implementation, which uses arrays for all indexing and contrary to Isabelle does not need to rely on bi-resolution.

6 Combining the Syntactic and Semantic Representations of Mizar

In order to re-verify the Mizar proofs, we would like to export the individual proof steps faithfully. The semantic Mizar XML introduced in Sect. 2 is insufficient for many cases. In particular the term abbreviations, reservations, and hidden quantifiers, which are already supported by Isabelle/Mizar, are not preserved in Mizar XML, and a modification of the Mizar CHECKER to preserve such information would be a tremendous task, as the information would need to be correctly processed by all parts of the large Mizar kernel. A simple example of two formulas with *exactly* the same representation in XML is: $\alpha \rightarrow (\beta \rightarrow \gamma)$ and $(\alpha \wedge \beta) \rightarrow \gamma$. Even with the Mizar XML hints, the two are identical. This is a problem in Isabelle, as there the proof skeleton must precisely correspond to the proof [30]. A further discussion of combining the proof steps is provided in [23].

6.1 Procedure Overview

The combination of the syntactic and semantic Mizar processing data consists of two parts. First, we will match all the items from these both representations filling the missing information, using a process similar to that of the Mizar ANALYSER. Then we will ensure that the combined information corresponds to our Isabelle meta-model of Mizar [15], matching the constants and lists of arguments. In the following we present the process in more detail.

First, we match single items (both top-level items and individual proof items). Then we perform a full identification of all objects present in the item. Constructors for all ambiguous terms are determined, and argument lists are expanded by the hidden arguments computed by the Mizar ANALYSER. To match this with the transformed syntactic form, we modify the latter imitating some of the ANALYSER processes, including logical formula normalization, simplification of selected constructions (such as predicate and attribute negations). This also needs to identify the types of variables bound by the quantifiers and matching atomic propositions, and further their term and subterm arguments recursively. Additionally Mizar local abbreviations (Mizar syntax: `set`), which have been

```

<proposition label="tarski_th_1">
  <app>
    <logic id="Ball" type="o" args="2" argsType="ty_abs"/>
    <const id="HIDDENM1" type="ty" args="0" argsType="set"/>
    <abs id="x" type="set" args="0">
      <app>
        <reservedconst id="IsType" type="o" args="2" argsType="set_ty"/>
        <var id="x" type="set" args="0" argsType="set"/>
        <const id="HIDDENM2" type="ty" args="0" argsType="set"/>
      </app>
    </abs>
  </app>
</proposition>

```

Fig. 3. The formulation of the Mizar theorem “Everything is a set” with the combined syntactic and semantic representation. All terms are expressed using applications and abstractions of a meta-logic allowing an easy correspondence to the meta-model of Mizar, in particular making it easy to express in a logical framework.

fully unfolded in the XML format, can be identified and folded, which is useful to improve the legibility of proof texts. All Mizar constructions can be presented as applications of meta-constants, which is unique and allows a uniform way of processing the information, as was needed in the previous two sections. An example of the combined information is presented in Fig. 3.

6.2 Background Information

The processing of a Mizar article requires a precisely defined environment. In order to allow external processing, selected background information needs to be transformed and exported. In Mizar these are of three kinds: `clusters` which aid the computation of types in the presence of attributes, `reductions` which denote rewrite rules, and `identify`, which resolves conflicts that arise when identifying objects based on their arguments. Of course all of these are semantically theorems, and could be exported as such (as is done for example by MPTP [25]). Our import would however include its own type inference mechanisms, so we can export exactly the information Mizar has. In particular, we include constants corresponding to each kind of background information (the Isabelle/Mizar constructions are given in [16]). A limited use of this information was shown in Sect. 5.

An example exported cluster and presented in Isabelle is:

<pre> registration let f be one-to-one Function; cluster f" → one-to-one; </pre>	<pre> thm funct_1_cl.10 f is one-to-one \implies f is Function $FUNCT_1M1 \implies$ (f⁻¹) is one-to-one </pre>
--	---

which states that the inverse of a one-to-one function is also one-to-one.

6.3 Definitions

We export each individual definition separately (rather than using Mizar definition blocks) including all the available information for each definition. The data that is again distributed between the different semantic processed files and needs matching with the syntactic part to reconstruct the missing information. For example a redefinition of a Mizar function, whose type has been expanded, and the modification only concerns the right-hand side of the definition, the folded type is only present in the correctness condition.

All kinds of defined objects (meta level functions, Mizar types and predicates) require different information, however they share multiple common parts: a unique binding allowing referring to the definition body (such as `newton_def_1`), a unique constant definition, a notation, and the types of all the arguments of the defined object including their dependent types (Mizar allows term arguments in types), as well as the assumptions for conditional definitions. To preserve the Mizar semantics and at the same time allow for uniformity, Mizar attribute definitions need to have their last assumption transformed to an argument. For example the definition of an empty set uses `X` as an argument rather than an assumption to reflect the meta semantics:

```

definition let X be set ;
  attr X is empty means
    not ex x st x in X ;
  mdef xboole_0.def.1 (empty) where
    attr empty for set means
      ( $\lambda X. \neg (\exists x : \text{object. } x \text{ in } X)$ )

```

The definition body can usually be expressed directly, however many Mizar constructs allow definitions per cases. For those, specific abstractions need to be introduced, for example the (somewhat non-standard) definition of an element of a set in Mizar is formulated as follows, while its export includes all the information necessary to transform the cases to multiple implications:

```

definition
  let X be set ;
  mode Element of X  $\rightarrow$  set means
    it in X if X is non empty
      otherwise it is empty ;
  mdef subset_1.def.1 (Element _) where
    mlet X is set
    mode Element X  $\rightarrow$  set means
      ( $\lambda \text{it. it in } TARSKIR2 \text{ } X \text{ if } X \text{ is non empty}$ 
        otherwise  $\lambda \text{it. it is empty}$ )

```

The final exported information from definitions are the properties of the defined concepts, which can be introduced in Mizar definition blocks. Similar to registrations, they can be expressed as theorems. Mizar does not include a formulation of the properties, and we recover this meaning along with the syntax in the exported environment. The reason for this in Mizar, is that the concept cannot be used in the definition block. Therefore we recover the folded form of each property:

<p>definition let X, Y be set; func $X \setminus Y \rightarrow \text{set}$ means for x holds x in it iff x in X or x in Y; existence... uniqueness... commutativity; idempotence;</p>	<p>thm XBOOLE_0_def_3 $X \text{ is set} \implies Y \text{ is set} \implies$ $x \text{ in } X \cup Y \iff x \text{ in } X \vee x \text{ in } Y$</p> <p>thm XBOOLE_0K2_commutativity $\forall X : \text{set}. \forall Y : \text{set}. X \cup Y = Y \cup X$</p> <p>thm XBOOLE_0K2_idempotence $\forall Y : \text{set}. Y = Y \cup Y$</p>
---	--

6.4 Redefinitions

Mizar redefinitions allow expanding and clarifying the information about already existing objects. It is possible to modify any kind of object (function, predicate, or Mizar type). There are four main categories of redefinitions, and we will need to adapt the export process for each of the below categories:

1. Adding a notation to an object. The name and the visible arguments need to be preserved, however the new notation can include more hidden arguments.
2. Changing the definition body of an object. This is particularly important, as Mizar proofs must correspond to definitions, therefore showing the equivalence of two definitions allows for different proof obligations when the definition is expanded.
3. Making the type more precise. The result types of functions can be refined, as well as the mother type can be refined for Mizar types. For example the exponential function could be defined with range type being the type of real numbers. Later, the user can show that the result is also non-negative and inform the type system about it.
4. Adding selected properties to defined objects.

Each redefinition introduces a notation, equivalent to the original constant, with the additional arguments removed. Each modification of the definition body is formulated as a theorem. These can later be processed in Isabelle by modifying the default introduction and elimination rules, just like it is done for regular definitions.

<p>definition let X, Y be set; redefine pred $X = Y$ means $X \subseteq Y \ \& \ Y \subseteq X$;</p>	<p>abbreviation $X =_{XBOOLE_0R4} Y \equiv X = Y$</p> <p>thm xboole_0_def_10 $X \text{ is set} \implies Y \text{ is set} \implies$ $X =_{XBOOLE_0R4} Y \iff X \subseteq Y \wedge Y \subseteq X$</p>
--	---

Proving that the result of a function has a more precise type is semantically close to a Mizar cluster (the two however require different syntax in Mizar). It can of course be specified as a theorem, but we want to preserve both the syntax and the semantics in our export, therefore we annotate the syntax and verify

that the semantics are correct already in the imported setting and that such rules can be used by the type inference mechanism:

<pre> definition let p be FinSequence; redefine func dom p → Subset of NAT; </pre>	<pre> thm finseq_1_add_3 p is FinSequence <i>FINSEQ_1M1</i> ⇒ dom <i>RELAT_1K1</i> p is Subset <i>SUBSET_1M2</i> NAT <i>NUMBERSK1</i> </pre>
--	---

Redefinitions that include properties, can be exported in a similar way to properties specified for original definitions. They give rise to uniquely identified theorems about the exported constants and their property names:

<pre> definition let X, Y be non empty set; redefine pred X misses Y; irreflexivity; </pre>	<pre> abbreviation X misses <i>SUBSET_1R1</i> Y ≡ X misses Y thm SUBSET_1R1_irreflexivity ∀ Y : non empty set. ¬ Y misses <i>SUBSET_1R1</i> Y </pre>
---	---

The Mizar language is very rich when it comes to notations. Notations are used for example for predicate synonyms and antonyms, function synonyms, or type abbreviations. Adding notations is very cheap, since all components of Mizar reason modulo a congruence closure algorithm. Rather than clarify these in the export, we imitate the Mizar syntax and semantics, by introducing a new constant along with its syntax for each such introduced notation. The new constant is defined to be equal to the previous one, modulo arguments. For example, the domain of a relation is the projection of the relation on its first component.

<pre> notation let R be Relation; synonym dom R for proj1 R; </pre>	<pre> abbreviation dom <i>RELAT_1K1</i> R ≡ proj1 R </pre>
---	---

Just like with attributes, to correctly preserve the Mizar semantics we would need to separate the last argument. However, as can be seen in the following example, that last argument can be removed in case of notations. To give complete semantics to an introduced type constant new defined in terms of an existing one old, rather than the Mizar *x is new for x is old* we need to define *new == old*. A concrete example for the Mizar type of uncountable sets is:

<pre> notation let X be set; antonym X is uncountable for X is countable; </pre>	<pre> abbreviation uncountable <i>CARD_3V6</i> ≡ non countable </pre>
--	--

Table 1. Facts available in the Isabelle/Mizar environment, subdivided into major fact categories.

Theorems	5994
Lemmas	478
Definitions	1851
User typing rules	1517
Derived introduction and elimination rules	540
Object typing rules	448
Schemes	270
Definition uniqueness	226
...	
Total imported facts	11920

7 Conclusion

We have proposed a combination of the syntactic and semantic Mizar information, creating an export that includes all the information needed to import Mizar into a logical framework. We imported the first 100 articles of the library into the Isabelle/Mizar object logic, which gives rise to 11,920 Isabelle imported facts of different kinds (see Table 1). The imported library is usable for further development. The first articles from the exported Mizar, the import infrastructure, and our re-formalization are available at: <http://cl-informatik.uibk.ac.at/cek/cicm2018.tgz>

The import process assumes a model of Mizar based on first order logic, therefore by defining a similar model of Mizar in another logical framework the import could be directly reproduced there. We have also exported most of the proofs, but various transformations are necessary to import it in Isabelle. So far, the import only verifies that the statements are consistent and type correct, only importing the proofs would allow complete re-verification of the Mizar library. This still poses a number of challenges. The Isabelle proof nested blocks are weaker than the Mizar now construction [30]. The exported structures generate a large number of selectors which we cannot process automatically yet. Finally, even if Isabelle includes strong general purpose automation [5], it may not be as powerful as the Mizar BY tailored for the Mizar proofs.

The main future work that goes beyond importing and certifying the whole Mizar library in Isabelle is to further develop the Isabelle/Mizar environment in order to allow more convenient proof development. This includes more Mizar-like infrastructure for notations that would allow overloading, disambiguation, and hidden arguments. Furthermore, Mizar includes many tools that allow optimizing complete proof scripts. We would like to generalize such tools to the level of a logical framework. The presence of two libraries with dependencies can allow improved proof automation [9]. Lastly, we also want to generate and provide

documentation for the articles imported in Isabelle, which will allow browsing multiple prover libraries with proofs within one system [8].

References

1. Alama, J., Kohlhase, M., Mamane, L., Naumowicz, A., Rudnicki, P., Urban, J.: Licensing the Mizar mathematical library. In: Davenport, J.H., Farmer, W.M., Urban, J., Rabe, F. (eds.) CICM 2011. LNCS (LNAI), vol. 6824, pp. 149–163. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22673-1_11
2. Aspinall, D., Kaliszyk, C.: What’s in a theorem name? In: Blanchette, J.C., Merz, S. (eds.) ITP 2016. LNCS, vol. 9807, pp. 459–465. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43144-4_28
3. Bancerek, G., et al.: Mizar: State-of-the-art and beyond. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) CICM 2015. LNCS (LNAI), vol. 9150, pp. 261–279. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20615-8_17
4. Bancerek, G., Urban, J.: Integrated semantic browsing of the Mizar mathematical library for authoring Mizar articles. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 44–57. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27818-4_4
5. Blanchette, J.C., Greenaway, D., Kaliszyk, C., Kühlwein, D., Urban, J.: A learning-based fact selector for Isabelle/HOL. *J. Autom. Reason.* **57**(3), 219–244 (2016). <https://doi.org/10.1007/s10817-016-9362-8>
6. Brown, C.E., Urban, J.: Extracting higher-order goals from the Mizar mathematical library. In: Kohlhase, M., Johansson, M., Miller, B., de Moura, L., Tompa, F. (eds.) CICM 2016. LNCS (LNAI), vol. 9791, pp. 99–114. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42547-4_8
7. Bylinski, C., Alama, J.: New developments in parsing Mizar. In: Jeuring, J., et al. (eds.) CICM 2012. LNCS (LNAI), vol. 7362, pp. 427–431. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31374-5_30
8. Corbineau, P., Kaliszyk, C.: Cooperative repositories for formal proofs. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) *Calculemus/MKM -2007*. LNCS (LNAI), vol. 4573, pp. 221–234. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73086-6_19
9. Gauthier, T., Kaliszyk, C.: Sharing HOL4 and HOL light proof knowledge. In: Davis, M., Fehner, A., McIver, A., Voronkov, A. (eds.) LPAR 2015. LNCS, vol. 9450, pp. 372–386. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48899-7_26
10. Grabowski, A., Kornilowicz, A., Naumowicz, A.: Four decades of Mizar. *J. Autom. Reason.* **55**(3), 191–198 (2015)
11. Hurd, J.: The OpenTheory standard theory library. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NFM 2011. LNCS, vol. 6617, pp. 177–191. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20398-5_14
12. Iancu, M., Kohlhase, M., Rabe, F., Urban, J.: The Mizar mathematical library in OMDoc: translation and applications. *J. Autom. Reason.* **50**(2), 191–202 (2013)
13. Irving, G., Szegedy, C., Alemi, A.A., Eén, N., Chollet, F., Urban, J.: DeepMath - deep sequence models for premise selection. In: Lee, D.D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.) NIPS, pp. 2235–2243 (2016)
14. Kaliszyk, C., Krauss, A.: Scalable LCF-style proof translation. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) ITP 2013. LNCS, vol. 7998, pp. 51–66. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39634-2_7

15. Kaliszzyk, C., Pał, K.: Semantics of Mizar as an Isabelle object logic (submitted). <http://cl-informatik.uibk.ac.at/cek/submitted/ckkp-jar17.pdf>
16. Kaliszzyk, C., Pał, K.: Presentation and manipulation of Mizar properties in an Isabelle object logic. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) C1CM 2017. LNCS (LNAI), vol. 10383, pp. 193–207. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_14
17. Kaliszzyk, C., Pał, K.: Progress in the independent certification of Mizar mathematical library in Isabelle. In: Ganzha, M., Maciaszek, L.A., Paprzycki, M. (eds.) Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017, pp. 227–236 (2017)
18. Kaliszzyk, C., Pał, K., Urban, J.: Towards a Mizar environment for Isabelle: foundations and language. In: Avigad, J., Chlipala, A. (eds.) Proceedings of the 5th Conference on Certified Programs and Proofs (CPP 2016), pp. 58–65. ACM (2016)
19. Kaliszzyk, C., Urban, J.: Learning-assisted theorem proving with millions of lemmas. *J. Symb. Comput.* **69**, 109–128 (2015)
20. Kaliszzyk, C., Urban, J.: MizAR 40 for Mizar 40. *J. Autom. Reason.* **55**(3), 245–256 (2015)
21. Naumowicz, A., Piliszek, R.: Accessing the Mizar library with a weakly strict Mizar parser. In: Kohlhase, M., Johansson, M., Miller, B., de Moura, L., Tompa, F. (eds.) C1CM 2016. LNCS (LNAI), vol. 9791, pp. 77–82. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42547-4_6
22. Obua, S., Skalberg, S.: Importing HOL into Isabelle/HOL. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 298–302. Springer, Heidelberg (2006). https://doi.org/10.1007/11814771_27
23. Pał, K.: Combining the syntactic and semantic representations of Mizar proofs (submitted)
24. Urban, J.: XML-izing Mizar: making semantic processing and presentation of MML easy. In: Kohlhase, M. (ed.) MKM 2005. LNCS (LNAI), vol. 3863, pp. 346–360. Springer, Heidelberg (2006). https://doi.org/10.1007/11618027_23
25. Urban, J.: MPTP 0.2: design, implementation, and initial experiments. *J. Autom. Reason.* **37**(1–2), 21–43 (2006)
26. Urban, J., Bancerek, G.: Presenting and explaining Mizar. *Electr. Notes Theor. Comput. Sci.* **174**(2), 63–74 (2007)
27. Urban, J., Sutcliffe, G.: ATP-based cross-verification of Mizar proofs: method, systems, and first experiments. *Math. Comput. Sci.* **2**(2), 231–251 (2008)
28. Wenzel, M.: Isabelle as document-oriented proof assistant. In: Davenport, J.H., Farmer, W.M., Urban, J., Rabe, F. (eds.) C1CM 2011. LNCS (LNAI), vol. 6824, pp. 244–259. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22673-1_17
29. Wenzel, M., Paulson, L.C., Nipkow, T.: The Isabelle framework. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2008. LNCS, vol. 5170, pp. 33–38. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71067-7_7
30. Wenzel, M., Wiedijk, F.: A comparison of Mizar and Isar. *J. Autom. Reason.* **29**(3–4), 389–411 (2002)



Discourse Phenomena in Mathematical Documents

Andrea Kohlhase¹, Michael Kohlhase^{2(✉)}, and Taweechai Ouypornkochagorn³

¹ Hochschule Neu-Ulm, Neu-Ulm, Germany

² Friedrich-Alexander University Erlangen-Nürnberg, Erlangen, Germany
michael.kohlhase@fau.de

³ Srinakharinwirot University, Nakhon Nayok, Thailand

Abstract. Much of the wealth of industrialized societies is based on knowledge that is laid down and communicated in scientific/technical/engineering/mathematical documents: highly structured documents that contain diagrams, images, and – most daunting to many readers – mathematical formulae. It seems clear that digital, interactive documents have the potential to improve reading these kind of documents, and thus learning and applying this kind of knowledge.

To understand how such improvements could be designed, we explore how formula understanding interacts with the surrounding text in mathematical documents. We report on an eye-tracking experiment with 23 engineering students reading a “solved problem” based on a simple differential equation. We observe for instance that – triggered by formulae – readers backjump to previously identified semantic loci and that this behavior is independent of depth of understanding in mathematically trained readers. Based on our observations, we propose novel in-document interactions that could potentially enhance reading efficiency.

1 Introduction

Millions of people engage in reading and understanding scientific/technical/engineering/mathematics (STEM) documents – Germany alone has about two million scientists and engineers. So even a single-digit improvement of this reading/understanding productivity will translate to considerable societal effects.

To raise the productivity we focus on the mathematical documents themselves. Currently, almost all documents are static – usually printed or in page description formats like PDF. They are highly structured, contain diagrams, images, and mathematical formulae, the last one being most daunting to many readers. As reading behaviour does not depend on the medium according to [ZC12], replacing these static documents by digital, interactive documents has the obvious potential to make reading them, and thus learning and applying STEM knowledge, more personal, efficient, effective, and fun. But first we need a better grasp on how people read and understand STEM documents, especially the mathematical parts.

In this paper we report on a design research study for establishing a nascent theory based on the observed phenomena on the discourse-level of embedded mathematics. That is, we were interested in the cognitive mechanisms of reading STEM documents that allow us to systematically design interactive features to improve the reading experience. **Design Research** is a rather young area in Human Computer Interaction – see [ZSF10, HC10]. It includes the “Research for Design” approach, which typically results in **nascent** theories: “*propose tentative answers to novel questions of how and why, often merely suggesting new connections between phenomena*” [EM07, p. 1158].

Concretely, we report on an eye-tracking study that focuses on the interplay of text and formulae in the written communication of mathematical knowledge. This is an interesting angle of attack, as there is a demonstrable correlation between what a participant attends to and where she is looking at – see for example [Ray98] for an overview. The “*eye-mind hypothesis*” [HWH99] even claims a correlation between the cognitive processing of information and the person’s gaze at the specific location of the information.

Related Work. Generally, a lot of studies were conducted to understand in which way eBooks should be designed to improve the reading experience, a summary is given in [Mar09]. When reading academic and scholarly materials, readers triage¹ documents by scanning. Studies like [BL07] have consistently found that titles, section headings, and emphasized text have a high value for document triage and facilitate reading this way. Marshall reports in [Mar09], that when reading more intensely, readers move back and forth through the document. Some authors assume that backjumps are “*an implicit sign that the reader is having difficulty understanding the material*” [Cha+16].

In previous work the first two authors have studied how humans read and understand mathematical formulae from their visual representations via eye-tracking experiments. In [KF16] we show that the level of affinity towards mathematics distinguishes how readers process formulae and in [KKF17] we show that mathematically trained readers process formulae by recursively identifying content-oriented patterns (Gestalts) that build up the formula semantics (Gestalt Tree Hypothesis). This first parsing phase is followed by a “formula understanding phase” in which salient semantic loci are systematically re-visited, e.g. for tracking bound variables in integrals.

Our literature search didn’t reveal any results concerning the **discourse level** – i.e., the phenomena above the phrase and formula structures: sentences, paragraphs, dialogue – in mathematics or even STEM documents.

Overview. After documenting the experimental setup in Sect. 2 we observe and discuss conspicuous patterns in subjects’ gaze behaviors in Sect. 3. In Sect. 4 we proceed with a quantitative analysis of gaze data on specifically selected document fragments. Observations and explanations are summarized in a nascent

¹ **Document triage** refers to readers’ practice of rapidly evaluating documents to determine whether they contain wanted information or not.

theory for reading mathematical documents in Sect. 5. Subsection 5.1 discusses how these findings can lead to better interaction with STEM documents and Sect. 6 concludes the paper.

2 The Eye-Tracking Study

Experimental Setup. This eye-tracking study was carried out with the students of a one-week special course “*Content and Form: How one manipulates the other*” at Srinakharinwirot University (SWU), Thailand. Most of the 23 participants were students of the Biomechanical Engineering Program at SWU, the remaining three were students of Electrical Engineering. The primary textbook of these programs is “*Advanced Engineering Mathematics*” by Erwin Kreyszig [Kre06]. Reading Kreyszig’s book in English is a well-practiced part of the program, so the English/Thai language barrier should be minimal for reading. We also note that in the sense of [KF16] all participants are math-oriented and mathematically trained.

In our study we used a solved problem called “*Radioactive Decay*” in [Kre06, p. 13], which had been transliterated into HTML5 with MathML by the authors, as mathematical document². Here, Kreyszig uses the example of radiocarbon-dating the Ötzi mummy to present standard methods for solving a boundary value problem induced by the simple ordinary differential equation $y' = ky$. This example comes very early in Kreyszig’s book, therefore we assume that all subjects were familiar with the content and able to understand the mathematics.

The 23 students (16 female, 7 male) were presented the example on a Tobii t60 Eye-Tracking Screen (17” and 4:3 ratio with 60 Hz) in a mobile setup. They were asked to think aloud while reading/understanding the document³; audio/video recordings were collected together with the eye-tracking data.

In a post-test, subjects were asked to write a from-memory summary of the document they read in the study in Thai. This was announced beforehand to subjects and was mainly intended to provide motivation for reading the document in detail. This document was later evaluated to assess the level of understanding of the participants. We believe that the “re-production” of results in Thai helped break down the language barrier.

² Figures 12 and 15 together show the full document content, see <https://kwarc.info/people/mkohlhase/data/DuStd-18/radiocarbon.html> for the transcription.

³ This has been unsuccessful, it seems that the English/Thai language barrier combined with a cultural reluctance to speak without preparation together with the unfamiliar situation induced prohibitive cognitive load which prevented students from speaking. When we realized this, we asked one student to “think aloud in Thai”, but this largely only resulted in a translation of the document, in particular not in the desired stream of cognition, so we dropped this idea.

Classifying the Understanding-Level of Our Test Subjects. The Thai summary documents were evaluated to classify the individual level of understanding at various levels. In particular, the post-test results were assessed according to whether the participants:

1. addressed the problem objective, i.e., determining the time of death of Ötzi;
2. addressed the physical background and/or solution methodology;
3. cite the eventual answer to the problem; and
4. correctly used formulae in the document.

We aggregated these to estimate the *level of understanding* by the students of the problem setting and methodology (U_p from 1. and 2.) and that of formulae in the text (U_f). For the former, we used a true/false scale, for the latter a five-point Likert scale: “yes”, “maybe”, “cannot judge”, “no formulae”, and “formulae catastrophically wrong”. Note that these judgements are not independent: A problem in understanding the methodology renders subsequent aspects not applicable: Indeed we had this in three cases, we found that subjects had misunderstood the problem to be “*Find the half-life of the mummy*”, “*Find the half-life of ${}_6C^{14}$ or ${}_6C^{12}$ will occur after the death*”, and “*Prove the ratio of Carbons is 52.66%*”.

It turns out that values in the understanding aspects U_p and U_f are largely identical where applicable, if we identify “yes” and “maybe” in U_p with “yes” in U_f and analogously “no formulae” and “formulae catastrophically wrong” in U_p with “no” in U_f . Therefore we used the “general aptitude” (high/low on the combined score) for grouping the participants into the *two groups* LOW and HIGH. See Fig. 1 for the distribution, where “na” stands for non-applicable as we couldn’t sort the respective participants into any of the groups.

LOW	7
HIGH	14
na	2
Total	23

Fig. 1. Participants

3 Patterns in the Gaze Plots

We will now analyze the results of the eye-tracking experiment qualitatively. Concretely, we will study gaze plots, i.e., visualizations of fixations⁴ over time, generated by the eye-tracker and show typical patterns. In general we distinguish between text, inline-math, and display-math areas and observe discourse phenomena as behavioural gaze patterns on these.

⁴ The eye moves discontinuously, making short stops (called **fixations**; we count any stop that is longer than 60 ms) and separating by rapid jumps (called **saccades**).

Jumps. The first pattern, we observed in the majority of participants' gazes, is a **jump** from a math expression in a display-math area back into a

Radioactive decay is governed by the $\frac{dy}{y} = k dt$. By separation and integration (where t is time and y_0 is the initial ratio of ${}_6C^{14}$ to ${}_6C^{12}$)

$$\frac{dy}{y} = k dt, \ln|y| = kt + c, y = y_0 e^{kt}$$

Fig. 2. Backjumping to equations

related inline-math expression close-by and above, a “regression”. Figure 2 shows an example: While the subject looks at the equation “ $\frac{dy}{y} = k dt$ ”, he/she jumps back to the ordinary differential equation “ODE $y' = ky$ ” in the text above which it was derived from. Afterwards he/she continues to the ensuing equations in the same display-math line. Interestingly, we did not observe once a forward jump from the display-math to an inline-math expression. That is quite natural at the beginning when one doesn't yet know what is about to come, but at the end, when all content is already known, we could have expected a different behaviour.

This backjumping pattern also happens in a more complex way between distinct display-math lines as in Fig. 3, where the participant looks at an equation – here $y_0 e^{kt} = 0.5 y_0$ – after fixating on the left hand side (fixations 1, 2) he jumps back to the equation $\frac{dy}{y} = k dt$ three lines above (fixations 3–5) and returns to the right hand side (fixations 6, 7).

$\frac{dy}{y} = k dt, \ln|y| = kt + c, y = y_0 e^{kt}$

Next we use the half-life $H = 5715$ to determine original substance is still present, thus

$y_0 e^{kH} = 0.5 y_0$

Fig. 3. Complex backjumping

Justification Processing. In Fig. 4 we see a participant reading the equation $\frac{dy}{y} = k dt$ and then jumping back to the phrase “By separation” and dwelling on this extensively before returning to the equation and moving on. Here “By separation and integration” is the justification for the equations $\frac{dy}{y} = k dt$ (separation) and $\ln|y| = kt + c$ (integration); indeed it is the main method introduced in the same chapter of [Kre06] that the document in our experiment was

Radioactive decay is governed by the ODE $y' = ky$. By separation and integration (where t is time and y_0 is the initial ratio of ${}_6C^{14}$ to ${}_6C^{12}$)

$$\frac{dy}{y} = k dt, \ln|y| = kt + c, y = y_0 e^{kt}$$

Fig. 4. Processing justifications

In the next equation $e^{kt} = 0.5$ something similar happens: after examining the equation, the subject jumps back to the equations $\ln|y| = kt + c$ and $y = y_0 e^{kt}$, glances at them without fully examining them again, and returns to the first equation on the lower line (glances at that and continues back to equation $e^{kt} = 0.5$ that was the origin of the detour).

taken from. So the **Justification Processing** pattern describes backjumps to justification keywords in the text that help to process the reasoning behind the solution.

Declaration Lookup. The next pattern is somewhat less frequent, but still observed regularly. Figure 5 shows a situation, where the test subject is

Radioactive decay is governed by the $\frac{dy}{y} = -kt$. By separation and integration (where t is time and y_0 the initial ratio of ${}^6C^{14}$ to ${}^6C^{12}$)

$$\frac{dy}{y} = -kt, \quad \ln y = -kt + c, \quad y = y_0 e^{-kt}$$

Fig. 5. Regression to a declaration

reading the equation $y = y_0 e^{-kt}$ and starts from the left with y (fixation 10), continues to y_0 (fixation 11), jumps to the phrase “ y_0 is the initial ratio ...”, and back to the equation, which is examined more closely before moving on. In this **declaration lookup** pattern the backjumps are captured that are reaffirming information. Humans keep information in short term memory only for 10–30 s, so such jumps are necessary to keep the content available.

fe $H = 5715$ to determine k . When $t = H$, half of the ill present, thus

$$e^{-kH} = 0.5, \quad k = \frac{\ln 0.5}{H} = -\frac{0.693}{5715} = -0.000$$

Fig. 6. Multiple declaration lookup

Figure 6 shows another instance of this. The participant reads the (exponent of the left-hand side in the) equation $e^{-kH} = 0.5$ and then glances at the k in the line above and then reads

“ $t = H$ half of the”, which reminds him of the description of the half-life H .

Re-check and Re-orientation. A very common pattern is that participants read the explanatory text and the equations essentially sequentially or linewise (see Fig. 7), with the latter being subject to the jumps and declaration lookups described above.

Radioactive decay is governed by the $\frac{dy}{y} = -kt$. By separation and integration (where t is time and y_0 the initial ratio of ${}^6C^{14}$ to ${}^6C^{12}$)

Fig. 7. Reading linewise

But when they reach the answer at the end, about 2/3 of the subjects went over large parts of the document again in a much more targeted fashion. We call this pattern a **re-check**. Figure 8 shows two typical situations. Extended re-checks also occurred before the final answer was reached, but this was much less frequent. In this situation extended excursions focused on the background, problem, and physical information sections – we call these **re-orientations**.

Physical Information

In the atmosphere and in living organisms, the ratio of radioactive ${}^6C^{14}$ (made radioactive by cosmic rays) to ordinary ${}^6C^{12}$ is constant. When an organism dies, its absorption of ${}^6C^{14}$ by breathing and eating terminates. Hence one can estimate the age of a fossil by comparing the radioactive carbon ration in the fossil with that of the atmosphere. To do this one needs to know the half-life of ${}^6C^{14}$, which is 5715 years.

Solution

Radioactive decay is governed by the ODE $y' = ky$. By separation and integration (where t is time and y_0 is the initial ratio of ${}^6C^{14}$ to ${}^6C^{12}$)

$$\frac{dy}{y} = k dt, \quad \ln |y| = kt + c, \quad y = y_0 e^{kt}$$

Next we use the half-life $H = 5715$ to determine k . When $t = H$, half of the original substance is still present, thus

$$y_0 e^{kH} = 0.5 y_0, \quad e^{kH} = 0.5, \quad k = \frac{\ln 0.5}{H} = -\frac{0.693}{5715} = -0.0001213.$$

Finally, we use the ration 52.5% for determining the time t when Ötzi died (actually was killed),

$$e^{kt} = e^{-0.0001213t} = 0.525, \quad t = \frac{\ln 0.525}{-0.0001213} = 5312 \quad \text{Answer: 5300 years ago}$$

(a) Text and Equations

$$\frac{dy}{y} = k dt, \quad \ln |y| = kt + c, \quad y = y_0 e^{kt}$$

Next we use the half-life $H = 5715$ to determine k . When $t = H$, half of the original substance is still present, thus

$$y_0 e^{kH} = 0.5 y_0, \quad e^{kH} = 0.5, \quad k = \frac{\ln 0.5}{H} = -\frac{0.693}{5715} = -0.0001213.$$

Finally, we use the ration 52.5% for determining the time t when Ötzi died (actually was killed),

$$e^{kt} = e^{-0.0001213t} = 0.525, \quad t = \frac{\ln 0.525}{-0.0001213} = 5312 \quad \text{Answer: 5300 years ago}$$

(b) Equations Only

Fig. 8. Final re-check after answer was read

Solution Pre-scan. Dually, we sometimes see a pre-scan that skips ahead to the eventual solution glancing over salient features of the document until the answer has been reached (see Fig. 9).

This is usually followed by a linewise reading of the solution text, which is supposedly informed by the pre-scanned text.

Multiple Ways of Reading Equations. We found different patterns of reading the equation $k = \frac{\ln 0.5}{5715} = \frac{0.693}{5715} = 0.0001213$: Figure 10(a) e.g. starts out with

Solution

Radioactive decay is governed by the ODE $y' = ky$. By separation and integration (where t is time and y_0 is the initial ratio of ${}^6C^{14}$ to ${}^6C^{12}$)

$$\frac{dy}{y} = kat, \quad \ln |y| = kt + c, \quad y = y_0 e^{kt}$$

Next we use the half-life $H = 5715$ to determine k . When $t = H$, half of the original substance is still present, thus

$$y_0 e^{kH} = 0.5 y_0, \quad e^{kH} = 0.5, \quad k = \frac{\ln 0.5}{H} = -\frac{0.693}{5715} = -0.0001213.$$

Finally, we use the ration 52.5% for determining the time t when Ötzi died (actually was killed),

$$e^{kt} = e^{-0.0001213t} = 0.525, \quad t = \frac{\ln 0.525}{-0.0001213} = 5312 \quad \text{Answer: 5300 years ago}$$

Fig. 9. Solution pre-scan

the middle of the three equality symbols, moved to the first, and then to the value 0.5 and then to the natural logarithm ln. From here the subject focuses on the value 0.693 for three fixations (6–8) before he/she moves on to the final value on the very right of the equation chain. Note that this sequence is consistent with the Gestalt Tree hypothesis from [KKF17] if we assume the middle = to be the main operator and the two others to be the main operators of its arguments.

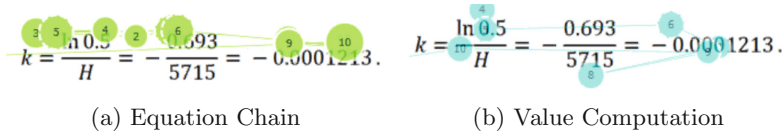


Fig. 10. Different ways to read equations

The proband of Fig. 10(b) comes in from the phrase “determine k ” and focuses on the first fraction, directly moves to the final value, and then fixates the middle right fraction, returns to the value, and then passes to the middle left fraction before moving on. We can assume that this participant interpreted the equation chain as an instance of the “value computation frame”: an equation chain with a variable on the left and a scalar value on the right. As in [KKF17], we assume that single identifiers can be interpreted without fixation, so the initial parse of the Gestalt identifies the “outer equation” $k = 0.0001213$, which is indeed the relevant information. The middle fractions are checked as a secondary objective.

Whitespace to Think With. Finally, we repeatedly found patterns like the ones in Fig. 11. We interpret this as participants seeking a place without distracting information to fixate while thinking about what they were reading. We also frequently observed long fixations of the part headings – especially the heading “Solution”, which seemed to serve a similar purpose.

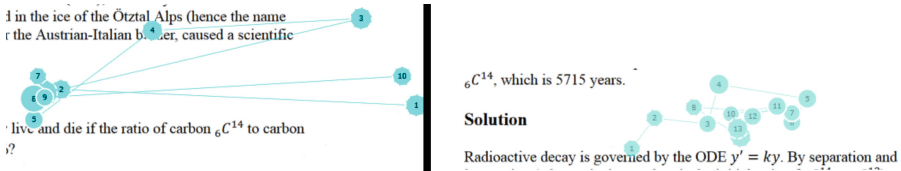


Fig. 11. Times for reflection: whitespace to think with

One could think that the fixations on this area were due to mindless reading times, that is reading at the same time as thinking about something else. But in [RRS10] it was shown that mindless reading was always immediately preceded by especially erratic eye movements, so we can refute this argument.

4 Patterns in the Elicited Data

To make use of the eye-trackers statistical data, one has to define **Areas of Interest** (AOI). For each AOI eye-tracking specific metrics like “Time to First Fixation” or “Visit Duration” are elicited from the gathered data. A **visit** is the

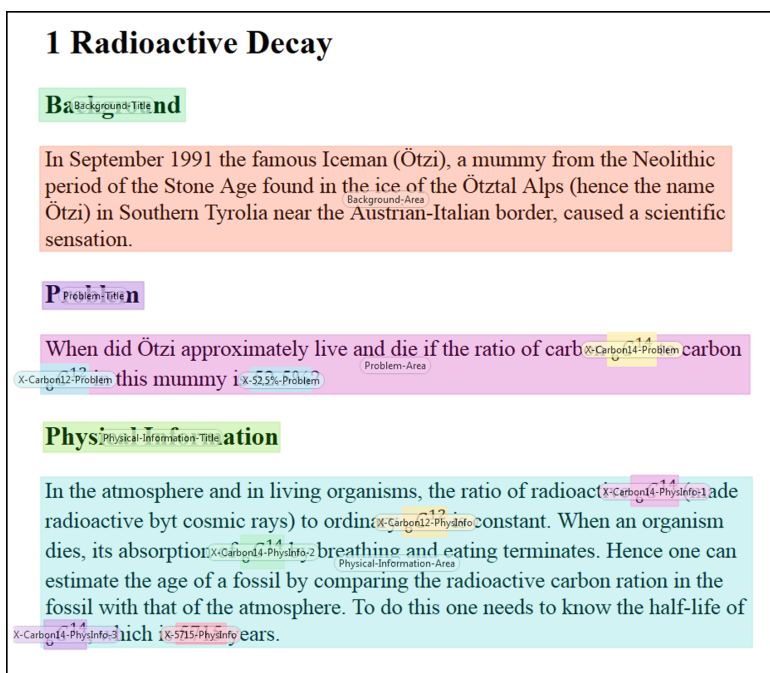


Fig. 12. Structure-driven AOIs (Color figure online)

period of time from entering an AOI to leaving it, whereas a **fixation** occurs, when the eye focuses on a position for a certain amount of time. A visit contains at least one fixation otherwise it is not counted as a visit as the test subject otherwise just jumped over the AOI without having time to perceive any information.

To get a better understanding of the Gestalt Tree we distinguished between the visual document structure and the visual structure of mathematical expressions, so we defined **structure-driven AOIs** as in Fig. 12 and **math-driven AOIs** as shown in Fig. 15. The colored areas in those figures represent such areas of interest. For instance, we marked the heading of each subsection in Fig. 12 independently from its content, so we have a Background-Title-AOI and a Background-Area-AOI. These were then used for analyzing the eye-tracker's data, predominantly using the mean for comparing the eye-tracking specific metrics.

4.1 Structure-Driven Areas of Interest

Kreyszig's example consists of four top-level text areas introduced by explicit headings: the background information, the problem statement, the solution-relevant physical background, and the solution. All these were given a clearly marked title and provide an obvious visual text structure (see AOIs on x-axis in Fig. 13).

First, we were interested how often our subjects visited or fixated the text components, respectively. In Fig. 13 we visualized the number of visits and fixations of the structural text components depending on participants' level of understanding.

An interesting observation consists in the strong difference between fixations and visits. Especially in the solution area the number of fixations is more than quadrupling the number of visits. That means that subjects jumped a lot between sub-elements within the respective area during a visit. So the interactivity rate for the solution area is highest, followed by the relevant physical information area. The background information was more often fixated than the problem description. Looking at the size of the areas, this is not surprising: the problem statement is about half the size of the background area, almost doubled by the physical information, which in turn is almost doubled by the solution. Therefore, we could expect a linear growth in the number of fixations. But if we look closely, we can see that it is more than linear.

At first glance surprisingly, there is neither a difference in terms of number of fixations nor visits between students of high or low level document understanding. [MGA14] already found a similar result with respect to text comprehension. Our analysis suggests that this is true with respect to mathematical text as well.

Next, we looked at other metrics to (nevertheless) found our intuition about existing differences among the participant groups HIGH and LOW.

Figure 14 shows the relative total duration of visiting or fixating specific areas, distinguishing the ones with a high level of understanding from the ones with a low level. Note that the visit duration has always to be higher than the fixation duration, as subjects' fixations depend on their visit of this area.

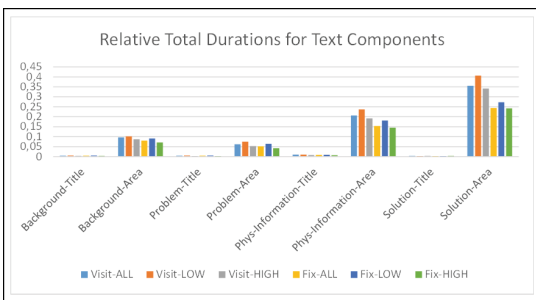


Fig. 14. Relative total visit- and fixation-duration for the structure-driven AOIs

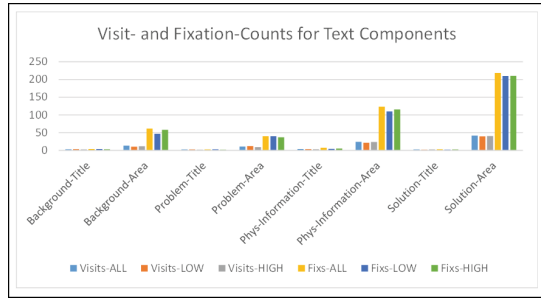


Fig. 13. Visits and fixations for S-AOIs

Moreover, we used the standard threshold for fixation length of 60 ms, so besides the durations for non-classified participants, all shorter fixations are also summed up in the general “*-ALL” variables.

Non-surprisingly, in general the title areas were very shortly looked at and the solution area the longest. The participants of the HIGH group

spent less time fixating and less time visiting all areas. The length of fixations and visits could therefore indicate aptitude to the task, that is, it could give us a measure for *personal complexity of information*.

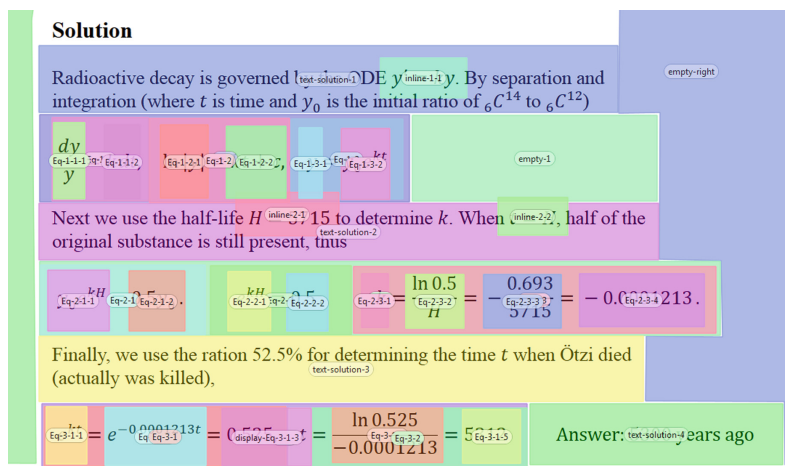


Fig. 15. Math-driven AOIs (Color figure online)

4.2 Math-Driven Areas of Interest

To get a better grasp about the discourse level when reading mathematics, we had a closer look to the solution itself as it contains paragraphs and formula areas. Except for the last text area all are of a comparable size, so that any differences can be attributed to the content itself. We distinguish four nested levels of formula areas (see Fig. 15):

- the display equation level containing the display math in a line represents the **highest level**, e.g. “Eq-1”.
- Each equation area contains several subareas with individual equations like “Eq-1-1” (separated by a comma), which we consider the **intermediate level**,
- which in turn consists of the sub-expressions on the left and the right of the equation sign of each subexpression on the **lowest level**, for instance “Eq-1-1-1”.
- In the text areas in the solution there are several occurrences of **inline math** like “inline-1-1”, that is math that is embedded in text.

We built the AOIs according to this structure, where Eq-1 – Eq-3 belong to the highest level, Eq-1-1 – Eq-3-2 belong to the intermediate level etc.

One would expect the number of the counts to equal the number of the sum of its parts. But this is not the case. Instead we can see in Fig. 16 that the sum of the subarea visits is consistently larger than the higher level area, the sum of the subarea fixations in contrast consistently lower.

As we left out some areas inside the higher level mathematical expressions to be covered by lower level ones, the higher number of fixations is explained by our subjects not always fixating on the sublevel areas but somewhere in-between. They didn't always fixate the formulae, only close-by - maybe on a recheck-jump in-between or to perceive the higher-level Gestalt information. The higher number of visits on the lower level areas means that the test subjects left and entered sublevel areas without leaving the higher level mathematical expression. Note that this inside-math-expression interactivity happened on all higher-levels and in a 50-50 ratio on the intermediate levels.

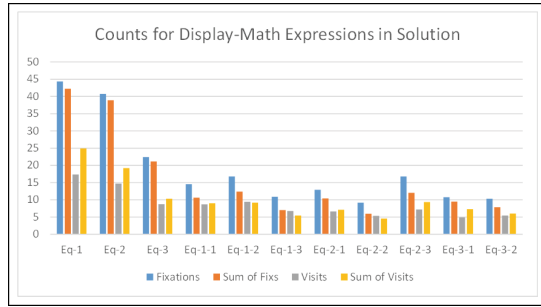


Fig. 16. Fixations and visits in the solution

4.3 What are Cognitive Units (Words) in Math?

Another point of interest is the distribution of gaze intensity over the document. Typically, this is assessed visually by heatmaps. But the AOIs allow us to “do the numbers” to see how intensity differs by type of content, here between text, inline, and display math.

But how to define **gaze intensity**? Heat maps visualize fixations per space. With AOIs, we can calculate the number of fixations per AOI visit, but we still have to normalize for “space”. For text the natural cognitive unit is a *word*, which is usually read holistically, or alternatively *word characters*, if we want to take word length into account. We can only compare gaze intensity on text and math, when we understand what the equivalent to a word or a word character in formulae is, but as we will see, that is not per se clear. To fortify our intuition, let us look at the following sentence from our document:

Radioactive decay is governed by the ODE $y' = ky$.

We can determine the number of words in the text part to be 7 and we count 34 letters ignoring the whitespaces. For the mathematical part it is indeed difficult: Does the entire equation represent a word, does the left part of the equation equal a word, or do we have to count the units explicitly spoken in the mathematical expression like “y prime equals k (times) y”. Furthermore, we have to decide whether the equality sign is comparable to punctuation or what else is.

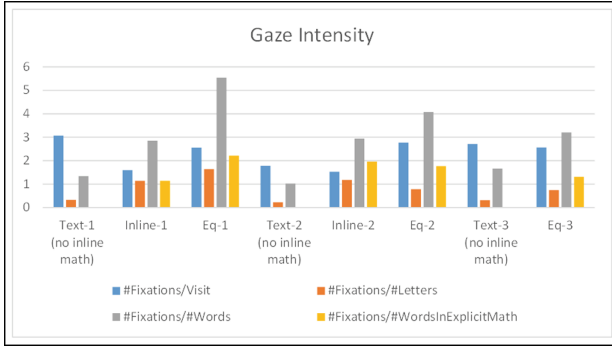


Fig. 17. Fixations per AOI size and word number (Color figure online)

Figure 17 shows the fixations per AOI and various space normalizations. Here, we looked only at the math-driven AOIs of type text, inline-math and display math on the highest level. Together these AOIs cover all non-empty areas in the solution section and each fixation falls into exactly one AOI. For #Words in formulae we counted the “equation

sides” whereas in #WordsInExplicitMath we counted operators and constants which roughly corresponds to words in “spoken formulae”, but normalizes for multi-word operators like “*e* to the power of *k* (times) *t*”. If we compare the intensity columns for the various AOIs, we see that the values are significantly higher on formulae than text (by factors ranging from 2 to 5 for #Letters and #Words). Only for the intensity measure that normalizes with #WordsInExplicitMath we see comparable values.

The first (blue) column in Fig. 17 shows the fixations per visit, which is not a “gaze intensity” as it is not normalized for (cognitive) space, but it is another intensity measure. The more fixations there are per visit, the more complex the information. We can interpret the results in terms of

- (i) *cognitive load* – and therefore probably information content and relevance to a STEM document: formulae are much more content-rich than text
- (ii) a *complexity measure of formulae*: nodes in the operator tree representation of formulae are an adequate size measure for formulae.

Note that these interpretations do not compete as they address different aspects. The latter interpretation is consistent with the Gestalt Tree Hypothesis from [KKF17] and the finding of [Bau99] that mathematical formulae (especially inline ones) have grammatical function in the surrounding text and are best modeled by integration of the mathematical and sentence grammars.

5 A Nascent Theory for Mathematical Documents

We summarize the most relevant observations to be used for explorative design in a first, admittedly very basic nascent theory.

NT1: The top level block structure of the experiment’s mathematical document was noticed and used while reading – reconfirming previous results for general text reading for mathematical document reading.

- NT2:** Participants mostly read the text linewise in document order, starting at the beginning. The more was read, the more regressions happened, especially when starting the solution area which has a high intensity of formulae. Here, we recognized several distinct patterns.
- *Local Regressions* to (i) identifier declarations, (ii) equations the current one is derived from, and (iii) justifications of the current equation. Contrary to [Cha+16] – which admittedly was not focused computer science papers, not on mathematics – we found that students with a high level of understanding used regressions – probably to deepen their knowledge.
 - *Non-Local Regressions* to the problem and background descriptions after they have read the solution. Recaps are largely driven by the display formulae in the derivation, whereas re-orientations are driven by terminology coreference.
- NT3:** Participants spend significantly more time on formulae than on text. This also holds for inline formulae, but only if they are more than one letter long. As a rule of thumb, one operator or constant in a formula is worth one word in a text.
- NT4:** There seems to be a lot of looking to the right margin and the section title areas, which can be interpreted as a need for “spaces to think with”.

5.1 Design Application: Interactive Documents

We give some very first examples for explorative design ideas based on **NT1–NT4** to showcase the value of a nascent theory.

For instance, if structural elements are easily findable, then we strengthen the effect of **NT1**: What about personalizing structural layout? That way everyone loves and lives with her/his own recognition clues.

When reading/understanding formulae, people look for declarations (**NT2–i**). With active documents – i.e., interactive documents that adapt themselves to the user; see [Koh+11] – we can support this process.

Consider for instance a simple instrumentation like the one shown in Fig. 18, where we have bound the hover event on the sub-formula T to highlight the declaration text and other occurrences of the same identifier via two lines of jQuery. Of course, we also systematically need label/ref annotations on corresponding sub-formulae, which is the real bottleneck.

We have shown in [WG10] that the vast majority of identifiers is declared near where they are used. [Kri+12, Sch16] present a simple algorithm and framework for “declaration spotting”, which could be used to generate such annotations automatically. Of course, this needs to be improved substantially to be practical for document instrumentation.

Let $T(t)$ be the temperature inside the building (Law). Then by Newton’s law,

$$\frac{dT}{dt} = k(T - T_A) \quad (1)$$

Fig. 18. Instrumenting documents

Similarly, we can instrument the document to support the re-check and re-orientation phases diagnosed in (**NT2**–*Non-Local Regressions*). For re-checking, we would instrument the formula dependency relation: for a formula F focused by the reader⁵ we could highlight all formulae (or alternatively their relevant parts) that compute objects in F . For instance, if F is e^{kt} on the left of the last equation line, then we could highlight the dependency $k = \frac{\ln 0.5}{H} = \frac{0.693}{5715} = 0.0001213$ and the declaration “ t is time”. Again, we are presuming annotations for the dependency relation in the document. Note that the recap/re-orientation patterns discussed in Sect. 3 suggest that a highlighting instrumentation is more effective than e.g. generated summary (at least when all recaps are on the same page): the subjects seemed to have a very clear notion of where to find the information they were looking for in the document.

Based on **NT3** we could explore the effect of enlarging inline math or we use the complexity measure based on mathematical words to assess the necessity of assistance features.

Introducing explicit thinking space to a mathematical document layout seems rather unusual, but might be helpful for the reflection process in individuals according to **NT4**.

We have experimented with these and other instrumentations during the course, but the eye-tracking studies on this were inconclusive, as the number of tested students were too small and the participants untrained in the new features. The latter was a main problem in all of the instrumentations: they were not easily discoverable – we did not want to change the appearance of the document too much – and subjects needed time for understanding what they were seeing.

6 Conclusion

In this paper we report on an eye-tracking experiment, observing engineering students reading a mathematical document. In contrast to other studies which focus on text or formulae alone, we focus on the discourse-level interaction of text and formulae. We have identified various conspicuous patterns in the data from the experiment and shown how these could be used to improve the reading and understanding experience of STEM practitioners. The nascent flavor of our theory notwithstanding it is already useful; we can e.g. refute the assumption that the occurrence of backjumps indicate a low level of understanding as suggested in [Cha+16].

In the future we plan to systematically design interactive features for STEM documents and evaluate the effectiveness and efficiency of reading and understanding STEM documents. On the other hand we plan eye-tracking studies that further elucidate the cognitive processes behind perceiving mathematical documents, hopefully ending with a “mature” theory, which “*present[s] well-developed*

⁵ There is of course the practical problem of how to determine whether F is focused. We could use in-place-and-time eye-tracking data instead of forcing the reader to e.g. hover the mouse over F to “focus” it, as this might be too distracting.

constructs and models that have been studied over time with increasing precision [...] resulting [...] in points of broad agreement” [EM07, p. 1158].

Acknowledgements. The authors gratefully acknowledge the Erasmus+ Staff Mobility for Teaching program that provided the institutional setting of the design research, and the students of the course for their participation in the eye-tracking study.

References

- [Bau99] Baur, J.: Syntax und Semantik mathematischer Texte - ein Prototyp. MA thesis. Fachrichtung Computerlinguistik, Universität des Saarlandes, Saarbrücken, Germany (1999)
- [BL07] Buchanan, G., Loizides, F.: Investigating document triage on paper and electronic media. In: Kovács, L., Fuhr, N., Meghini, C. (eds.) ECDL 2007. LNCS, vol. 4675, pp. 416–427. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74851-9_35
- [Cha+16] Mozaffari Chanijani, S.S., et al.: An eye movement study on scientific papers using wearable eye tracking technology. In: 2016 Ninth International Conference on Mobile Computing and Ubiquitous Networking (ICMU), pp. 1–6. IEEE (2016). <https://doi.org/10.1109/ICMU.2016.7742085>
- [EM07] Edmondson, A., McManus, S.: Methodological fit in management field research. *Acad. Manag. Rev.* **32**, 1155–1179 (2007)
- [HC10] Hevner, A., Chatterjee, S.: *Design Research in Information Systems: Theory and Practice*, 1st edn. Springer, New York (2010). <https://doi.org/10.1007/978-1-4419-5653-8>
- [HWH99] Henderson, J.M., Weeks Jr., P.A., Hollingworth, A.: The effects of semantic consistency on eye movements during complex scene viewing. *J. Exp. Psychol. Hum. Percept. Perform.* **25**(1), 210–228 (1999). <https://doi.org/10.1037/0096-1523.25.1.210>
- [KF16] Kohlhasse, A., Fürsich, M.: Understanding mathematical expressions: an eye-tracking study. In: Kohlhasse, A., Libbrecht, P. (eds.) *Mathematical User Interfaces Workshop*, July 2016. <http://ceur-ws.org/Vol-1785/M2.pdf>
- [KKF17] Kohlhasse, A., Kohlhasse, M., Fürsich, M.: Visual structure in mathematical expressions. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) *CICM 2017*. LNCS (LNAI), vol. 10383, pp. 208–223. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_15
- [Koh+11] Kohlhasse, M., et al.: The planetary system: Web 3.0 & active documents for STEM. In: *Procedia Comput. Sci.* **4** (2011). (Special issue: In: Sato, M., et al. (eds.) *Proceedings of the International Conference on Computational Science (ICCS)*, pp. 598–607. <https://doi.org/10.1016/j.procs.2011.04.063>)
- [Kre06] Kreyszig, E.: *Advanced Engineering Mathematics*, 9th edn. Wiley, Hoboken (2006)
- [Kri+12] Kristianto, G.Y., et al.: Extracting definitions of mathematical expressions in scientific papers. In: *The 26th Annual Conference of the Japanese Society for Artificial Intelligence* (2012). <https://kaigi.org/jsai/webprogram/2012/pdf/719.pdf>
- [Mar09] Marshall, C.C.: *Reading and Writing the Electronic Book*. Morgan and Claypool Publishers, San Rafael (2009)

- [MGA14] Martínez-Gómez, P., Aizawa, A.: Recognition of understanding level and language skill using measurements of reading behavior. In: Proceedings of the 19th International Conference on Intelligent User Interfaces, IUI 2014, pp. 95–104. ACM, Haifa (2014). <https://doi.org/10.1145/2557500.2557546>
- [Ray98] Rayner, K.: Eye movements in reading and information processing: 20 years of research. English. Psychol. Bull. **124**(3), 372–422 (1998)
- [RRS10] Reichle, E.D., Reineberg, A.E., Schooler, J.W.: Eye movements during mindless reading, vol. 21, pp. 1300–1310, September 2010. <https://doi.org/10.1177/0956797610378686>
- [Sch16] Schaefer, J.F.: Declaration spotting in mathematical documents. B.Sc. thesis. Jacobs University Bremen (2016). <https://gl.kwarc.info/supervision/BSc-archive/blob/master/2016/schaefer-frederick.pdf>
- [WG10] Wolska, M., Grigore, M.: Symbol declarations in mathematical writing: a corpus study. In: Sojka, P. (ed.) Towards Digital Mathematics Library, DML Workshop, pp. 119–127. Masaryk University, Brno (2010). http://dml.cz/bitstream/handle/10338.dmlcz/702580/DML.003-2010-1_14.pdf
- [ZC12] Zambambieri, D., Carniglia, E.: Eye movement analysis of reading from computer displays, eReaders and printed books. Ophthalmic Physiol. Opt. **32**(5), 390–396 (2012). <https://doi.org/10.1111/j.1475-1313.2012.00930.x>
- [ZSF10] Zimmerman, J., Stolterman, E., Forlizzi, J.: An analysis and critique of research through design: towards a formalization of a research approach. In: Proceedings of the 8th ACM Conference on Designing Interactive Systems, DIS 2010, pp. 310–319. ACM, Aarhus (2010). <https://doi.org/10.1145/1858171.1858228>



Finding and Proving New Geometry Theorems in Regular Polygons with Dynamic Geometry and Automated Reasoning Tools

Zoltán Kovács^(✉) 

The Private University College of Education of the Diocese of Linz,
Salesianumweg 3, 4020 Linz, Austria
zoltan@geogebra.org

Abstract. In 1993 Watkins and Zeitlin published a method [1] to simply compute the minimal polynomial of $\cos(2\pi/n)$, based on the Chebyshev polynomials of the first kind. For the work presented in this paper we have implemented a small augmentation, based on Watkins and Zeitlin's work, to the dynamic mathematics tool GeoGebra. We show that this improves GeoGebra's capability to discover and automatically prove various non-trivial properties of regular n -gons.

Discovering and proving, in this context, means that the user can sketch a conjecture by drawing the geometric figure with the tools provided by GeoGebra. Then, even if the construction is just approximate, in the background a rigorous proof is computed, ensuring that the conjecture can be confirmed, or must be rejected.

In this paper the potential interest of automated reasoning tools will be illustrated, by describing such new results in detail, obtained by some recently implemented features in GeoGebra.

Besides confirming well known results, many interesting new theorems can be found, including statements on a regular 11-gon that are impossible to represent with classical means, for example, with a compass and a straightedge, or with origami.

Keywords: Automated theorem proving · Computer algebra
Regular polygons · GeoGebra · Regular 11-gon
Chebyshev polynomials

1 Introduction

There are many well-known theorems on interesting properties in regular polygons. Most results, however, deal with constructible polygons, whose vertices can be constructed with traditional means including compass and straightedge, or origami, for example. From the very start of the availability of computer algebra

systems (CAS) and dynamic geometry software (DGS), namely, the 1990s, however, non-constructible polygons can also be better observed, either numerically or symbolically.

In this paper we focus on the symbolic approach on proving facts on regular polygons. In particular, we search for equally long segments, congruent triangles, perpendicular or incident lines, that can appear in a regular polygon by drawing the diagonals, intersecting them, and creating new points by the intersections, then, by connecting the new points, we can obtain more and more new lines as well, and so on.

The way we here highlight is well-known in mechanical geometry theorem proving, and has been being used for about 30 years since the revolutionary book [2] of Chou, that presents 512 mathematical statements with an automated proof for each. In Chou's approach—which is based on Wu's algebraic geometry method [3]—an equation system can describe a geometric construction, and by performing some manipulations on the equation system, a mechanical proof can be obtained. Wu's work, and most contributions of his followers, focus mainly on constructible setups, that is, mostly on such constructions that can be created only by using the classic approach, namely by compass and straightedge. In this paper we will call this well known approach *algebraic geometry approach* (AGA).

By contrast, this paper introduces some new equations which are able to extend the AGA to work with regular n -gons as well. The roots of computing the minimal polynomial of $\cos(2\pi/n)$ are already present in Lehmer's work [4] in 1933, but a useful formula for CAS has just recently been highlighted by Watkins and Zeitlin [1] in 1993, and recapitulated very recently by Gurtas [5] in 2017.

The paper consists of the following parts: In Sect. 2 the mathematical background is explained. Section 3 presents some new results. Finally, Sect. 4 depicts some possible future steps.

2 Mathematical Background

In this section first we recall two classic theorems on constructibility. Then an algebraic formula will be shown by using previous work.

2.1 Constructibility

In AGA the key step is algebraization of the setup of a planar geometry statement. In most theorems in AGA the classic Euclidean construction steps are translated into algebraic equations. There is, however, a proof on Morley's trisector theorem presented which assumes a non-Euclidean, cubic way to make it constructed, but, in fact, the way of construction is successfully avoided in the equations [2, p. 120].

It is well known (Gauß [19], see [6, 7]) that a regular n -gon is constructible by using compass and straightedge if and only if n is the product of a power of 2 and any number of distinct Fermat primes (including none). We recall that a

Fermat prime is a prime number of the form $2^{2^m} + 1$. By using this theorem the list of the constructible regular n -gons are:

$$n = 3, 4, 5, 6, 8, 10, 12, 15, 16, 17, 20, \dots$$

A generalization of this result (Pierpont, see [8]) by allowing an angle trisector as well (for example, origami folding steps), is that a regular n -gon is constructible if and only if

$$n = 2^r \cdot 3^s \cdot p_1 \cdot p_2 \cdots p_k,$$

where $r, s, k \geq 0$ and the p_i are distinct primes of form $2^t \cdot 3^u + 1$ [9]. The first constructible regular n -gons of this kind are

$$n = 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, \dots$$

We note that the second list contains the first list.

From the second list the case $n = 11$ is missing, and, as a natural consequence, there are many fewer scientific results known on regular 11-gons than for n -gons appearing in the lists.

2.2 An Algebraic Formula for the Vertices

In this part of the paper we derive a formula for the coordinates of the vertices of a regular n -gon.

From now on we assume that $n \geq 1$. The cases $n = 1, 2$ have actually no geometrical meaning, but they will be useful from the algebraic point of view.

In AGA the usual way to describe the points of a construction is to assign coordinates (x_i, y_i) for a given point P_i ($i = 0, 1, 2, \dots$). When speaking about a polygon, in many cases the first vertices are put into coordinates $P_0 = (0, 0)$ and $P_1 = (1, 0)$, and the other coordinates are described either by using exact rationals, or the coordinates are expressed as possible solutions of algebraic equations.

For example, when defining a square, $P_2 = (1, 1)$ and $P_3 = (0, 1)$ are appropriate, but for a regular triangle two equations for $P_2 = (x_2, y_2)$ are required, namely $x_2^2 + y_2^2 = 1$ and $(x_2 - 1)^2 + y_2^2 = 1$. It is easy to see that this equation system has two solutions, namely $x_2 = \frac{1}{2}, y_2 = \frac{\sqrt{3}}{2}$ and $x_2 = \frac{1}{2}, y_2 = -\frac{\sqrt{3}}{2}$. It is well known that there is no way in AGA to avoid such duplicates, unless the coordinates are rational. In other words, if both minimal polynomials of the coordinates are linear (or constant), then the duplicates can be avoided, otherwise not. Here, for x_2 we have $2x_2 - 1 (= 0)$, but for y_2 the minimal polynomial is $4y_2^2 - 3 (= 0)$. We remark that the minimal polynomials are irreducible over \mathbb{Z} .

Clearly, minimal polynomials of a regular n -gon with vertices $P_0 = (0, 0)$ and $P_1 = (1, 0)$ can play an important role here. The paper [1] suggests an algorithm to obtain the minimal polynomial $p_c(x)$ of $\cos(2\pi/n)$, based on the Chebyshev polynomials $T_j(x)$ of the first kind (see Algorithm 1).

Algorithm 1. Computing the minimal polynomial of $\cos(2\pi/n)$

```

1: procedure COS2PIOVERNMINPOLY( $n$ )
2:    $p_c \leftarrow T_n - 1$ 
3:   for all  $j \mid n \wedge j < n$  do
4:      $q \leftarrow T_j - 1$ 
5:      $r \leftarrow \gcd(p_c, q)$ 
6:      $p_c \leftarrow p_c/r$ 
7:   return SquarefreeFactorization( $p_c$ )

```

Table 1. List of minimal polynomials of $\cos(2\pi/n)$, $n \leq 17$

n	Minimal polynomial of $\cos(2\pi/n)$
1	$x - 1$
2	$x + 1$
3	$2x + 1$
4	x
5	$4x^2 + 2x - 1$
6	$2x - 1$
7	$8x^3 + 4x^2 - 4x - 1$
8	$2x^2 - 1$
9	$8x^3 - 6x + 1$
10	$4x^2 - 2x - 1$
11	$32x^5 + 16x^4 - 32x^3 - 12x^2 + 6x + 1$
12	$4x^2 - 3$
13	$64x^6 + 32x^5 - 80x^4 - 32x^3 + 24x^2 + 6x - 1$
14	$8x^3 - 4x^2 - 4x + 1$
15	$16x^4 - 8x^3 - 16x^2 + 8x + 1$
16	$8x^4 - 8x^2 + 1$
17	$256x^8 + 128x^7 - 448x^6 - 192x^5 + 240x^4 + 80x^3 - 40x^2 - 8x + 1$

Also, adding the equation $p_c(x)^2 + p_s(y)^2 = 1$ to the equation system, we have managed to describe a polynomial $p_s(y)$ such that $p_s(\sin(2\pi/n)) = 0$. Table 1 shows the minimal polynomials for $n \leq 17$.

It is clear, that—not considering the cases $n = 1, 2, 3, 4, 6$ —the number of roots of p_c is more than one, therefore the solution of the equation system $\{p_c(x) = 0, p_s(x) = 0\}$ is not unique. The number of solutions for $p_c(x) = 0$ depends on the degree of p_c , and—not considering the cases $n = 1, 2$ —the number of solutions for $p_s(x) = 0$ is two for each root of $p_c(x)$, therefore the number of solutions for $\{p_c(x) = 0, p_s(y) = 0\}$ is usually $2 \cdot \deg(p_c)$. As a result, the point

$$P = (\cos(2\pi/n), \sin(2\pi/n))$$

can be exactly determined by an algebraic equation in AGA only in case $n = 4$, as shown in Table 2.

Table 2. Degree of ambiguity for $(\cos(2\pi/n), \sin(2\pi/n))$, $3 \leq n \leq 13$

n	Degree
3	2
4	2
5	4
6	2
7	6
8	4
9	6
10	4
11	10
12	4
13	12

It seems to make sense that the degree of ambiguity (not considering the case $n = 4$) can be computed with Euler’s totient function, that is, the degree equals to $\varphi(n)$. Later we will give a short proof on this.

Now we are ready to set up additional formulas to describe the coordinates of the vertices of a regular n -gon, having its first vertices $P_0 = (0, 0)$ and $P_1 = (1, 0)$, and the remaining vertices $P_2 = (x_2, y_2), \dots, P_{n-1} = (x_{n-1}, y_{n-1})$ are to be found. By using consecutive rotations and assuming $x = \cos(2\pi/n), y = \sin(2\pi/n)$, we can claim that

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_{i-1} \\ y_{i-1} \end{pmatrix} = \begin{pmatrix} x & -y \\ y & x \end{pmatrix} \cdot \left(\begin{pmatrix} x_{i-1} \\ y_{i-1} \end{pmatrix} - \begin{pmatrix} x_{i-2} \\ y_{i-2} \end{pmatrix} \right)$$

and therefore

$$x_i = -xy_{i-1} + x_{i-1} + xx_{i-1} + yy_{i-2} - xx_{i-2}, \tag{1}$$

$$y_i = y_{i-1} + xy_{i-1} + yx_{i-1} - xy_{i-2} - yx_{i-2} \tag{2}$$

for all $i = 2, 3, \dots, n - 1$.

3 Example Statements

In this section we present some statements on regular polygons that can be obtained by using the formulas from the previous section.

3.1 Lengths in a Regular Pentagon

A basic example on how our computation works will be discussed first.

Theorem 1. Consider a regular pentagon (Fig. 1) with vertices P_0, P_1, \dots, P_4 . Let $A = P_0, B = P_2, C = P_1, D = P_3, E = P_0, F = P_2, G = P_1, H = P_4$. Let us define diagonals $d = AB, e = CD, f = EF, g = GH$ and intersection points $R = d \cap e, S = f \cap g$. Now, when the length of P_0P_1 is 1, then the length of RS is $\frac{3-\sqrt{5}}{2}$.

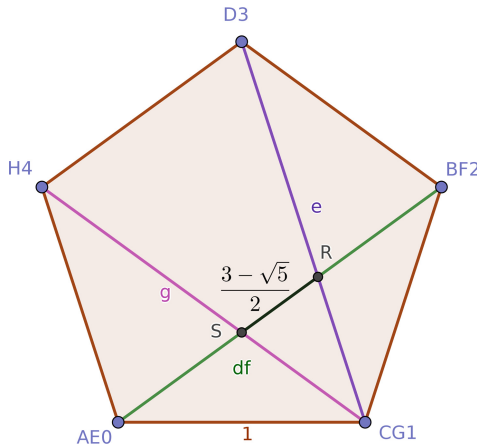


Fig. 1. A well-known theorem on a regular pentagon (for convenience we use only the indices of the points in the figure, that is, $0, 1, \dots, n - 1$ stand for P_0, P_1, \dots, P_{n-1} , respectively)

This result is well-known from elementary geometry, but here we provide a proof that uses the formulas developed in Sect. 2. We will use the variables x_0, x_1, x_2, x_3, x_4 for the x -coordinates of the vertices, y_0, y_1, y_2, y_3, y_4 for the y -coordinates, and x and y for the cosine and sine of $2\pi/5$, respectively. Points P_0 and P_1 will be $(0, 0)$ and $(1, 0)$.

By using Table 1 and Eqs. (1) and (2), we have the following hypotheses (we use the notion *hypotheses* here to describe the required geometric facts—such equations will be generated automatically later by GeoGebra in Sect. 3.3):

$$\begin{aligned}
 h_1 &= 4x^2 + 2x - 1 = 0, \\
 h_2 &= x^2 + y^2 - 1 = 0, \\
 h_3 &= x_0 = 0, \\
 h_4 &= y_0 = 0, \\
 h_5 &= x_1 - 1 = 0,
 \end{aligned}$$

$$\begin{aligned}
h_6 &= y_1 = 0, \\
h_7 &= -x_2 - xy_1 + x_1 + xx_1 + yy_0 - xx_0 = 0, \\
h_8 &= -y_2 + y_1 + xy_1 + yx_1 - xy_0 - yx_0 = 0, \\
h_9 &= -x_3 - xy_2 + x_2 + xx_2 + yy_1 - xx_1 = 0, \\
h_{10} &= -y_3 + y_2 + xy_2 + yx_2 - xy_1 - yx_1 = 0, \\
h_{11} &= -x_4 - xy_3 + x_3 + xx_3 + yy_2 - xx_2 = 0, \\
h_{12} &= -y_4 + y_3 + xy_3 + yx_3 - xy_2 - yx_2 = 0.
\end{aligned}$$

Since $R \in d$ and $R \in e$, we can claim that

$$h_{13} = \begin{vmatrix} x_0 & y_0 & 1 \\ x_2 & y_2 & 1 \\ x_r & y_r & 1 \end{vmatrix} = 0, h_{14} = \begin{vmatrix} x_1 & y_1 & 1 \\ x_3 & y_3 & 1 \\ x_r & y_r & 1 \end{vmatrix} = 0,$$

where $R = (x_r, y_r)$. Similarly,

$$h_{15} = \begin{vmatrix} x_0 & y_0 & 1 \\ x_2 & y_2 & 1 \\ x_s & y_s & 1 \end{vmatrix} = 0, h_{16} = \begin{vmatrix} x_1 & y_1 & 1 \\ x_4 & y_4 & 1 \\ x_s & y_s & 1 \end{vmatrix} = 0,$$

where $S = (x_s, y_s)$. Finally we can define the length $|RS|$ by stating

$$h_{17} = |RS|^2 - \left((x_r - x_s)^2 + (y_r - y_s)^2 \right) = 0.$$

From here we can go ahead with two methods:

1. We directly prove that $|RS| = \frac{3-\sqrt{5}}{2}$. As we will see, this actually does not follow from the hypotheses, because they describe a different case as well, shown in Fig. 2. That is, we need to prove a weaker thesis, namely that $|RS| = \frac{3-\sqrt{5}}{2}$ or $|RS| = \frac{3+\sqrt{5}}{2}$, which is equivalent to

$$\left(|RS| - \frac{3-\sqrt{5}}{2} \right) \cdot \left(|RS| - \frac{3+\sqrt{5}}{2} \right) = 0.$$

Unfortunately, this form is still not complete, because $|RS|$ is defined implicitly by using $|RS|^2$, that is, if $|RS|$ is a root, also $-|RS|$ will appear. The correct form for a polynomial t that has a root $|RS|$ is therefore

$$\begin{aligned}
t &= \left(|RS| - \frac{3-\sqrt{5}}{2} \right) \cdot \left(|RS| - \frac{3+\sqrt{5}}{2} \right) \\
&\quad \cdot \left(-|RS| - \frac{3-\sqrt{5}}{2} \right) \cdot \left(-|RS| - \frac{3+\sqrt{5}}{2} \right) = 0,
\end{aligned}$$

that is, after expansion,

$$t = (|RS|^2 - 3|RS| + 1) \cdot (|RS|^2 + 3|RS| + 1) = |RS|^4 - 7|RS|^2 + 1 = 0.$$

The proof the thesis is performed by showing the negation of t . This is accomplished by adding $t \cdot z - 1 = 0$ to the equation system $\{h_1, h_2, \dots, h_{17}\}$ and obtaining a contradiction. This approach is based on the Rabinowitsch trick, introduced by Kapur in 1986 (see [10]).

2. We can also discover the exact value of $|RS|$ by eliminating all variables from the ideal $\langle h_1, h_2, \dots, h_{17} \rangle$, except $|RS|$. This second method was suggested by Recio and Vélez in 1999 (see [11]).

The first method can be used only *after* one has a conjecture already. In contrast, the second method can be used *before* having a conjecture, namely, to find a conjecture *and* its proof at the same time.

For the first method we must admit that in AGA there is no way to express that the length of a segment is $\frac{3-\sqrt{5}}{2}$. Instead, we need to use its minimal polynomial, having integer (or rational) coefficients. Actually, $|RS|^2 - 3|RS| + 1$ is a minimal polynomial of both $\frac{3-\sqrt{5}}{2}$ and $\frac{3+\sqrt{5}}{2}$, and $|RS|^2 + 3|RS| + 1$ is of $-\frac{3-\sqrt{5}}{2}$ and $-\frac{3+\sqrt{5}}{2}$. In fact, given a length $|RS|$ in general, we need to prove that the equation $t = t_1 \cdot t_2 = 0$ is implied where t_1 and t_2 are the minimal polynomials of the expected $|RS|$ and $-|RS|$, respectively. Even if geometrically t_1 is implied, from the algebraic point of view $t_1 \cdot t_2$ is to be proven.

When using the second method, by using elimination (here we utilize computer algebra), we will indeed obtain that

$$\langle h_1, h_2, \dots, h_{17} \rangle \cap \mathbb{Q}[|RS|] = \langle |RS|^4 - 7|RS|^2 + 1 \rangle.$$

Star-Regular Polygons. Before going further, we need to explain the situation with the star-regular pentagon in Fig. 2. Here we need to mention that the equation $h_1 = 4x^2 + 2x - 1 = 0$ describes not only $\cos(2\pi/5)$ but also $\cos(2 \cdot 2\pi/5)$, $\cos(3 \cdot 2\pi/5)$ and $\cos(4 \cdot 2\pi/5)$, however, because of symmetry, the first and last, and the second and third values are the same. (We can think of these values as the projections of z_1, z_2, z_3, z_4 on the real axis, where

$$z_j = (\cos(2\pi/5) + i \sin(2\pi/5))^j = \cos(j \cdot 2\pi/5) + i \sin(j \cdot 2\pi/5),$$

$j = 1, 2, 3, 4$.)

That is, in this special case (for $n = 5$) h_1 is a minimal polynomial of $\text{Re } z_1 (= \text{Re } z_4)$ and $\text{Re } z_2 (= \text{Re } z_3)$. By considering the formulas (1) and (2) we can learn that the rotation is controlled by the vector (x, y) , where $2\pi/n$ is the external angle of the regular n -gon. When changing the angle to a double, triple, \dots , value, we obtain star-regular n -gons, unless the external angle describes a regular (or star-regular) m -gon ($m < n$).

This fact is well-known in the theory of regular polytopes [12], but let us illustrate this property by another example. When choosing $n = 6$, we have $h'_1 = 2x - 1 = 0$ that describes $\cos(2\pi/6) = \cos(5 \cdot 2\pi/6)$. Now by considering $z'_1, z'_2, z'_3, z'_4, z'_5$ where

$$z'_j = \cos(j \cdot 2\pi/6) + i \sin(j \cdot 2\pi/6),$$

$j = 1, 2, 3, 4, 5$, we can see that z'_2 can also be considered as a generator for $\cos(1 \cdot 2\pi/3)$ (when projecting it on the x -axis) since $2 \cdot 2\pi/6 = 1 \cdot 2\pi/3$. That is, $z'_2 (= \overline{z'_4})$ is not used when generating the minimal polynomial of $\cos(2\pi/6)$ (it occurs at the creation of the minimal polynomial of $\cos(2\pi/3)$), and this is the case also for z'_3 (because it is used for the minimal polynomial of $\cos(2\pi/2)$).

An immediate consequence is that z'_j is used as a generator in the minimal polynomial of $\cos(2\pi/6)$ if and only if j and 6 are coprimes, but since $\cos(2\pi/6) = \cos(5 \cdot 2\pi/6)$, only the first half of the indices j play a technical role. In general, when n is arbitrary, the number of technically used generators are $\varphi(n)/2$ (the other $\varphi(n)/2$ ones produce the same projections).

Finally, when considering the equation $x^2 + y^2 = 1$ as well, if $n \geq 3$, there are two solutions in y , hence the hypotheses describe *all* cases when j and n are coprimes (not just for the half of the cases, that is, for $1 \leq j \leq n/2$). Practically, the hypotheses depict not just the regular n -gon case, but also *all* star-regular n -gons. It is clear, after this chain of thoughts, that the number of cases is $\varphi(n)$ (which is the number of positive coprimes to n , less than n). From this immediately follows that the degree of ambiguity for $(\cos(2\pi/n), \sin(2\pi/n))$ is *exactly* $\varphi(n)$.

Also, it is clear that there exists essentially only one regular 5-gon and one star 5-gon (namely, $\{5/2\}$, when using the Schäfli symbol, see [12]). But these are just two different cases. The other two ones, according to $\varphi(5)$, are symmetrically equivalent cases. The axis of symmetry is the x -axis in our case.

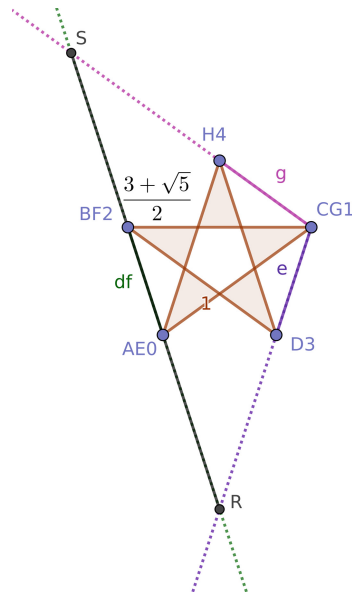


Fig. 2. A variant of the theorem in a star-regular pentagon

On the other hand, by using our method, it is not always possible to distinguish between these $\varphi(n)$ cases. In general we have multiple cases:

1. $t = |RS|^2 - c$ where c is a rational. In this case clearly $|RS| = \sqrt{c}$ follows.
2. Otherwise, the resulting polynomial t is a product of two polynomials $t_1, t_2 \in \mathbb{Q}[|RS|]$, and the half of the union of their roots are positive, while the others are negative. On the other hand, the positive roots can be placed in several combinations in t_1 and t_2 in general:

- (a) In our concrete example there are two positive roots in t_1 and two negative ones in t_2 . When considering similar cases, the positive roots can always occur in, say t_1 , and the negative roots then in t_2 . Albeit the elimination delivers the product $t = t_1 \cdot t_2$, clearly t_2 cannot play a geometrical role, therefore t_1 can be concluded.

However, if t_1 contains more than one (positive) root, those roots cannot be distinguished. This is the case in our concrete example as well.

- (b) In general, t_1 may contain a few positive solutions, but t_2 may also contain some other ones. In such cases the positive solutions in t_1 and t_2 cannot be distinguished from each other.

Such an example is the polynomial $t = t_1 \cdot t_2$ where $t_1 = |RS|^2 - |RS| - 1$ and $t_2 = |RS|^2 + |RS| - 1$. It describes the length of the diagonal of a regular (star-) pentagon, namely both lengths $\frac{\sqrt{5} \pm 1}{2}$. Here t_1 contains one of the positive roots, namely $\frac{\sqrt{5} + 1}{2}$, while t_2 the other one, $\frac{\sqrt{5} - 1}{2}$. At the end of the day, only t can be concluded, none of its factors can be dropped because both contain geometrically useful data.

Further investigation of the structure of several other statements on regular polygons is still a work-in-progress. Here we refer to the papers [13, 14] that already introduced the basic notions for some future work.

3.2 Lengths in a Regular 11-gon

In Sect. 2 we mentioned that scientific results on a regular 11-gon are not very well-known because it is not constructible by typical means. Here we show some—for us, previously unknown—results that have been obtained by our method.

Theorem 2. *A regular 11-gon (having sides of length 1) is defined by points A, B, C, \dots, J, K . Diagonals CE, CF, CG, CH, DF, DK and HK are drawn. Then intersection points L, M, N and O are defined as shown in Fig. 3. The following properties hold:*

1. $b = c$,
2. $d = e$,
3. triangles CLM and CON are congruent,
4. $a = l$ (that is, $|AB| = |DL|$).
5. Let $P = BJ \cap CD$. Then $|OP| = \sqrt{3}$.

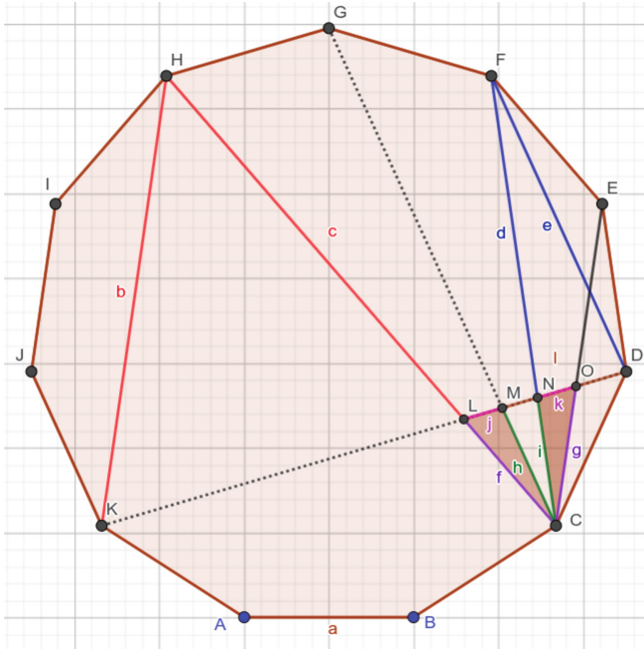


Fig. 3. Some properties of a regular 11-gon

6. $|BO| \neq \frac{5}{3}$ (but it is very close to it, $|BO| \approx 1,66686\dots$, it is a root of the polynomial $x^{10} - 16x^8 + 87x^6 - 208x^4 + 214x^2 - 67 = 0$).

Proof. By using the method described above, all of these statements can be mechanically proved in a straightforward way.

3.3 Implementation in GeoGebra

The above mentioned theorems can also be proven with GeoGebra, a dynamic mathematics tool, freely available for download or for on-line use. Practically, the formulas derived in Sect. 2 have been programmed for GeoGebra by the author to extend its dynamic geometry capabilities. Further results containing statements on regular polygons can be found at <https://www.geogebra.org/m/AXd5ByHX>, where some of the statements shown in this paper are visually demonstrated, and a proof can also be obtained.

Here “proof” means that the computer symbolically proves the result, but the steps of the derivation are not shown—the number of steps can be extremely large, because the applied methods (here Gröbner basis computation) can consist of millions of steps. Instead, a yes/no result is shown, and, if possible, non-degeneracy conditions are also given.

Here we are illustrating how a GeoGebra user would use the interface to build a figure and then check whether a conjecture is true or false, by constructing the figure in Theorem 2, statement 1.

1. The user creates a regular 11-gon by choosing the *Regular Polygon* tool from the *Toolbar*, by defining two points A and B and entering the number of vertices, 11. (It is important that points A and B are free, that is, they should not lie on the axes.)
2. The user connects C and H , D and K , and H and K , by using the *Segment* tool.
3. The intersection point L of segments CH and DK is to be created by using the *Intersect* tool.
4. Again, by using the *Segment* tool, segment HL is to be created.
5. Now segments HL and HK can be compared by using the *Relation* tool. GeoGebra reports numerical equality. Then, by clicking the “More...” button, after some seconds the numerical check will be completed with a symbolical check, namely, that HL has the same length as HK under the condition “the construction is not degenerate”. (In general, degeneracy means that there is a “small” set where the statement is not true. In many cases, that problematic set can be defined in geometric terms, like “a triangle is degenerate”, or “some given segments do not have intersections”. But in this particular case GeoGebra did not find any simple geometric terms, so it leaves the information in the form “the construction is not degenerate”. See [15, Chap. 6, Sect. 4] for more on degeneracy conditions.)

In addition, on the one hand, to prove statement 5 one needs to type the command `Relation(OP2, 3AB2)` in the *Input Bar*, and—after getting promising results on the numerical check—the “More...” button will compute the symbolic proof again.

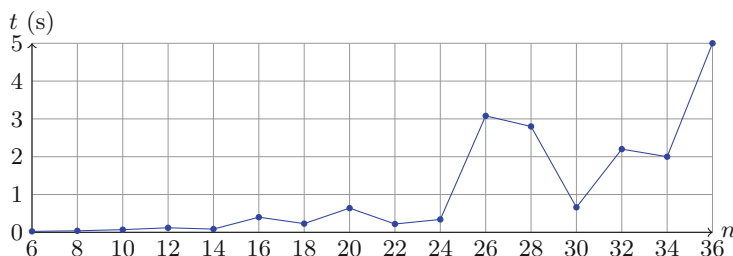


Fig. 4. Timings of proofs of a simple theorem for various n

On the other hand, disproving false statements is also possible in GeoGebra. For example, the statement 6 can be checked by typing `ProveDetails(5/3 · AB == BO)` and GeoGebra reports in its *Algebra View* that the statement is {false}. Here we cannot use the *Relation* command, because the numerical check

compares several digits of the two segments, and its negative result is already enough to lose interest to prove/disprove the statement in general. This behavior in GeoGebra, can be, however, misleading sometimes. (See [16, p. 62, Fig. 18] for an example.)

Here we highlight that our augmentation to GeoGebra consists of automatically creating the hypotheses h_1, \dots by using Algorithm 1 and Eqs. 1 and 2.

More details about GeoGebra’s Automated Reasoning Tools can be found in [17].

4 Conclusion and Future Work

We presented a method that helps obtaining various new theorems on regular polygons, based on the work of [1, 3, 11].

The obtained theorems were manually found, but systematic methods can also be defined. One possible approach can be found in the software tool `RegularNGons` [18] that looks for “nicely looking” statements by using elimination.

We admit that for large n our computations can be much slower than for small numbers. That is, finding new theorems in regular n -gons may be computationally infeasible if n becomes too large. As a particular example, Fig. 4 shows timings for the general

Theorem 3. *Let n be an even positive number, and let us denote the vertices of a regular n -gon by P_0, P_1, \dots, P_{n-1} . Let $A = P_0$, $B = P_1$, $C = P_2$, $D = P_{n/2}$. Moreover, let $R = AB \cap CD$. If $|AB| = 1$, then $|P_1R| = 1$.*

We note that in GeoGebra, by default, the computations will be aborted after 5 seconds. However, for computing several theorems on regular polygons, this timeout is not exceeded. Also, in Fig. 4, higher values for n do not necessarily imply slower computations, for instance the case $n = 30$ is computed in 660 ms, but for $n = 28$ the required time is 2800 ms.

Let us finally remark that finding new theorems on regular polygons may open new horizons on detecting truth on parts—here we recall the papers [13, 14].

Acknowledgments. The author was partially supported by a grant MTM2017-88796-P from the Spanish MINECO (Ministerio de Economía y Competitividad) and the ERDF (European Regional Development Fund).

Many thanks to Tomás Recio and Francisco Botana for their valuable comments and suggestions.

References

1. Watkins, W., Zeitlin, J.: The minimal polynomial of $\cos(2\pi/n)$. *Am. Math. Mon.* **100**, 471–474 (1993)
2. Chou, S.C.: *Mechanical Geometry Theorem Proving*. Springer, Dordrecht (1987)
3. Wu, W.T.: *On the Decision Problem and the Mechanization of Theorem-Proving in Elementary Geometry* (1984)

4. Lehmer, D.H.: A note on trigonometric algebraic numbers. *Am. Math. Mon.* **40**, 165–166 (1933)
5. Gurtas, Y.Z.: Chebyshev polynomials and the minimal polynomial of $\cos(2\pi/n)$. *Am. Math. Mon.* **124**, 74–78 (2017)
6. Wantzel, P.: Recherches sur les moyens de reconnaître si un problème de géométrie peut se résoudre avec la règle et le compas. *Journal de Mathématiques Pures et Appliquées* **1**, 366–372 (1837)
7. Sethuraman, B.: *Rings, Fields, and Vector Spaces: An Introduction to Abstract Algebra via Geometric Constructibility*. Springer, New York (1997). <https://doi.org/10.1007/978-1-4757-2700-5>
8. Pierpont, J.: On an undemonstrated theorem of the disquisitiones arithmeticae. *Bull. Am. Math. Soc.* **2**, 77–83 (1895)
9. Gleason, A.M.: Angle trisection, the heptagon, and the triskaidecagon. *Am. Math. Mon.* **95**, 185–194 (1988)
10. Kapur, D.: Using Gröbner bases to reason about geometry problems. *J. Symb. Comput.* **2**, 399–408 (1986)
11. Recio, T., Vélez, M.P.: Automatic discovery of theorems in elementary geometry. *J. Autom. Reason.* **23**, 63–82 (1999)
12. Coxeter, H.S.M.: *Regular Polytopes*, 3rd edn. Dover Publications, New York (1973)
13. Kovács, Z., Recio, T., Sólyom-Gecse, C.: Automatic rewrites of input expressions in complex algebraic geometry provers. In: Narboux, J., Schreck, P., Streinu, E. (eds.) *Proceedings of ADG 2016*, Strasbourg, France, pp. 137–143 (2016)
14. Kovács, Z., Recio, T., Vélez, M.P.: Detecting truth, just on parts. *CoRR abs/1802.05875* (2018)
15. Cox, D., Little, J., O’Shea, D.: *Ideals Varieties, and Algorithms*. Springer, New York (2007). <https://doi.org/10.1007/978-0-387-35651-8>
16. Kovács, Z.: Automated reasoning tools in GeoGebra: a new approach for experiments in planar geometry. *South Bohemia Math. Lett.* **25**(1), 48–65 (2017). <http://home.pf.jcu.cz/~sbml/wp-content/uploads/Kovacs.pdf>
17. Kovács, Z., Recio, T., Vélez, M.P.: gg-art-doc (GeoGebra Automated Reasoning Tools. A tutorial). A GitHub project (2017). <https://github.com/kovzol/gg-art-doc>
18. Kovács, Z.: RegularNGons. A GitHub project (2018). <https://github.com/kovzol/RegularNGons>
19. Gauß, C.F.: *Disquisitiones Arithmeticae*. Fleischer, Lipsiae (Leipzig) (1801)



Gröbner Bases of Modules and Faugère's F_4 Algorithm in Isabelle/HOL

Alexander Maletzky¹  and Fabian Immler² 

¹ RISC, Johannes Kepler Universität Linz, Linz, Austria
alexander.maletzky@risc.jku.at

² Institut für Informatik, Technische Universität München, Munich, Germany
immler@in.tum.de

Abstract. We present an elegant, generic and extensive formalization of Gröbner bases, an important mathematical theory in the field of computer algebra, in Isabelle/HOL. The formalization covers all of the essentials of the theory (polynomial reduction, S-polynomials, Buchberger's algorithm, Buchberger's criteria for avoiding useless pairs), but also includes more advanced features like reduced Gröbner bases. Particular highlights are the first-time formalization of Faugère's matrix-based F_4 algorithm and the fact that the entire theory is formulated for modules and submodules rather than rings and ideals. All formalized algorithms can be translated into executable code operating on concrete data structures, enabling the certified computation of (reduced) Gröbner bases and syzygy modules.

1 Introduction

Since their origins in Buchberger's PhD thesis [3], *Gröbner bases* have become one of the most powerful and most widely used tools in computer algebra. Their importance stems from the fact that they generalize, at the same time, Gauss' algorithm for solving systems of linear equations and Euclid's algorithm for computing the GCD of univariate polynomials. Gröbner bases enable the effective, systematic solution of a variety of problems in polynomial ideal theory, ranging from the decision of ideal/submodule membership and congruence, the solution of systems of algebraic equations, to as far as automatic theorem proving. We refer the interested reader to any standard textbook about Gröbner bases, like [15], for a more thorough account on the subject.

The main achievement we report on in this paper is the first-time formalization of said theory in the proof assistant Isabelle/HOL [19]. Although Gröbner bases have been formalized in other proof assistants already, our work features the, to the best of our knowledge, first computer-certified implementation of Faugère's F_4 algorithm [7] for computing Gröbner bases by matrix reductions,

A. Maletzky—The research was funded by the Austrian Science Fund (FWF): P 29498-N31.

F. Immler—The research was funded by DFG Koselleck Grant NI 491/16-1.

as well as the (again to the best of our knowledge) first-time formal treatment of the theory in the more general setting of *modules* and *submodules* rather than rings and ideals.

Summarizing, the highlights of our elaboration are:

- an abstract view of power-products that allows us to represent power-products by functions of type $\text{nat} \Rightarrow \text{nat}$ with finite support (Sect. 2.1);
- the proof of Buchberger’s theorem about the connection between Gröbner bases and S-polynomials (Sect. 3.2);
- a generic algorithm schema for computing Gröbner bases, of which both Buchberger’s algorithm and Faugère’s F_4 algorithm are instances (Sect. 3.3);
- the implementation of Buchberger’s criteria for increasing the efficiency of said algorithm schema;
- the formally verified implementation of the F_4 algorithm (Sect. 4);
- the definition and a constructive proof of existence and uniqueness of reduced Gröbner bases (Sect. 5.1);
- a formally verified algorithm for computing Gröbner bases of syzygy modules (Sect. 5.2);
- the proper set-up of Isabelle’s code generator to produce certified executable code.

Because of the size constraints imposed on this exposition, it is impossible to present all aspects of our formalization in detail. Therefore, an extended version of the paper is available as a technical report [17].

An Isabelle2017-compatible version of the formalization presented in this paper is available online [13]. Furthermore, a big portion of the formalization has already been added to the development version of the Archive of Formal Proofs (AFP). Note also that there is a Gröbner-bases entry in the release version of the AFP [12] (which will be replaced by the one in the development version upon the next release of Isabelle), but it lacks many features compared to [13]. For the sake of clarity and readability, various technical details of the formalization are omitted or presented in simplified form in this paper.

1.1 Related Work

Gröbner bases (without F_4 and only for rings and ideals) have been formalized in a couple of other proof assistants already, among them being Coq [14, 20, 24], Mizar [21], ACL2 [18] and Theorema [4, 6, 16]. See [16, 17] for a more thorough comparison of the different formalizations.

Apart from that, Gröbner bases have been successfully employed by various proof assistants (among them HOL [11] and Isabelle/HOL [5]) as a means for proving universal propositions over rings. In a nutshell, this proceeds by showing that the system of polynomial equalities and inequalities arising from refuting the original formula is unsolvable, which in turn is accomplished by finding a combination of these polynomials that yields a non-zero constant polynomial – and this is exactly what Gröbner bases can do. The computation of Gröbner

bases, however, is taken care of by a “black-box” ML program whose correct behavior is irrelevant for the correctness of the overall proof step, since the obtained witness is independently checked by the trusted inference kernel of the system. The work described in this paper is orthogonal to [5] in the sense that it formalizes the theory underlying Gröbner bases and proves the total correctness of the algorithm, which is not needed in [5].

Our work builds upon existing formal developments of multivariate polynomials [23] (to which we also contributed) and abstract rewrite systems [22], and F_4 in addition builds upon Gauss-Jordan normal forms of matrices [25].

2 Multivariate Polynomials

Gröbner bases are concerned with (vectors of) commutative multivariate polynomials over fields; therefore, before presenting our formalization of Gröbner bases theory, we first have to spend some words on the formalization of multivariate polynomials we build upon.

The formal basis of multivariate polynomials are so-called *polynomial mappings*, originally formalized by Haftmann *et al.* [10], extended by Bentkamp [2], and now part of the AFP-entry *Polynomials* [23] in the development version of the Archive of Formal Proofs. A polynomial mapping is simply a function of type $\alpha \Rightarrow \beta :: \text{zero}$ with finite support, i. e., all but finitely many arguments are mapped to 0¹. In Isabelle/HOL, as well as in the remainder of this paper, the type of polynomial mappings is called `poly_mapping` and written in infix form as $\alpha \Rightarrow_0 \beta$, where β is tacitly assumed to belong to type class `zero`. The importance of type `poly_mapping` stems from the fact that not only polynomials, but also power-products (i. e. products of indeterminates, like $x_0^3 x_1^2$) can best be thought of as terms of this type: in power-products, indeterminates are mapped to their exponents (with only finitely many being non-zero), and in polynomials, power-products are mapped to their coefficients (again only finitely many being non-zero). Hence, a (scalar) polynomial would typically have the type $(\chi \Rightarrow_0 \text{nat}) \Rightarrow_0 \beta$, where χ is the type of the indeterminates.

2.1 Power-Products

Instead of fixing the type of power-products to $\chi \Rightarrow_0 \text{nat}$ throughout the formalization, we opted to develop the theory slightly more abstractly: power-products are not fixed to type $\chi \Rightarrow_0 \text{nat}$, but they can be of *arbitrary* type, as long as the type belongs to a certain type class that allows us to prove all key results of the theory. Said type class is called `graded_dickson_powerprod` and is defined as

```
class graded_dickson_powerprod = cancel_comm_monoid_mult + dvd +
fixes lcm :: " $\alpha \Rightarrow \alpha \Rightarrow \alpha$ "
```

¹ $\beta :: \text{zero}$ is a type-class constraint on type β , stipulating that there must be a distinguished constant 0 of type β . See [9] for information on type classes in Isabelle/HOL.


```

assumes dvd_lcm: "s dvd (lcm s t)"
assumes lcm_dvd: "s dvd u  $\implies$  t dvd u  $\implies$  (lcm s t) dvd u"
assumes lcm_comm: "lcm s t = lcm t s"
assumes times_eq_one: "s * t = 1  $\implies$  s = 1"

```

The base class of `graded_dickson_powerprod` is the class of *cancellative commutative multiplicative monoids* which in addition feature a *divisibility relation* (infix `dvd`). Furthermore, types belonging the class must also provide a function called `lcm` that possesses the usual properties of *least common multiple*, and obey the law that a product of two factors can only be 1 if both factors are 1, i.e. 1 is the only invertible element.

Under certain weak conditions on χ , the type $\chi \Rightarrow_0 \text{nat}$ belongs to type class `graded_dickson_powerprod`. In particular, `nat` fulfills these conditions, meaning that power-products can be represented conveniently as objects of type $\text{nat} \Rightarrow_0 \text{nat}$ (with an infinite supply of indeterminates) in actual computations, without having to a-priori introduce dedicated types for univariate/bivariate/trivariate/... polynomials². There are, however, some intricacies when allowing infinite sets of indeterminates in connection with Gröbner bases, especially regarding the termination of algorithms. How our formalization handles these intricacies is described in [17].

In order to formalize Gröbner bases we need to fix an *admissible order relation* \preceq on power-products: \preceq is called admissible if and only if it is linear, 1 is the least element and multiplication is monotonic in both arguments. Since there are infinitely many admissible relations and we did not want to restrict the formalization to a particular one, we parametrized all definitions, theorems and algorithms over such a relation through the use of a *locale* [1], called `gd_powerprod`.

2.2 Polynomials

Having described our formalization of power-products, we now turn to polynomials. In fact, most definitions related to, and facts about, multivariate polynomials that are required by our Gröbner bases formalization were already formalized [2, 10]: addition, multiplication, coefficient-lookup (called `lookup` in the formal theories but denoted by the more intuitive `coeff` in the remainder) and support (called `keys`). These things are all pretty much standard, so we do not go into more detail here. We only emphasize that henceforth α is the type of power-products, i.e. is tacitly assumed to belong to type-class `graded_dickson_powerprod`.

What is certainly more interesting is the way how we represent vectors of polynomials: since we formulate the theory of Gröbner bases in the context of free modules over polynomial rings over fields, i.e. for structures of the form $K[x_1, \dots, x_n]^k$, we need to specify what the formal type of such structures is in our formalization. Any $f \in K[x_1, \dots, x_n]^k$ can be written as a K -linear combination of *terms* of the form $t e_i$, where t is a power-product and e_i is the i -th canonical basis vector of K^k (for $0 \leq i < k$). So, vector-polynomials are objects

² This representation is also suggested in [10].

of type $(\alpha \times \kappa) \Rightarrow_0 \beta$ in the formalization, where κ is the type of component-indices (e.g. `nat`). A term $t e_i$, thus, is represented by the pair (t, i) .

Example 1. Consider the two-dimensional vector of polynomials

$$p = \begin{pmatrix} x^2 - xy \\ 2y + 3 \end{pmatrix} = 1 \cdot x^2 \underbrace{\begin{pmatrix} 1 \\ 0 \end{pmatrix}}_{=e_0} - 1 \cdot xy \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 2 \cdot y \underbrace{\begin{pmatrix} 0 \\ 1 \end{pmatrix}}_{=e_1} + 3 \cdot 1 \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Then p is represented by the polynomial mapping that maps $(x^2, 0)$ to 1, $(xy, 0)$ to -1 , $(y, 1)$ to 2, $(1, 1)$ to 3, and all other pairs to 0.

As Example 1 shows, for representing a k -dimensional polynomial κ does not need to have exactly k elements, but only *at least* k elements. This saves us from introducing dedicated types for 1, 2, 3, ... dimensions in computations, as we can use `nat` throughout. Nonetheless, we do not fix κ to `nat`, because in some situations it is still desirable to state definitions or theorems only for ordinary scalar polynomials while reusing notions introduced for vector-polynomials, which can be achieved easily by instantiating κ by the unit type `unit`³.

We now need to extend the order relation \preceq on power-products to a *term order* \preceq_t on terms of type $\alpha \times \kappa$: similar to admissible orders on power-products, \preceq_t only needs to satisfy certain properties. And just as for admissible orders we again employ a locale to parametrize all subsequent definitions and lemmas over any such admissible relation:

```
locale gd_term =
  gd_powerprod ord ord_strict +
  ord_term_lin: linorder ord_term ord_term_strict
  for ord :: " $\alpha \Rightarrow \alpha :: \text{graded\_dickson\_powerprod} \Rightarrow \text{bool}$ " (infix " $\preceq$ " 50)
  and ord_strict (infix " $\prec$ " 50)
  and ord_term :: " $(\alpha \times \kappa) \Rightarrow (\alpha \times \kappa :: \text{wellorder}) \Rightarrow \text{bool}$ " (infix " $\preceq_t$ " 50)
  and ord_term_strict (infix " $\prec_t$ " 50) +
  assumes stimes_mono: " $v \preceq_t w \implies t \otimes v \preceq_t t \otimes w$ "
  assumes ord_termI: " $\text{fst } v \preceq \text{fst } w \implies \text{snd } v \preceq \text{snd } w \implies v \preceq_t w$ "
```

So, `gd_term` extends `gd_powerprod` by \preceq_t and \prec_t , requires κ to be well-ordered by \preceq , and requires \preceq_t to be a linear ordering satisfying the two axioms *stimes-mono* and *ord-termI*. \otimes is defined as

```
definition stimes :: " $\alpha \Rightarrow (\alpha \times \kappa) \Rightarrow (\alpha \times \kappa)$ " (infixl " $\otimes$ " 75)
  where "stimes t v = (t * fst v, snd v)"
```

i.e. it multiplies the power-product of its second argument with its first argument. `fst` and `snd` are built-in Isabelle/HOL functions that access the first and second, respectively, component of a pair.

The two most important instances of \preceq_t are so-called *position-over-term* (POT) and *term-over-position* (TOP) orders. In POT, when comparing two pairs (s, i) and (t, j) one first compares the component-indices i and j w. r. t. the ordering \preceq on κ , and if they are equal, the power-products s and t w. r. t. \preceq on α ;

³ `unit` contains only one single element, so $\alpha \times \text{unit}$ is isomorphic to α .

in TOP, one first compares power-products and afterward components. Therefore, both POT and TOP are actually lexicographic orderings, and the reader can easily convince himself that indeed both satisfy *stimes-mono* and *ord-termI*. POT is particularly important for computing Gröbner bases of syzygy modules, described in Sect. 5.2.

Based on \otimes we introduce multiplication of a polynomial by a coefficient $c :: \beta$ and a power-product $t :: \alpha$ in the obvious way: all coefficients are multiplied by c , and all terms are multiplied by t via \otimes . The resulting function is called `monom_mult` and will feature a prominent role in polynomial reduction, defined in Sect. 3.2.

Having \preceq_t fixed in the context, we can introduce the all-important concepts of *leading term* (called `lt`), *leading power-product* (`lp`), *leading coefficient* (`lc`) and *tail* (`tail`): `lt(p)`⁴ returns the greatest (w.r.t. \preceq_t) term appearing in the polynomial p with non-zero coefficient, `lp(p)` returns the power-product of `lt(p)`, `lc(p)` returns the coefficient of `lt(p)` in p , and `tail(p)` returns the “tail” of p , i.e. sets the leading coefficient of p to 0 and leaves all other coefficients unchanged.

Example 2. Let p as in Example 1, and assume \preceq is the lexicographic order relation with $y \prec x$. If \preceq_t is the POT-extension of \preceq , then `lt(p) = (y, 1)` and `lc(p) = 2`; if it is the TOP-extension, then `lt(p) = (x2, 0)` and `lc(p) = 1`.

Before we finish this section, we introduce two more notions related to polynomials that we will need later. First, we extend divisibility of power-products of type α to divisibility of terms of type $\alpha \times \kappa$:

definition `dvd_term` :: " $(\alpha \times \kappa) \Rightarrow (\alpha \times \kappa) \Rightarrow \text{bool}$ " (infix "dvd_t" 50)
where "`dvd_term u v` \longleftrightarrow (`snd u = snd v` \wedge (`fst u` `dvd` (`fst v`)))"

Finally, we also introduce the notion of a *submodule* generated by a set B of polynomials as the smallest set that contains $B \cup \{0\}$ and that is closed under addition and under `monom_mult`, denoted by `pmdl(B)`.

3 Gröbner Bases and Buchberger’s Algorithm

From now on we tacitly assume, unless stated otherwise, that all definitions and theorems are stated in context `gd_term` (meaning that all parameters and axioms of `gd_term` are available for use, that κ belongs to type-class `wellorder`, and that α belongs to type-class `graded_dickson_powerprod`), and that the type of coefficients is $\beta :: \text{field}$. Readers familiar with Gröbner bases in rings but not with Gröbner bases in modules will spot only few differences to the ring-setting, as the module-setting parallels the other in many respects.

⁴ We take the liberty to use standard mathematical notation here, because Isabelle syntax does not integrate so well with informal text.

3.1 Polynomial Reduction

The first crucial concept of Gröbner bases theory is *polynomial reduction*, i. e. a binary relation `red` parametrized over sets of polynomials. In Isabelle/HOL, the corresponding definitions look as follows:

```
definition red_single :: "(( $\alpha \times \kappa \Rightarrow_0 \beta$ )  $\Rightarrow$  ( $\alpha \times \kappa \Rightarrow_0 \beta$ )  $\Rightarrow$  ( $\alpha \times \kappa \Rightarrow_0 \beta$ )  $\Rightarrow$   $\alpha \Rightarrow$ 
  bool"
where "red_single p q f t  $\longleftrightarrow$  (f  $\neq$  0  $\wedge$  coeff p (t  $\otimes$  lt f)  $\neq$  0  $\wedge$ 
  q = p - monom_mult ((coeff p (t  $\otimes$  lt f)) / lc f) t f)"
```

```
definition red :: "(( $\alpha \times \kappa \Rightarrow_0 \beta$ ) set  $\Rightarrow$  ( $\alpha \times \kappa \Rightarrow_0 \beta$ )  $\Rightarrow$  ( $\alpha \times \kappa \Rightarrow_0 \beta$ )  $\Rightarrow$  bool"
where "red F p q  $\longleftrightarrow$  ( $\exists f \in F. \exists t. \text{red\_single } p \ q \ f \ t$ )"
```

`red_single(p, q, f, t)` expresses that polynomial p reduces to q modulo the individual polynomial f , multiplying f by power-product t . Likewise, `red(F)(p, q)` expresses that p reduces to q modulo the set F in one step; hence, `red(F)` is the actual *reduction relation* modulo F , and `(red(F))**` denotes its reflexive-transitive closure. In in-line formulas we will use the conventional infix notations $p \rightarrow_F q$ and $p \rightarrow_F^* q$ instead of the more clumsy `red(F)(p, q)` and `(red(F))**(p, q)`, respectively. `is_red(F)(p)`, finally, expresses that p is reducible modulo F .

After introducing the above notions, we are able to prove, for instance, that \rightarrow_F is well-founded. This justifies implementing a function `trd`, which totally reduces a given polynomial p modulo a finite list fs of polynomials and, thus, computes a *normal form* of p modulo fs :

```
lemma trd_red_rtrancl: "(red (set fs))** p (trd fs p)"
```

```
lemma trd_irred: " $\neg$  is_red (set fs) (trd fs p)"
```

So, `trd` really computes *some* normal form of the given polynomial modulo the given list of polynomials. But note that normal forms are in general not unique, i. e. the reduction relation modulo an arbitrary set F is in general not confluent.

3.2 Gröbner Bases

The fact that \rightarrow_F is in general not confluent motivates the definition of a *Gröbner basis* as a set F that does induce a confluent reduction relation:

```
definition is_Groebner_basis :: "(( $\alpha \times \kappa \Rightarrow_0 \beta$ ) set  $\Rightarrow$  bool"
where "is_Groebner_basis F  $\longleftrightarrow$  is_confluent (red F)"
```

where `is_confluent` is the predicate-analogue of CR from *Abstract-Rewriting* [22].

The definition of Gröbner bases cannot be used to decide whether a given set is a Gröbner basis or not, since infinitely many polynomials and possible reduction-paths would have to be investigated. Fortunately, however, Buchberger proved an alternative characterization of Gröbner bases that indeed gives rise to a decision algorithm. This alternative characterization is based on so-called *S-polynomials*, defined as follows:

```

definition spoly :: " $((\alpha \times \kappa) \Rightarrow_0 \beta) \Rightarrow ((\alpha \times \kappa) \Rightarrow_0 \beta) \Rightarrow ((\alpha \times \kappa) \Rightarrow_0 \beta)$ "
  where "spoly p q = (if snd (lt p) = snd (lt q) then
    let l = lcm (lp p) (lp q) in
      (monom_mult (1 / (lc p)) (1 / (lp p)) p) -
      (monom_mult (1 / (lc q)) (1 / (lp q)) q)
    else 0)"

```

The S-polynomial of p and q is precisely the difference of the *critical pair* of p and q , so it roughly corresponds to the smallest element where reduction modulo $\{p, q\}$ might diverge. It first tests whether the component-indices of $\text{lt}(p)$ and $\text{lt}(q)$ agree; if not, it gives 0, otherwise it returns a difference of multiples of p and q where the leading terms cancel.

The Main Theorem of the theory of Gröbner bases states that a set F is a Gröbner basis if the S-polynomials of all pairs of elements in F can be reduced to 0 modulo F , i.e.

```

theorem Buchberger_thm_finite:
  assumes "finite F"
  assumes " $\bigwedge p q. p \in F \implies q \in F \implies (\text{red } F)^{**} (\text{spoly } p \ q) \ 0$ "
  shows "is_Groebner_basis F"

```

Our proof of Theorem *Buchberger-thm-finite* exploits various results about well-founded binary relations formalized in [22]. Thanks to that theorem, for deciding whether a finite set is a Gröbner basis it suffices to compute normal forms of finitely many S-polynomials and check whether they are 0. In fact, also the converse of *Buchberger-thm-finite* holds, so whenever one finds a non-zero normal form of an S-polynomial, the given set cannot be a Gröbner basis.

3.3 An Algorithm Schema for Computing Gröbner Bases

Theorem *Buchberger-thm-finite* not only yields an algorithm for deciding whether a given finite set F is a Gröbner basis or not, but also an algorithm for *completing* F to a Gröbner basis in case it is not. This algorithm, called *Buchberger's algorithm*, is a classical critical-pair/completion algorithm that repeatedly checks whether all S-polynomials reduce to 0, and if not, adds their non-zero normal forms to the basis to make them reducible to 0 (note that for every $p \neq 0$ we have $p \rightarrow_{\{p\}} 0$); the new elements that are added to the basis obviously do not change the submodule generated by the basis, since reduction preserves submodule membership.

In our formalization, we did not directly implement Buchberger's algorithm, but instead considered a more general algorithm schema first, of which both Buchberger's algorithm and Faugère's F_4 algorithm (cf. Sect. 4) are particular instances. This algorithm schema is called `gb_schema_aux` and implemented by the following tail-recursive function:

```

function gb_schema_aux :: " $((\alpha \times \kappa) \Rightarrow_0 \beta)$  list  $\Rightarrow$ 
   $((\alpha \times \kappa) \Rightarrow_0 \beta) \times ((\alpha \times \kappa) \Rightarrow_0 \beta)$  list  $\Rightarrow$ 
   $((\alpha \times \kappa) \Rightarrow_0 \beta)$  list" where
  "gb_schema_aux bs [] = bs" |
  "gb_schema_aux bs ps =

```

```
(let sps = sel bs ps; ps0 = ps -- sps; hs = compl bs ps0 sps in
  gb_schema_aux (bs @ hs) (add_pairs bs ps0 hs))"
```

The first argument of `gb_schema_aux`, bs , is the so-far computed basis, and the second argument ps is the list of all pairs of polynomials from bs whose S-polynomials might not yet reduce to 0 modulo bs . Hence, as soon as ps is empty all S-polynomials reduce to 0, then by virtue of Theorem *Buchberger-thm-finite* the list bs constitutes a Gröbner basis.

The auxiliary function `add_pairs`, when applied to arguments bs , ps_0 and hs , returns a new list of pairs of polynomials which contains precisely (i) all pairs from ps_0 , (ii) the pair (h, b) for all $h \in hs$ and $b \in bs$ ⁵, and (iii) one of the pairs (h_1, h_2) or (h_2, h_1) for all $h_1, h_2 \in hs$ with $h_1 \neq h_2$. The auxiliary function `diff_list` (infix “--”) is the analogue of set-difference for lists, i. e. it removes all occurrences of all elements of its second argument from its first argument.

The two functions `sel` and `compl` are additional parameters of the algorithm; they are not listed among the arguments of `gb_schema_aux` here merely for the sake of better readability. Informally, they are expected to behave as follows:

- If ps is non-empty, `sel(bs, ps)` should return a non-empty sublist sps of ps .
- `compl(bs, ps, sps)` should return a (possibly empty) list hs of polynomials such that (i) $0 \notin hs$, (ii) $hs \subseteq \text{pmdl}(bs)$, (iii) $\text{spoly}(p, q) \rightarrow_{bs \cup hs}^* 0$ for all $(p, q) \in sps$, and (iv) $\neg \text{lt}(b) \text{ dvd}_t \text{lt}(h)$ for all $b \in bs$ and $h \in hs$.

Typically, concrete instances of `sel` do not take bs into account, but in any case it does not harm to pass it as an additional argument. Any instances of the two parameters that satisfy the above requirements lead to a partially correct procedure for computing Gröbner bases, since `compl` takes care that all S-polynomials of the selected pairs sps reduce to 0. However, the procedure is not only partially correct, but also terminates for every input. Readers interested in the (not immediately obvious) proof of this claim are referred to any textbook on Gröbner bases, e. g. to Theorem 2.5.5 in [15].

Function `gb_schema`, finally, calls `gb_schema_aux` with the right initial values:

```
definition gb_schema :: " $((\alpha \times \kappa) \Rightarrow_0 \beta)$  list  $\Rightarrow ((\alpha \times \kappa) \Rightarrow_0 \beta)$  list"
  where "gb_schema bs = gb_schema_aux bs (add_pairs [] [] bs)"
```

Remark 1. The actual implementation of `gb_schema_aux` incorporates two standard criteria, originally due to Buchberger, for detecting so-called *useless pairs* (i. e. pairs which do not need to be considered in the algorithm) and hence improving the efficiency of the algorithm: the *product criterion* and the *chain criterion*. More information can be found in [17].

3.4 Buchberger’s Algorithm

The function implementing the usual Buchberger algorithm, called `gb`, can immediately be obtained from `gb_schema` by instantiating

⁵ Abusing notation, $x \in xs$, for a list xs , means that x appears in xs .

- `sel` to a function that selects a single pair, i. e. returns a singleton list, and
- `compl` to a function that totally reduces $\text{spoly}(p, q)$ to some normal form h using `trd`, where (p, q) is the pair selected by the instance of `sel`, and returns the singleton list $[h]$ if $h \neq 0$ and the empty list otherwise.

These instances of `sel` and `compl` can easily be proved to meet the requirements listed above, so we can finally conclude that `gb` indeed always computes a Gröbner basis of the submodule generated by its input:

theorem `gb_isGB`: `is_Groebner_basis (set (gb bs))`

theorem `gb_pmdl`: `pmdl (set (gb bs)) = pmdl (set bs)`

Gröbner bases have many interesting properties, one of them being as follows: if G is a Gröbner basis, then a polynomial p is in the submodule generated by G if and only if the unique normal form of p modulo G is 0. Together with the two previous theorems this observation leads to an effective answer to the membership problem for submodules represented by finite lists of generators:

theorem `in_pmdl_gb`: `"p ∈ pmdl (set bs) ⟷ (trd (gb bs) p) = 0"`

Example 3 (Example 2.5.7. in [15]). Let $f_1 = x^2$, $f_2 = xy + y^2$ and \preceq be the lexicographic order with $y \prec x$; we apply Buchberger's algorithm to compute a Gröbner basis of $\{f_1, f_2\} \subseteq \mathbb{Q}[x, y]$.

We start with the S-polynomial of f_1 and f_2 :

$$\text{spoly}(f_1, f_2) = y f_1 - x f_2 = -xy^2 \rightarrow_{f_2, y} y^3.$$

$f_3 = y^3$ is irreducible modulo $\{f_1, f_2\}$, so we must add it to the current basis. This ensures that $\text{spoly}(f_1, f_2)$ can be reduced to 0 modulo the enlarged basis, but it also means that we have to consider $\text{spoly}(f_1, f_3)$ and $\text{spoly}(f_2, f_3)$, too:

$$\begin{aligned} \text{spoly}(f_1, f_3) &= y^3 f_1 - x^2 f_3 = 0. \\ \text{spoly}(f_2, f_3) &= y^2 f_2 - x f_3 = y^4 \rightarrow_{f_3, y} 0. \end{aligned}$$

Since all S-polynomials can be reduced to 0 modulo $G = \{f_1, f_2, f_3\}$, G is a Gröbner basis of $\{f_1, f_2\}$.

4 Faugère's F_4 Algorithm

In Buchberger's algorithm, in each iteration precisely *one* S-polynomial is reduced modulo the current basis, giving rise to at most *one* new basis element. However, as Faugère observed in [7], it is possible to reduce several S-polynomials *simultaneously* with a considerable gain of efficiency (especially for *large* input). To that end, one selects some pairs from the list ps , reduces them modulo the current basis, and adds the resulting non-zero normal forms to the basis; in short, several iterations of the usual Buchberger algorithm are combined into one single iteration. This new algorithm is called F_4 .

The crucial idea behind F_4 , and the reason why it can be much faster than Buchberger’s algorithm, is the clever implementation of simultaneous reduction by computing the *reduced row echelon form* of certain coefficient matrices. Before we can explain how this works, we need a couple of definitions; let fs always be a list of polynomials of length m , and vs be a list of terms of length ℓ .

- `Keys_to_list(fs)` returns the list of all distinct terms appearing in fs , sorted *descending* w.r.t. \preceq_t .
- `polys_to_mat(vs, fs)` returns a matrix A (as formalized in [25]) of dimension $m \times \ell$, satisfying $A_{i,j} = \text{coeff}(fs_i, vs_j)$, for $0 \leq i < m$ and $0 \leq j < \ell$ ⁶.
- `mat_to_polys(vs, A)` is the “inverse” of `polys_to_mat`, i.e. if A is a matrix of dimension $m \times \ell$ it returns the list gs of polynomials satisfying $\text{coeff}(gs_i, vs_j) = A_{i,j}$ and $\text{coeff}(gs_i, v) = 0$ for all other terms v not contained in vs .
- `row_echelon(A)` returns the reduced row echelon form of matrix A ; it is defined in terms of `gauss_jordan` from [25].

With these auxiliary functions at our disposal we can now give the formal definitions of two concepts whose importance will become clear below:

```
definition Macaulay_mat :: " $((\alpha \times \kappa) \Rightarrow_0 \beta)$  list  $\Rightarrow \beta$  :: field mat"
  where "Macaulay_mat fs = polys_to_mat (Keys_to_list fs) fs"
```

```
definition Macaulay_red :: " $((\alpha \times \kappa) \Rightarrow_0 \beta)$  list  $\Rightarrow ((\alpha \times \kappa) \Rightarrow_0 \beta :: \text{field})$  list"
  where "Macaulay_red fs =
    (let lts = map lt (filter ( $\lambda f. f \neq 0$ ) fs) in
     filter ( $\lambda f. f \neq 0 \wedge \text{lt } f \notin \text{set lts}$ )
       (mat_to_polys (Keys_to_list fs) (row_echelon (Macaulay_mat fs))))"
```

`Macaulay_mat(fs)` is called the *Macaulay matrix* of fs . `Macaulay_red(fs)` constructs the Macaulay matrix of fs , transforms it into reduced row echelon form, and converts the resulting matrix back to a list of polynomials, from which it filters out those non-zero polynomials whose leading terms are not among the leading terms of the original list fs .

`Macaulay_red` is the key ingredient of the F_4 algorithm, because the list it returns is precisely the list hs that must be added to bs in an iteration of `gb_schema_aux`. The only question that still remains open is which argument list fs it needs to be applied to; this question is answered by an algorithm called *symbolic preprocessing*, implemented by function `sym_preproc` in our formalization. `sym_preproc` takes two arguments, namely the current basis bs and the list of selected pairs sps , and informally behaves as follows:

1. For each $(p, q) \in sps$ compute the two polynomials

$$\begin{aligned} & \text{monom_mult}(1/\text{lc}(p), \text{lcm}(\text{lp}(p), \text{lp}(q))/\text{lp}(p), p) \\ & \text{monom_mult}(1/\text{lc}(q), \text{lcm}(\text{lp}(p), \text{lp}(q))/\text{lp}(q), q) \end{aligned}$$

whose difference is precisely $\text{spoly}(p, q)$ (unless $\text{spoly}(p, q)$ is 0). Collect all these polynomials in an auxiliary list fs' .

⁶ We use 0-based indexing of lists, vectors and matrices, just as Isabelle/HOL.

2. Collect all monomial multiples of each $b \in bs$ that are needed to totally reduce the elements of fs' in a list fs'' . That means, for all b, f, g, h with $b \in bs, f \in fs', f \rightarrow_{bs}^* g$ and $\text{red_single}(g, h, b, t)$, the monomial multiple $\text{monom_mult}(1, t, b)$ of b must be included in fs'' .
3. Return the concatenation $fs'@fs''$ of fs' and fs'' .

Explaining in detail how exactly symbolic preprocessing works is not in the scope of this paper; see [7] instead. We only point out that Step 2 of the above procedure can be accomplished without actually carrying out the reductions.

Putting everything together, the function `f4_red` is obtained as

```
definition f4_red :: "(( $\alpha \times \kappa \Rightarrow_0 \beta :: \text{field}$ ) list  $\Rightarrow$ 
  ( $\alpha \times \kappa \Rightarrow_0 \beta \times (\alpha \times \kappa \Rightarrow_0 \beta)$ ) list  $\Rightarrow$  (( $\alpha \times \kappa \Rightarrow_0 \beta$ ) list)"
where "f4_red bs sps = Macaulay_red (sym_preproc bs sps)"
```

and proved to be a feasible instance of parameter *compl*; in particular, the leading terms of the polynomials in $hs = \text{f4_red}(bs, sps)$ are not divisible by the leading terms of the polynomials in bs , and indeed all S-polynomials originating from pairs in sps are reducible to 0 modulo the enlarged basis $bs@hs$:

```
lemma f4_red_not_dvd:
assumes "h  $\in$  set (f4_red bs sps)" and "b  $\in$  set bs" and "b  $\neq$  0"
shows " $\neg$  lt b dvdt lt h"
```

```
lemma f4_red_spoly_reducible:
assumes "set sps  $\subseteq$  set bs  $\times$  set bs" and "(p, q)  $\in$  set sps"
shows "(red (set (bs @ (f4_red bs sps))))** (spoly p q) 0"
```

Eventually, the resulting instance of `gb_schema` which implements Faugère's F_4 algorithm is called `f4`.

Summarizing, the simultaneous reduction of several S-polynomials boils down to the computation of the reduced row echelon form of Macaulay matrices over the coefficient field K . These matrices are typically very big, very rectangular (i.e. have much more columns than rows) and extremely sparse. Therefore, if F_4 is to outperform Buchberger's algorithm, such matrices must be stored efficiently, and the computation of reduced row echelon forms must be highly optimized; we again refer to [7] for details. Formalizing such efficient representations of matrices in Isabelle/HOL is future work.

5 Further Features

5.1 Reduced Gröbner Bases

Gröbner bases are not unique, even if the ordering \preceq_t is fixed. For instance, if G is a Gröbner basis and $B \subseteq \text{pmdl}(G)$, then $G \cup B$ is still a Gröbner basis of the same submodule. One can, however, impose stronger constraints on generating sets of submodules than giving rise to a confluent reduction relation, which *do* ensure uniqueness; the resulting concept is that of *reduced Gröbner bases*.

The central idea behind reduced Gröbner bases is *auto-reducedness*: a set B of polynomials is auto-reduced if and only if no $b \in B$ can be reduced modulo $B \setminus \{b\}$. A reduced Gröbner basis, then, is simply an auto-reduced Gröbner basis of non-zero monic⁷ polynomials:

```

definition is_reduced_GB :: " $((\alpha \times \kappa) \Rightarrow_0 \beta)$  set  $\Rightarrow$  bool"
  where "is_reduced_GB B  $\longleftrightarrow$  is_Groebner_basis B  $\wedge$  is_auto_reduced B  $\wedge$ 
    is_monic_set B  $\wedge$  0  $\notin$  B"

```

After having defined reduced Gröbner bases as above, one can prove that every finitely-generated submodule of $K[x_1, \dots, x_n]^k$ has a unique reduced Gröbner basis (unique only modulo the implicitly fixed term order \preceq_t , of course):

```

theorem exists_unique_reduced_GB_finite:
  assumes "finite F"
  shows " $\exists!$ G. is_reduced_GB G  $\wedge$  pmdl G = pmdl F"

```

The proof of *exists-unique-reduced-GB-finite* we give in the formalization is even constructive, in the sense that we formulate an algorithm which auto-reduces and makes monic a given set (or, more precisely, list) of polynomials, and, therefore, when applied to *some* Gröbner basis returns *the* reduced Gröbner basis of the submodule it generates.

5.2 Gröbner Bases of Syzygy Modules

Given a list $bs = [b_0, \dots, b_{m-1}]$ of m polynomials, one could ask oneself what the polynomial relations among the elements of bs are. In other words, one wants to find all m -dimensional vectors $s = (s_0, \dots, s_{m-1})^T \in K[x_1, \dots, x_n]^m$ such that $\sum_{i=0}^{m-1} s_i b_i = 0$. In the literature, such a vector s is called a *syzygy* of bs [15], and as one can easily see the set of all syzygies of bs forms a submodule of $K[x_1, \dots, x_n]^m$.

Example 4. If $m = 2$ and $b_0, b_1 \in K[x_1, \dots, x_n] \setminus \{0\}$, then a non-trivial syzygy is obviously given by $\begin{pmatrix} b_1 \\ -b_0 \end{pmatrix}$, because $b_1 b_0 + (-b_0) b_1 = 0$. More generally, each list of scalar polynomials with at least two non-zero elements admits non-trivial syzygies of the above kind.

As it turns out, it is not difficult to compute Gröbner bases of syzygy modules. We briefly outline how it works in theory; so, assume that bs is the m -element list $[b_0, \dots, b_{m-1}]$ of scalar polynomials in $K[x_1, \dots, x_n]$.

1. Turn each of the b_i into a $(m + 1)$ -dimensional vector of polynomials, such that b_i becomes $(0, \dots, 0, 1, 0, \dots, 0, b_i)^T$, where the 1 occurs precisely in the i -th component ($0 \leq i < m$). Call the resulting list bs' .
2. Compute a Gröbner basis gs of bs' w. r. t. a POT-extension \preceq_t of some admissible order \preceq on power-products.

⁷ A polynomial is called *monic* if and only if its leading coefficient is 1.

- From gs extract those elements of the form $(s_0, \dots, s_{m-1}, 0)^T$, and restrict them to their first m components $(s_0, \dots, s_{m-1})^T$. These vectors constitute a Gröbner basis w. r. t. \preceq_t of the syzygy module of bs .

In the formalization we prove that the above algorithm indeed computes a Gröbner basis of the syzygy module in question; we leave out the details here but refer to [17] instead. Only note that syzygy modules can also be computed if the initial list bs contains vector-polynomial already, making it possible to compute syzygy modules of syzygy modules, and therefore also *exact sequences* and *free resolutions*. Obviously, this cannot be done with Gröbner bases in the ordinary setting of rings.

6 Conclusion

We hope we could convince the reader that the work described in this paper is an elegant and generic formalization of an interesting and important mathematical theory in commutative algebra. Moreover, the formalized algorithms can be turned into executable Haskell/OCaml/Scala/SML code by means of Isabelle’s *Code Generator* [8], to do certified computations of Gröbner bases; see [17] for more information and a discussion of performance issues⁸. Even though other formalizations of Gröbner bases in other proof assistants exist, ours is the first featuring the F_4 algorithm and Gröbner bases of modules. Besides, our work also gives an affirmative answer to the question whether multivariate polynomials as formalized in [10] can effectively be used for formalizing algorithms in computer algebra.

Our own contributions to multivariate polynomials make up approximately 8600 lines of proof, and the Gröbner-bases related theories make up another 16700 lines of proof (3000 lines of which are about Macaulay matrices and the F_4 algorithm), summing up to a total of 25300 lines. Most proofs are intentionally given in a quite verbose style for better readability. The formalization effort was roughly eight person-months of full-time work, distributed over two years of part-time work. This effort is comparable to what the authors of other formalizations of Gröbner bases theory in other proof assistants report; see Sect. 1.1 for a list thereof.

Acknowledgments. We thank the anonymous referees for their valuable comments.

⁸ The performance issues mainly concern the non-optimal representation of polynomials by *unordered* associative lists. We are currently working on a much more efficient representation by *ordered* associative lists.

References

1. Ballarin, C.: Tutorial to locales and locale interpretation. In: Lambán, L., Romero, A., Rubio, J. (eds.) *Contribuciones Científicas en Honor de Mirian Andrés Gómez*, pp. 123–140. Servicio de Publicaciones de la Universidad de La Rioja (2010). Part of the Isabelle documentation: <https://isabelle.in.tum.de/dist/Isabelle2017/doc/locales.pdf>
2. Bentkamp, A.: Expressiveness of deep learning. *Archive of Formal Proofs* (2016). Formal proof development: http://isa-afp.org/entries/Deep_Learning.html
3. Buchberger, B.: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal. Ph.D. thesis, Mathematisches Institut, Universität Innsbruck, Austria (1965). English Translation in *J. Symb. Comput.* **41**(3–4), 475–511 (2006)
4. Buchberger, B.: Towards the automated synthesis of a Gröbner bases algorithm. *RACSAM, Serie A: Math.* **98**(1), 65–75 (2004)
5. Chaieb, A., Wenzel, M.: Context aware calculation and deduction. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) *Calculemus/MKM 2007*. LNCS (LNAI), vol. 4573, pp. 27–39. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73086-6_3
6. Craciun, A.: Lazy thinking algorithm synthesis in Gröbner bases theory. Ph.D. thesis, RISC, Johannes Kepler University Linz, Austria (2008)
7. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases (F_4). *J. Pure Appl. Algebra* **139**(1), 61–88 (1999). [https://doi.org/10.1016/S0022-4049\(99\)00005-5](https://doi.org/10.1016/S0022-4049(99)00005-5)
8. Haftmann, F.: Code Generation from Isabelle/HOL Theories. Part of the Isabelle documentation: <http://isabelle.in.tum.de/dist/Isabelle2017/doc/codegen.pdf>
9. Haftmann, F.: Haskell-Like Type Classes with Isabelle/Isar. Part of the Isabelle documentation: <http://isabelle.in.tum.de/dist/Isabelle2017/doc/classes.pdf>
10. Haftmann, F., Lochbihler, A., Schreiner, W.: Towards Abstract and Executable Multivariate Polynomials in Isabelle (2014). Contributed talk at Isabelle Workshop 2014, associated with ITP 2014, Vienna, Austria: http://www.risc.jku.at/publications/download/risc_5012/IWS14.pdf
11. Harrison, J.: Complex quantifier elimination in HOL. In: Boulton, R.J., Jackson, P.B. (eds.) *TPHOLs 2001: Supplemental Proceedings*, pp. 159–174. Division of Informatics, University of Edinburgh (2001). <http://www.informatics.ed.ac.uk/publications/report/0046.html>
12. Immler, F., Maletzky, A.: Gröbner bases theory. *Archive of Formal Proofs* (2016). Formal proof development: http://isa-afp.org/entries/Groebner_Bases.html
13. Immler, F., Maletzky, A.: Gröbner Bases of Modules and Faugère’s F_4 Algorithm (formalization) (2018). Compatible with Isabelle 2017: <http://www.risc.jku.at/people/amaletzky/CICM2018.html>
14. Jorge, J.S., Guilas, V.M., Freire, J.L.: Certifying properties of an efficient functional program for computing Gröbner bases. *J. Symb. Comput.* **44**(5), 571–582 (2009). <https://doi.org/10.1016/j.jsc.2007.07.016>
15. Kreuzer, M., Robbiano, L.: *Computational Commutative Algebra 1*. Springer, Heidelberg (2000). <https://doi.org/10.1007/978-3-540-70628-1>
16. Maletzky, A.: Computer-assisted exploration of Gröbner bases theory in Theorema. Ph.D. thesis, RISC, Johannes Kepler University Linz, Austria (2016)
17. Maletzky, A., Immler, F.: Gröbner bases of modules and Faugère’s F_4 algorithm in Isabelle/HOL (extended version). Technical report (2018). [arXiv:1805.00304](https://arxiv.org/abs/1805.00304) [cs.LO]

18. Medina-Bulo, I., Palomo-Lozano, F., Ruiz-Reina, J.L.: A verified common lisp implementation of Buchberger's algorithm in ACL2. *J. Symb. Comput.* **45**(1), 96–123 (2010). <https://doi.org/10.1016/j.jsc.2009.07.002>
19. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL—A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002). <https://doi.org/10.1007/3-540-45949-9>
20. Persson, H.: An integrated development of Buchberger's algorithm in Coq. Technical report 4271. INRIA Sophia Antipolis (2001)
21. Schwarzweiler, C.: Gröbner bases – theory refinement in the Mizar system. In: Kohlhase, M. (ed.) MKM 2005. LNCS (LNAI), vol. 3863, pp. 299–314. Springer, Heidelberg (2006). https://doi.org/10.1007/11618027_20
22. Sternagel, C., Thiemann, R.: Abstract rewriting. *Archive of Formal Proofs* (2010). Formal proof development: <http://isa-afp.org/entries/Abstract-Rewriting.html>
23. Sternagel, C., Thiemann, R., Maletzky, A., Immler, F.: Executable multivariate polynomials. *Archive of Formal Proofs* (2010). Formal proof development: <http://isa-afp.org/entries/Polynomials.html>
24. Théry, L.: A machine-checked implementation of Buchberger's algorithm. *J. Autom. Reason.* **26**(2), 107–137 (2001). <https://doi.org/10.1023/A:1026518331905>
25. Thiemann, R., Yamada, A.: Matrices, Jordan normal forms, and spectral radius theory. *Archive of Formal Proofs* (2015). Formal proof development: http://isa-afp.org/entries/Jordan_Normal_Form.html



MathChat: Computational Mathematics via a Social Machine

Manfred Minimair^(✉) 

Seton Hall University, South Orange, NJ 07079, USA
Manfred.Minimair@shu.edu

Abstract. The main question of this research is: How does a social machine discover algorithmic mathematical knowledge? A social machine is a system of humans and computers engaged in some purposeful activity. To address the main question, an empiric and theoretical framework for algorithmic mathematical knowledge discovered by the social machine is proposed. The framework is derived from findings in Distributed Cognition documenting how collaborators evolve a mathematical algorithm. By combining Distributed Cognition with the standard Message Passing Model of Distributed Computing, a formalism is introduced to specify the activities of the social machine and its algorithmic knowledge. Furthermore, the software system MathChat is introduced which provides an online environment for social machines engaged in mathematical computations. An application of MathChat in network analysis education is described which outlines a social machine covered by the proposed framework.

Keywords: Collaborative computational mathematics
Collaborative chat · Mathematical social machines

1 Introduction

The main question of this research is: *How does a social machine discover algorithmic mathematical knowledge?* The main contribution of this work is to provide an answer by an empiric and theoretical framework for algorithmic mathematical knowledge discovered by a social machine. A social machine is broadly defined as a “combination of people and computers as a single problem-solving entity” and a mathematical social machine is one that engages in mathematics [12]. Accordingly, the current research studies social machines as a whole while making mathematical algorithmic discoveries, and not how its human components learn individually through collaborating. (As background information, the relations among collaboration, social machines and individual learning are reviewed in the following Sect. 2.) Note that for the purpose of this work the term “computer” in the definition of social machine is interpreted broadly, including

Supported by Comcast Innovation Fund.

any type of computing device and also computational software not necessarily bound to a specific hardware.

To address the main question of this work, the research answers three subordinated questions, which in combination provides an empiric and theoretical framework addressing the main question:

1. How does a social machine discover a mathematical algorithm?
2. How may algorithmic mathematical knowledge stemming from this process of knowledge discovery be formally specified?
3. How may the discovery process be reproduced?

To answer Question 1, Sect. 3 examines prior findings from the research literature [7] suggesting a hypothesis outlining a process for collaborative discovery of mathematical algorithms. Answering Question 2, Sect. 4 proposes a formalization of the notion of algorithmic knowledge of a mathematical social machine. This formalization clarifies what can be understood as algorithmic knowledge and therefore contributes towards answering the main research question. Together, Sects. 3 and 4 formally explain how a social machine, as derived from the research literature [7, 12], discovers algorithmic mathematical knowledge. Then Sect. 5 presents the software system MathChat [14], which the author has developed to create mathematical social machines in an online environment, and a pilot investigation, eliciting algorithmic discoveries in the context of online education. Some additional details of the design and implementation of the MathChat software system for collaborative computing are described in the appendix. Therefore, in combination, the Sects. 3, 4 and 5 propose a fundamental framework for understanding, both empirically and theoretically, the discovery of algorithmic knowledge by mathematical social machines. Section 6 discusses limitations and future directions of this work. Finally, the appendix elaborates on additional design and implementation details of the MathChat system.

2 Technical Background Information

Collaboration in mathematics [2] allows the individual workers in a team, incorporating state-of-the-art mathematical software systems [13], to transcend their own limitations [7, 10] and therefore make progress in mathematics [2, 12]. For example, the Polymath project, organized through an online blog, allows anybody to join into efforts solving mathematical problems because the blog is available to anybody with access to the Web [3]. Therefore, mathematicians with diverse expertise are enabled to collaborate in the Polymath project. The diversity of knowledge and skills of the participants benefits the research endeavor, allowing the team of mathematicians to arrive at solutions to problems more efficiently than any individual alone.

When viewed as a system engaged in mathematics, the collaborators together with their tools are called a mathematical social machine [12]. In psychology, this system view is powerful because it allows to explain cognitive processes, such as memorization, learning, and computation, that occur when people interact

with their surroundings. This is the goal of the Theory of Distributed Cognition [6, 7]. This theory is based on the important concept of Distribution of Cognitive Processes, namely:

- Cognitive processes may be distributed across the members of a social group.
- Cognitive processes may involve coordination between internal and external (material and environmental) structure.
- Processes may be distributed through time in such a way that the products of earlier events can transform the nature of later events.

In mathematics, the cognitive power of teams has also been studied, namely, for gaining mathematical knowledge. At the secondary education-level, the Virtual Math Teams project investigates online environments and teams of learners solving mathematical problems [20]. The online environments include chat rooms, online sketch pads and a computer algebra system. The project led to the development of a theory of group cognition [19]. This theory aims at understanding how a small team of online collaborators construct knowledge as a group in an online environment, but not only as the sum of the team members learning individually.

While the system view is necessary to explain cognitive abilities transcending individuals, the aim of Computer Supported Collaborative Learning research [20] is to understand how students learn individually from participating in online teams and how collaboration software may be designed to support such learning. In contrast, the current work focuses on social machines as systems that discover mathematical algorithmic knowledge and not how the individual participants learn through collaboration.

3 Collaboratively Discovering a Mathematical Algorithm

3.1 Example of Collaborative Algorithm Discovery

When developing the Theory of Distributed Cognition, Hutchins [7] investigated an application of mathematical computing, namely, the navigation of naval vessels, when GPS was not yet in use. In an emergency situation as the ship's gyrocompass fails, a team of sailors approximates the position of their ship incorporating various measurements available to them, such as direction of a landmark relative to the ship's head, direction of the ship with respect to the magnetic north and difference between the true north and magnetic north. Hutchins shows that during the course of a series of such calculations, the team corrects its initial faulty approach and eventually evolves an efficient distributed algorithm for computing the ship's position. That is, after 38 of 66 computations, the team of sailors had evolved a stable scheme for efficiently computing the position, without being individually aware of the distributed algorithm they were executing.

The team of sailors can be regarded as a social machine consisting of the sailors and their computing tools, including calculators. Therefore Hutchin's example suggests that a mathematical social machine may be able to discover an algorithm by computing examples. The remainder of this section is dedicated to giving a more formal and exact statement of this hypothesis.

3.2 Distributed Cognition Foundation

Distributed Cognition studies how information flows among the components of systems of humans and their tools through communication channels [6]. Such systems can be represented as graphs where the nodes are humans or their tools and the edges are the communication channels among the connected edges, where any types of tools or communication channels are permissible. For the case of computational mathematics, tools may include calculators, whiteboards and mathematical software systems. Moreover, the communication channels may be varied, such as audio-visual, from human to human, textual, from human to computer, through commands entered on a keyboard to initiate mathematical computations, or digital and network-based, to connect remote computers.

There is an issue with the theory of Distributed Cognition. Being a general theory on cognition, it does not provide any formal notation for specifying the activities carried out or algorithms implemented inside the mathematical social machine when performing mathematical computations. Therefore, the current paper proposes to enhance Distributed Cognition with the formalism provided by the Message Passing Model of Distributed Computing [1].

Distributed Cognition has been blended with other theories or descriptive systems to support specific studies, for example with Activity Theory [16] and Actor Network Theory [15]. However, it seems it has not yet been combined with the message passing formalism of distributed computing, as proposed here.

3.3 Collaborative Computation Model (CCM)

To allow the formal specification of the activities of a system engaged in mathematical computing, the formalism from the Asynchronous Message Passing Model (AMPM) [1] of Distributed Computation is adopted to Distributed Cognition as follows.

Standard Asynchronous Message Passing Model. The AMPM considers systems of nodes that communicate asynchronously. In such an asynchronous message-passing system (AMPS), processors exchange information by sending messages over bi-directional channels. The channels among the processors are assumed to be fixed, and the pattern of connections is called the system's communication topology, usually represented as a graph. For example, Fig. 1 shows a system with four nodes, the processor p_0, p_1, p_2 and p_3 , where the connecting lines among the nodes indicate the communication channels. The processors in the AMPS are assumed to be finite state machines, each with a transition function acting on the machine's internal state and the messages it has received. Applying the machine's transition function results in potentially altering the internal state of the machine and/or scheduling a produced output to be send as a message through a specific communication channel. Then an execution segment of an AMPS is defined as the sequence of the following form: $C_0, \phi_1, C_1, \phi_2, C_2, \phi_3, \dots$, where the configuration C_i is the list of the states of the processors p_i and the symbol ϕ_k represents an event in the system. There are two types of event,

namely, either sending of a message from processor p_i to processor p_j or a computation by processor p_i through applying its transition function. Furthermore, an execution of an AMPS is defined as an execution segment starting with a specific initial configuration C_0 . Additionally, an algorithm executed by the AMPS is specified by describing the algorithms computing the transition functions of the processors. When an AMPS executes an algorithm, some nodes are accordingly designated as input and output nodes. The initial states of the input nodes and the final states of the output nodes represent the input and respectively output of the algorithm.

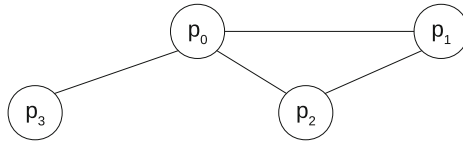


Fig. 1. Example for the communication topology of an AMPS [1]

Collaborative Computation Model Extension

Definition 1. The Collaborative Computation Model (CCM) extends the AMPM by allowing humans to act as processors in addition to finite state machines. Let such an extended message passing system be called Collaborative Computation System (CCS).

For an example see Fig. 2 where the processors p_1 and p_2 of Fig. 1 have been replaced with the humans h_1 and h_2 .

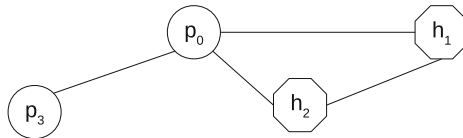


Fig. 2. CCS including humans

By including humans, the CCS is a type of system that is subject of the theory of Distributed Cognition. It is also a special instance in this theory because the non-human nodes of the system are required to be finite state machines, whereas Distributed Cognition allows any kind of nodes. Furthermore, communication and messages sent among human nodes and the processors may be of various formats, such as audio-visual or textual, depending on the channel and receiving node, as already pointed out in Sect. 3.2.

The addition of humans in the CCS requires extending the definitions of configuration C_i and event ϕ_k in execution segments, and redefining what it means

that a system executes an algorithm. Since humans do not have an observable state as processors, the configuration C_i shall not include any information regarding the state of the human processors. Furthermore, since humans do not have an observable transition function, the expression “applying the transition function” of a human node shall stand for the human carrying out an internal cognitive process which potentially results in some statement by the human which is scheduled to be transmitted through the communication channel.

As mentioned before, it cannot be assumed that humans have an observable transition function, and consequently there is no algorithm computing the transition function. Therefore, the subsequent Definition 2 refers to an AMPS for defining what it means that a CCS executes an algorithm.

For the following definition and the remaining sections, let \mathcal{L} be the class of algorithms [4] implementing the computable functions $A \rightarrow Z$, for some appropriately representable sets A and Z .

Definition 2. The CCS S executes the algorithm $l \in \mathcal{L}$ iff there is the AMPS P with the same communication topology as S executing l and, for all inputs $a \in A$ (with representations suitable for processing with S and P), the CCS S produces an execution that is also generated by P for input a , except that events that are cognitive processes are replaced by the applications of the transition functions of the finite state machines substituted for the corresponding human nodes in S .

Remark 1. Potentially, one can obtain P from S by consecutively replacing all human nodes in S by finite state machines that carry out the same tasks as the humans. To enable such a process, the above definition does not require that the finite state machines occurring in S also exist in P . Replacing a human node in S with a finite state machine may require changing the way messages are communicated with the neighboring nodes originally connected to the human node. Therefore, the neighboring nodes themselves will potentially be modified to be able to use the new type of communication channel. For example, a human may communicate with a finite state machine by typing on a keyboard, whereas two finite state machines communicate by sending messages through a computer network.

3.4 Formalized Hypothesis on Algorithm Discovery

Section 3.1 suggests that a social mathematical machine represented by a CCS may be able to discover an algorithm by computing examples. In other words, the CCS may *know* an algorithm after computing a series of examples. Thus, to formalize this empirical hypothesis, it needs to be clarified what one should understand by the statement that a CCS knows an algorithm. While a CCS in general may not possess knowledge in the philosophical sense, as justified true belief of a person [8], one may observe what it is capable of doing. Therefore, we say a CCS is able to compute a function on a finite set of inputs if this behavior can be observed.

Definition 3. The CCS S is able to compute the function $f : A \rightarrow Z$ on the finite subset $B \subseteq A$ iff S executes an algorithm for computing f and computes $f(x)$ for any input $x \in B$.

Now we are ready to state the formalized hypothesis on algorithm discovery.

Hypothesis 1. Let \mathcal{S} be the set of all CCS that are able to compute f on the sequence $B_0 \subseteq B_1 \subseteq \dots \subseteq B_j$, for some function $f : A \rightarrow Z$ and sequence of finite subsets $B_0 \subseteq B_1 \subseteq \dots \subseteq B_j$ of A . Then for some f and some sequence of B_i 's there is a CCS in \mathcal{S} that is able to compute f on any finite subset B of A .

There is at least one CCS, namely, the one observed by Hutchins, that agrees with the hypothesis. The pilot investigation of Sect. 5 elicits algorithmic discoveries from other systems.

Remark 2. Note that in Hypothesis 1, it is assumed that the CCS compute f on a sequence of finite subsets B_i rather than on a sequence of elements $b_i \in A$. This equivalent and slightly redundant assumption has been made to maintain a consistent presentation in the next section.

4 Formally Describing Learned Algorithmic Knowledge

This section describes how the formalism of the CCM from Sect. 3 may be applied to formally describe the algorithmic knowledge learned by a system repeatedly carrying out the same type of mathematical computation as in Hypothesis 1. Since the usual CCS, like the team of sailors observed by Hutchins, neither is able to tell us what it knows nor can we fully observe the internal states of its components, the following definitions propose to represent the knowledge that can be inferred from *observing* the CCS (as an inductive inference, but not a mathematical induction), and therefore it may be assumed, if so desired, that the CCS possesses this knowledge. It will be shown that such inducible algorithmic knowledge may formally be viewed as the colimit (inductive limit) [11] of a certain functor derived from the CCM and its observed computations.

Consider the arbitrary but fixed function $f : A \rightarrow Z$. Observing that S computes the function $f : A \rightarrow Z$ on the subset $B \subseteq A$ implies that S executes one of possibly many algorithms implementing some function that agrees with f when restricted to B . Accordingly, the following definition proposes to formally capture this conclusion about the knowledge of the CCS. The choice of name to be defined (Inducible Knowledge How) is motivated by the philosophical concept of Knowing How, which addresses knowledge for doing something [5].

Definition 4. For the following definitions, introducing some auxiliary notions for Definition 5, suppose that the CCS S is able to compute the function $f : A \rightarrow Z$ on the sequence of subsets $B_0 \subseteq B_1 \subseteq B_2 \subseteq \dots$ of A .

- Let $F(l)$ be the function implemented by the algorithm $l \in \mathcal{L}$ and $M_f(l)$ be the maximal subset of A on which $F(l) = f$.

- Define the equivalence relation $l \sim k$ such that l and k are equivalent iff $M_f(l) = M_f(k)$.
- Furthermore, \tilde{l} denote the corresponding equivalence classes in $\tilde{\mathcal{L}} = \mathcal{L}/\sim$.
- Note that by these definitions the function M_f may naturally be extended to the equivalence classes in $\tilde{\mathcal{L}}$ through $M_f(\tilde{l}) = M_f(l)$.
- Furthermore, let $\mathbb{L}_f = (\tilde{\mathcal{L}}, \rightarrow)$ be the category that has arrows $\tilde{k} \rightarrow \tilde{l}$ iff $M_f(\tilde{k}) \subseteq M_f(\tilde{l})$.
- Let ω stand for the linear order category $0 \rightarrow 1 \rightarrow 2 \rightarrow \dots$ [11].

Now, we are ready to define Inducible Knowledge How.

Definition 5 (Inducible Knowledge How). Let the CCS S be able to compute f on the sequence of finite sets $B_0 \subseteq B_1 \subseteq B_2 \subseteq \dots$. Then, the Inducible Knowledge How of S over the sequence B_i is defined as the colimit of the functor $K : \omega \rightarrow \tilde{\mathcal{L}}$, mapping any arrow $j \rightarrow (j+1)$ to $\tilde{l}_j \rightarrow \tilde{l}_{j+1}$, such that $M_f(\tilde{l}_i) = B_i$ for all i .

For any finite set B there is an algorithm l with $M_f(l) = B$ because B is finite, which ensures that Inducible Knowledge How exists as defined. Furthermore, the functor K and therefore the Inducible Knowledge How are uniquely defined because $M_f(\tilde{l}) = M_f(\tilde{k})$ implies that $\tilde{l} = \tilde{k}$.

Remark 3. Practically, one may observe the CCS S completing its computations only for a finite number of times. That is, B_i is a finite sequence with $B = B_j = B_{j+1} = \dots$ for some sufficiently large j . Then the Inducible Knowledge How of S over the sequence B_i is represented by the equivalence class \tilde{l} of algorithms such that $M_f(\tilde{l}) = B$, which covers the CCS conforming with Hypothesis 1.

5 Implementing Mathematical Social Machines

5.1 MathChat Features for Realizing the Social Machines

For realizing the mathematical social machines, the author has designed and implemented the MathChat software system. The MathChat system facilitates collaborative command entry for mathematical computations in the scripting language Python. The design of the MathChat software system integrates chat with command entry in one user interface, thus enabling communication and coordination among the collaborating users.

The MathChat system represents a CCS with an arbitrary number of human participants h_i and one processor p_0 carrying out mathematical computations, as illustrated by Fig. 3. The humans communicate among each other with textual chat and with the processor with textual commands in the scripting language. Therefore, there are chat communication channels among all the humans and additional channels between the humans and the processor to transfer textual

commands to and outputs of the commands from the processor. To avoid overplotting, Fig. 3 does not show the channels among the humans, but indicates that all humans are connected with a dashed line surrounding the corresponding nodes.

The humans interact with the kernel through user client software, called ChatConsole, running on the users' work stations. The ChatConsoles accept commands for the processor and chat messages for the other users. Figure 4 shows an excerpt from a session where two users, Cindy and Fred, collaboratively analyze some network. Both users run their individual ChatConsoles and see the same output simultaneously on their own work stations. More details of the design and implementation of the MathChat system are presented in the appendix.

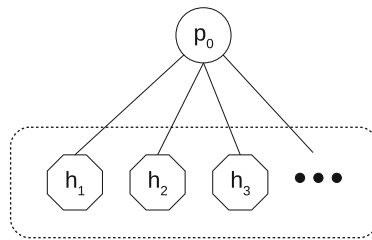


Fig. 3. Communication topology of the MathChat CCS

5.2 Application in Online Education and Pilot Investigation

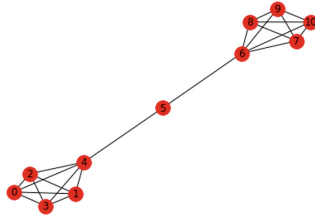
While [7] gives an example of a basic mathematical social machine active on a naval vessel, the pilot investigation of this section aims at creating social machines in the context of online technology. Being a *pilot* investigation, it aims at covering a limited aspect of algorithmic discovery, namely, distribution of tasks among human nodes in the social machine. A more comprehensive approach goes beyond the scope of the current work.

Since the author is not only a researcher but also an instructor for an online course in data analytics, his students have been readily available as participants in mathematical social machines. That is, to increase the learners' outcomes at the instructor's institution, the online students are encouraged to forge connections among each others by collaborating on problems. Therefore, motivated by the online course material, the students naturally form mathematical social machines. It is important to point out that including active students limits the data that can be shared from the application. That is, privacy laws and ethical considerations prevent the author to present data on individual interactions of students, unlike related human subject research which collects data from controlled experiments [20]. Therefore, this section presents the application of MathChat to enhance the educational experience of the online students, which

```

fred> In [288]# What do we need to do?
cindy> In [289]# Create a barbell graph and find cliques
cindy> In [290]: g = nx.barbell_graph(m1=5, m2=1)
cindy> In [291]# How do I plot it?
fred> In [292]: nx.draw(g, with_labels=True); plt.show()

```



```

cindy> In [293]# Here are the cliques
cindy> In [294]: list(nx.find_cliques(g))
Out[294]: [[4, 0, 1, 2, 3], [4, 5], [6, 5], [6, 7, 8, 9, 10]]

```

Fig. 4. Users collaborating

also yields insights about mathematical social machines working on algorithmic discoveries, as formalized in Sects. 3 and 4.

During the spring semester of 2018, the MathChat software system has been used in an online course on data analytics taught by the author at Seton Hall University. The online course includes 28 students (undergraduate and graduate) from a variety of backgrounds, including mathematics, computer science, psychology and public administration. Most of the students have not had any experience in data analytics nor Python coding.

After learning the basics of data and network analysis with Python, the students work on group assignments in ChatConsole. The students are assigned to teams of up to six students and solve a series of similar problems in network analysis. The problems asked the students to determine characteristic properties of a series of networks G1, G2, G3 and G4 (represented as multigraphs with directed edges) to reveal how the networks are different. Earlier in the course they had learned that important properties are determined through computing the number of nodes and edges, identifying the central nodes in a network (according to some appropriate measure) or finding the largest connected components for a series of networks. The assignment was purposefully presented as open-ended to allow the students to develop a way of distributing different tasks. To encourage distributing the tasks among the team members and to avoid that the whole assignment be solved by any single ambitious individual among the team, the students were required to come up with a solution where each member enters at most three Python commands.

Among the six groups, four succeeded and two failed completing the assignment. Note that Hypothesis 1 also allows for failures. After the due date, an anonymous survey was given to collect feedback about the students' learning experience. In the survey, 75% of the students stated that they found it very easy, easy, or neither easy nor difficult to collaborate, 25% found it difficult or very difficult and the rest did not respond with a rating. Furthermore, it was reported that some students did several practice runs for the collaborative effort to come up with an effective task distribution, which corresponds to the observations from Sects. 3 and 4, thus matching the proposed theoretical framework.

6 Discussion, Limitations and Future Directions

This research studied how social machines discover algorithmic mathematical knowledge. Motivated by some findings from the research literature, an empiric and theoretical framework for algorithmic mathematical knowledge discovered by a social machine has been proposed. The framework is founded on the insight that social machines learn algorithms by repeatedly computing examples for the algorithm. It is important to recall that this insight is *empiric* and therefore it is conceivable that mathematical social machines still possess other capabilities for discovering algorithms that are not covered by the proposed framework. If such capabilities were revealed, the framework would have to be expanded which is part of the natural life-cycle of scientific theories.

Definition 5 represents the algorithmic knowledge of a mathematical social machine as an abstract colimit, which is of inherent theoretical interest. Additionally, the author plans to use this formalism for ongoing and future work. The author is developing an inductive programming assistant [9] for the human participants of the social machine. Eventually, the formalism of Definition 5 will aid in proving the correctness of the inductive programming capability.

Section 5.2 describes instances of mathematical social machines that learn task distribution among their team members in the context of online education. To allow a more comprehensive study of mathematical social machines discovering distributed algorithms, the author is designing human subject experiments to collect and analyze interaction data.

Acknowledgments. The author thanks the anonymous reviewers for valuable suggestions for improving the paper and Comcast Innovation Fund for financially supporting the project.

Appendix

The author has developed the software system MathChat to realize mathematical social machines in an online environment. For an introduction to its functionality see Sect. 5. This appendix elaborates on key design and implementation issues.

The software system MathChat supports collaborative computational mathematics by tightly integrating mathematical command entry with chat for communications among the collaborators. Its main software component ChatConsole

is available at [14] as an open-source project. The software, written in the programming language Python, is based on the Jupyter project [17] which defines a protocol allowing user client software to communicate with computational kernels (processor of Fig. 3), such as iPython [17] or the Sage system [18] which runs under Python. In Fig. 4, the users enter commands in the scripting language Python. The author's software project [14] currently focuses on computational kernels accepting commands in Python, but the implementation techniques described in this section are general enough to be used for kernels accepting other languages.

To distinguish chat from commands, the chat messages are prefixed with a pound sign, which is also the sign for a comment in Python. Furthermore, the input lines show the names of the users who entered the commands or chat messages. The users enter text in an input area, shown at the bottom of the screen shot in Fig. 5. The users may switch the input area between chat- and code-entry mode by clicking on the corresponding button in the lower left-hand corner. Additionally, when in code mode, the ChatConsole automatically interprets any entered line starting with a pound sign as a chat message. The chat mode is more convenient when entering multi-line chat messages. By integrating chat with the commands in the output shown in ChatConsole, the chat messages serve the team's communication and coordination during an ongoing collaboration session and document the common work after it is completed. In addition to enabling chat, the ChatConsole user clients keep track of users connecting to and disconnecting to an ongoing session with the computational kernel.

The implementation of ChatConsole is driven by the Jupyter protocol [17], which the software components use to communication with each other. To distribute chat messages and notifications about users connecting and disconnecting in the system, ChatConsole uses an additional protocol embedded in messages sent through the Jupyter protocol. This addition does not require modifying the Jupyter protocol or kernel. In the Jupyter protocol, the kernel echos all received lines of code to all the connected clients. This echo also includes commands that have no computational effect, such as comments, prefixed with a pound sign in Python. Using the echo, ChatConsole sends special messages, starting with a pound sign and a secret string followed by some special textual data (Fig. 6), to all the clients. Correspondingly, the clients filter and process messages received according to the secret string. (If ChatConsole is connected to a kernel expecting commands in a language other than Python, then the pound sign would be replaced with the comment symbol of that language.) Messages that are identified with the secret strings are treated as chat communications relayed to all the connected users or internal instructions for user management, such as "user cindy has joined", which allow the ChatConsoles to update their internal lists of connected users. The secret string must be known to all connected clients. Since each Jupyter session has a unique secret identifier determined by the kernel, the secret string may be automatically generated from the session identifier, which makes it available to the clients upon connecting to the kernel. The encoding of the data, transporting the chat messages or user management instructions, is

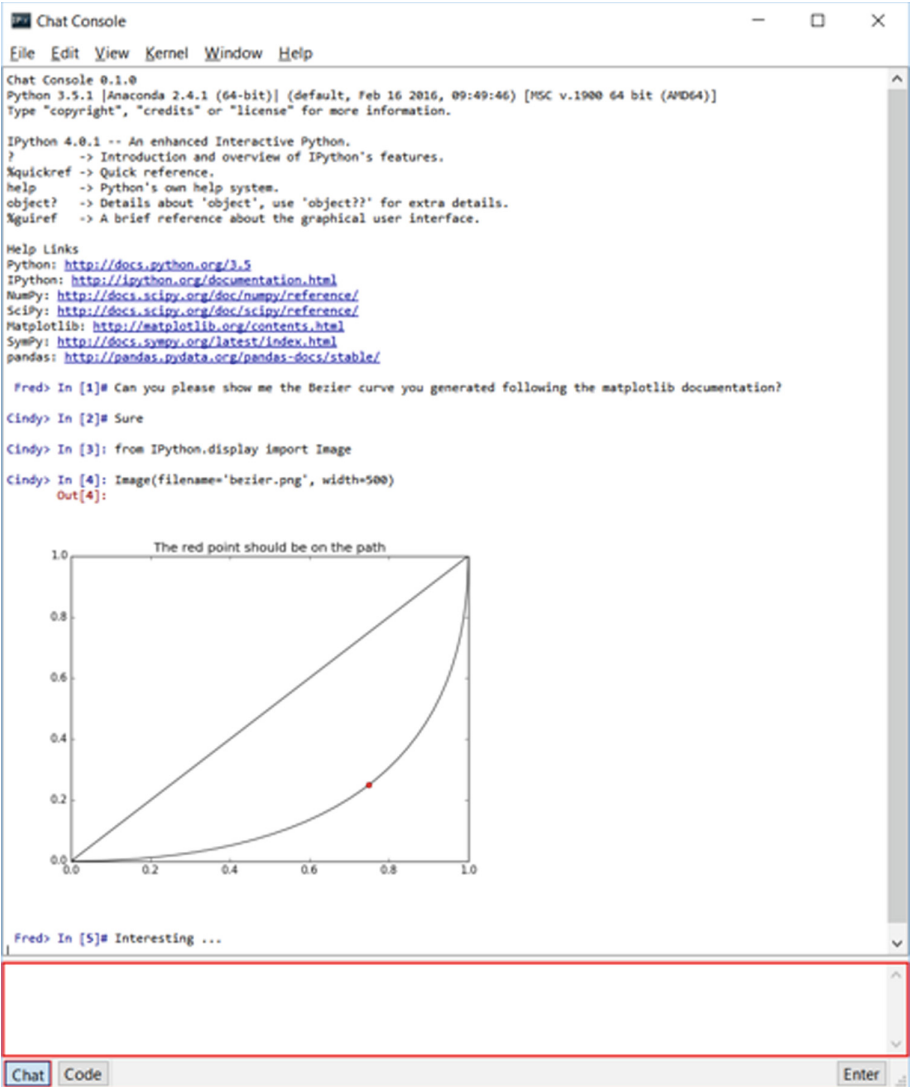


Fig. 5. Input area

#<secret string><data encoding chat or instructions for user management>

Fig. 6. Additional special data embedded in transmitted comments

documented in the software repository [14] and subject to change as new features are added to ChatConsole requiring the expansion of the encoding.

References

1. Attiya, H., Welch, J.: Distributed Computing: Fundamentals, Simulations, and Advanced Topics. Wiley Series on Parallel and Distributed Computing. Wiley, Hoboken (2004)
2. Bunt, A., Terry, M., Lank, E.: Challenges and opportunities for mathematics software in expert problem solving. *Hum.-Comput. Interact.* **28**(3), 222–264 (2013)
3. Cranshaw, J., Kittur, A.: The polymath project: lessons from a successful online collaboration in mathematics. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2011, pp. 1865–1874. ACM, New York (2011)
4. Davis, M., Sigal, R., Weyuker, E.J.: Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science. Elsevier, New York (1994)
5. Fantl, J.: Knowing-how and knowing-that. *Philos. Compass* **3**(3), 451–470 (2008)
6. Hollan, J., Hutchins, E., Kirsh, D.: Distributed cognition toward a new foundation for human-computer interaction research. *ACM Trans. Comput.-Hum. Interact.* **7**(2), 174–196 (2000)
7. Hutchins, E.: *Cognition in the Wild*. MIT Press, Cambridge (1995)
8. Kitcher, P.: *The Nature of Mathematical Knowledge*. Cambridge University Press, Cambridge (1984)
9. Kitzelmann, E.: Inductive programming: a survey of program synthesis techniques. In: Schmid, U., Kitzelmann, E., Plasmeijer, R. (eds.) AAIP 2009. LNCS, vol. 5812, pp. 50–73. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11931-6_3
10. Kohlhase, M.: The flexiformalist manifesto. In: 2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 30–35, September 2012
11. Mac Lane, S.: *Categories for the Working Mathematician* GTM, vol. 5. Springer, New York (1978). <https://doi.org/10.1007/978-1-4757-4721-8>
12. Martin, U., Pease, A.: Mathematical practice, crowdsourcing, and social machines. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (eds.) CICM 2013. LNCS (LNAI), vol. 7961, pp. 98–119. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39320-4_7
13. Minimair, M.: Collaborative computer algebra. In: Kotsireas, I., Martínez-Moro, E. (eds.) ACA 2015. Springer Proceedings in Mathematics & Statistics, vol. 198, pp. 289–303. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56932-1_20
14. Minimair, M.: ChatConsole, April 2018. <https://github.com/minicode/chconsole>
15. Moran, S., Nakata, K., Inoue, S.: Bridging the analytical gap between distributed cognition and actor network theory using a tool for information trajectory analysis. In: Proceedings of the 30th European Conference on Cognitive Ergonomics, ECCE 2012, pp. 72–77. ACM, New York (2012)
16. Nardi, B.A.: Studying context: a comparison of activity theory, situated action models, and distributed cognition. In: Nardi, B.A. (ed.) *Context and Consciousness*, pp. 69–102. MIT Press, Cambridge (1996)
17. Ragan-Kelley, M., Perez, F., Granger, B., Kluyver, T., Ivanov, P., Frederic, J., Bussonier, M.: The Jupyter/IPython architecture: a unified view of computational research, from interactive exploration to communication and publication. AGU Fall Meeting Abstracts 44, December 2014. <http://adsabs.harvard.edu/abs/2014AGUFM.H44D..07R>

18. Sage Foundation: Sage - Open-Source Mathematical Software System. <http://www.sagemath.org/>
19. Stahl, G.: Group Cognition. MIT Press, Cambridge (2006)
20. Stahl, G.: Studying Virtual Math Teams, vol. 11. Springer, Boston (2009). <https://doi.org/10.1007/978-1-4419-0228-3>



Automatically Finding Theory Morphisms for Knowledge Management

Dennis Müller¹(✉), Michael Kohlhase¹(✉), and Florian Rabe^{1,2}(✉) 

¹ Computer Science, FAU Erlangen-Nürnberg, Erlangen, Germany
{dennis.mueller,michael.kohlhase}@fau.de, florian.rabe@gmail.com

² LRI, Université Paris Sud, Orsay, France

Abstract. We present a method for finding morphisms between formal theories, both within as well as across libraries based on different logical foundations. As they induce new theorems in the target theory for any of the source theory, theory morphisms are high-value elements of a modular formal library. Usually, theory morphisms are manually encoded, but this practice requires authors who are familiar with source and target theories at the same time, which limits the scalability of the manual approach.

To remedy this problem, we have developed a morphism finder algorithm that automates theory morphism discovery. In this paper we present an implementation in the MMT system and show specific use cases. We focus on an application of *theory discovery*, where a user can check whether a (part of a) formal theory already exists in some library, potentially avoiding duplication of work or suggesting an opportunity for refactoring.

1 Introduction

Motivation. “Semantic Search” – a very suggestive term, which is alas seriously under-defined – has often been touted as the “killer application” of semantic technologies. With a view finder, we can add another possible interpretation: searching mathematical ontologies (here modular theorem prover libraries) at the level of theories – we call this **theory classification**.

The basic use case is the following: Jane, a mathematician, becomes interested in a class of mathematical objects, say – as a didactic example – something she initially calls “beautiful subsets” of a base set \mathcal{B} (or just “beautiful over \mathcal{B} ”). These have the following properties Q :

1. the empty set is beautiful over \mathcal{B}
2. every subset of a beautiful set is beautiful over \mathcal{B}
3. If A and B are beautiful over \mathcal{B} and A has more elements than B , then there is an $x \in A \setminus B$, such that $B \cup \{x\}$ is beautiful over \mathcal{B} .

To see what is known about beautiful subsets, she types these three conditions into a theory classifier, which computes any theories in a library \mathcal{L} that match

these (after a suitable renaming). In our case, Jane learns that her “beautiful sets” correspond to the well-known structure of matroids [MWP], so she can directly apply matroid theory to her problems.

In extended use cases, a theory classifier finds theories that share significant structure with Q , so that Jane can formalize Q modularly with minimal effort. Say Jane was interested in “dazzling subsets”, i.e. beautiful subsets that obey a fourth condition, then she could just contribute a theory that extends matroid by a formalization of the fourth condition – and maybe rethink the name.

In this paper we reduce the theory classification problem to the problem of finding theory morphisms (views) between theories in a library \mathcal{L} : given a query theory Q , the algorithm computes all (total) views from Q into \mathcal{L} and returns presentations of target theories and the assignments made by the views.

Related Work. Existing systems have so far only worked with explicitly given views, e.g., in IMPS [FGT93] or Isabelle [Pau94]. Automatically and systematically searching for new views was first undertaken in [NK07] in 2006. However, at that time no large corpora of formalized mathematics were available in standardized formats that would have allowed easily testing the ideas in practice.

This situation has changed since then as multiple such exports have become available. In particular, we have developed the MMT language [RK13] and the concrete syntax of the OMDoc XML format [Koh06] as a uniform representation language for such corpora. And we have translated multiple proof assistant libraries into this format, among others those of PVS in [Koh+17]. Building on these developments, we are now able, for the first time, to apply generic methods—i.e., methods that work at the MMT level—to search for views in these libraries.

While inspired by the ideas of [NK07], our design and implementation are completely novel. In particular, the theory makes use of the rigorous language-independent definitions of *theory* and *view* provided by MMT, and the practical implementation makes use of the MMT system, which provides high-level APIs for these concepts.

[GK14] applies techniques related to ours to a related problem. Instead of views inside a single corpus, they use machine learning to find similar constants in two different corpora. Their results can roughly be seen as a single partial view from one corpus to the other.

Approach and Contribution. Our contribution is twofold. Firstly, we present the design and implementation of a generic view finder that works with arbitrary corpora represented in MMT. The algorithm tries to match two symbols by unifying their types. This is made efficient by separating the term into a hashed representation of its abstract syntax tree (which serves as a fast plausibility check for pre-selecting matching candidates) and the list of symbol occurrences in the term, into which the algorithm recurses.

Secondly, we apply this view finder in two case studies: In the first, we start with an abstract theory and try to figure out if it already exists *in the same library* – the use case mentioned above. In the second example, we write down a

simple theory of commutative operators in one language to find all commutative operators in *another library based on a different foundation*.

Overview. In Sect. 2, we revise the basics of MMT and views. Section 3 presents the view finding algorithm restricted to the intra-library case and showcases it for the theory classification use case. In Sect. 4, we extend the algorithm to inter-library view finding discuss results of applying it to the PVS/NASA library. Section 5 concludes the paper and discusses additional applications.

A more extensive version of this paper with additional details can be found at [MRK].

2 Preliminaries: MMT and Views

Intuitively, MMT is a declarative language for theories and views over an arbitrary object language. Its treatment of object languages is abstract enough to subsume most logics and type theories which are practically relevant.

Figure 1 gives an overview of the fundamental MMT concepts. In the simplest case, **theories** Σ are lists of **constant declarations** $c : E$, where E is an expression that may use the previously declared constants. Naturally, E must be subject to some type system (which MMT is also parametric in), but the details of this are not critical for our purposes here. We say that Σ' includes Σ if it contains every constant declaration of Σ .

meta-theory: a fixed theory M		
	Theory Σ	View $\sigma : \Sigma \rightarrow \Sigma'$
set of	typed constant declarations $c : E$	assignments $c \mapsto E'$
Σ -expressions E	formed from M - and Σ -constants	mapped to Σ' expressions

Fig. 1. Overview of MMT concepts

Correspondingly, a **view** $\sigma : \Sigma \rightarrow \Sigma'$ is a list of **assignments** $c \mapsto e'$ of Σ' -expressions e' to Σ -constants c . To be well-typed, σ must preserve typing, i.e., we must have $\vdash_{\Sigma'} e' : \bar{\sigma}(E)$. Here $\bar{\sigma}$ is the homomorphic extension of σ , i.e., the map of Σ -expressions to Σ' -expressions that substitutes every occurrence of a Σ' -constant with the Σ' -expression assigned by σ . We call σ **simple** if the expressions e' are always Σ' -constants rather than complex expressions. The type-preservation condition for an assignment $c \mapsto c'$ reduces to $\bar{\sigma}(E) = E'$ where E and E' are the types of c and c' . We call σ **partial** if it does not contain an assignment for every Σ -constant and **total** otherwise. A partial view from Σ to Σ' is the same as a total view from some theory included by Σ to Σ' .

Importantly, we can then show generally at the MMT-level that if σ is well-typed, then $\bar{\sigma}$ preserves all typing and equality judgments over Σ . In particular, if we represent proofs as typed terms, views preserve the theoremhood of propositions. This property makes views so valuable for structuring, refactoring, and integrating large corpora.

MMT achieves language-independence through the use of **meta-theories**: every MMT-theory may designate a previously defined theory as its meta-theory. For example, when we represent the HOL Light library in MMT, we first write a theory L for the logical primitives of HOL Light. Then each theory in the HOL Light library is represented as a theory with L as its meta-theory. In fact, we usually go one step further: L itself is a theory, whose meta-theory is a logical framework such as LF. That allows L to concisely define the syntax and inference system of HOL Light.

However, for our purposes, it suffices to say that the meta-theory is some fixed theory relative to which all concepts are defined. Thus, we assume that Σ and Σ' have the same meta-theory M , and that $\bar{\sigma}$ maps all M -constants to themselves.

It remains to define the exact syntax of expressions. In the grammar on the right c refers to constants (of the meta-theory or previously declared in the current theory) and x refers to bound variables. Complex expressions are of the form $o[x_1 : t_1, \dots, x_m : t_m](a_1, \dots, a_n)$, where

$$\begin{aligned} \Gamma &::= (x : E)^* \\ E &::= c \mid x \mid E[\Gamma](E^+) \end{aligned}$$

- o is the operator that forms the complex expression,
- $x_i : t_i$ declares variable of type t_i that are bound by o in subsequent variable declarations and in the arguments,
- a_i is an argument of o .

The bound variable context may be empty, and we write $o(\mathbf{a})$ instead of $o[\cdot](\mathbf{a})$. For example, the axiom $\forall x : \mathbf{set}, y : \mathbf{set}. \mathbf{beautiful}(x) \wedge y \subseteq x \Rightarrow \mathbf{beautiful}(y)$ would instead be written as

$$\forall [x : \mathbf{set}, y : \mathbf{set}] (\Rightarrow (\wedge (\mathbf{beautiful}(x), \subseteq (y, x)), \mathbf{beautiful}(y)))$$

Finally, we remark on a few additional features of the MMT language that are important for large-scale case studies but not critical to understand the basic intuitions of results. MMT provides a module system that allows theories to instantiate and import each other. The module system is conservative: every theory can be *elaborated* into one that only declares constants. MMT constants may carry an optional definiens, in which case we write $c : E = e$. Defined constants can be eliminated by definition expansion.

3 Intra-Library View Finding

Let C be a corpus of theories with the same fixed meta-theory M . We call the problem of finding theory views between theories of C the **view finding problem** and an algorithm that solves it a **view finder**. Note that a view finder is sufficient to solve the theory classification use case from the introduction: Jane provides a M -theory Q of beautiful sets, the view finder computes all (total) views from Q into C .

Efficiency Considerations. The cost of this problem quickly explodes. First of all, it is advisable to restrict attention to simple views. Eventually we want to search for arbitrary views as well. But that problem is massively harder because it subsumes theorem proving: a view from Σ to Σ' maps Σ -axioms to Σ' -proofs, i.e., searching for a view requires searching for proofs.

Secondly, if C has n theories, we have n^2 pairs of theories between which to search. (It is exactly n^2 because the direction matters, and even views from a theory to itself are interesting.) Moreover, for two theories with m and n constants, there are n^m possible simple views. (It is exactly n^m because views may map different constants to the same one.) Thus, we can in principle enumerate and check all possible simple views in C . But for large C , it quickly becomes important to do so in an efficient way that eliminates ill-typed or uninteresting views early on.

Thirdly, it is desirable to search for *partial* views as well. In fact, identifying refactoring potential in libraries is only possible if we find partial views: then we can refactor the involved theories in a way that yields a total view. Moreover, many proof assistant libraries do not follow the little theories paradigm or do not employ any theory-like structuring mechanism at all. These can only be represented as a single huge theory, in which case we have to search for partial views from this theory to itself. While partial views can be reduced to and then checked like total ones, searching for partial views makes the number of possible views that must be checked much larger.

Finally, even for a simple view, checking reduces to a set of equality constraints, namely the constraints $\vdash_{\Sigma'} \bar{\sigma}(E) = E'$ for the type-preservation condition. Depending on M , this equality judgment may be undecidable and require theorem proving.

Algorithm Overview. A central motivation for our algorithm is that equality in M can be soundly approximated very efficiently by using a normalization function on M -expressions. This has the additional benefit that relatively little meta-theory-specific knowledge is needed, and all such knowledge is encapsulated in a single well-understood function. This way we can implement view-search generically for arbitrary M .

Our algorithm consists of two steps. First, we preprocess all constant declarations in C with the goal of moving as much intelligence as possible into a step whose cost is linear in the size of C . Then, we perform the view search on the optimized data structures produced by the first step.

3.1 Preprocessing

The preprocessing phase computes for every constant declaration $c : E$ a normal form E' and then efficiently stores the abstract syntax tree of E' . Both steps are described below.

Normalization involves two steps: **MMT-level normalization** performs generic transformations that do not depend on the meta-theory M . These include

elaboration of structured theories and definition expansion, which we mentioned in Sect. 2. Importantly, we do not fully eliminate defined constant declarations $c : E = e$ from a theory Σ : instead, we replace them with primitive constants $c : E$ and replace every occurrence of c in other declarations with e . If Σ is the domain theory, we can simply ignore $c : E$ (because views do not have to provide an assignment to defined constants). But if the Σ is the codomain theory, retaining $c : E$ increases the number of views we can find; in particular in situations where E is a type of proofs, and hence c a theorem.

Meta-theory-level normalization applies an M -specific normalization function. In general, we assume this normalization to be given as a black box. However, because many practically important normalization steps are widely reusable, we provide a few building blocks, from which specific normalization functions can be composed. Skipping the details, these include:

1. Top-level universal quantifiers and implications are rewritten into the function space of the logical framework using the Curry-Howard correspondence.
2. The order of curried domains of function types is normalized as follows: first all dependent arguments types ordered by the first occurrence of the bound variables; then all non-dependent argument types A ordered by the abstract syntax tree of A .
3. Implicit arguments, whose value is determined by the values of the others are dropped, e.g. the type argument of an equality. This has the additional benefit of shrinking the abstract syntax trees and speeding up the search.
4. Equalities are normalized such that the left hand side has a smaller abstract syntax tree.

Above multiple normalization steps make use of a total order on abstract syntax trees. We omit the details and only remark that we try to avoid using the names of constants in the definition of the order—otherwise, declarations that could be matched by a view would be normalized differently. Even when breaking ties between requires comparing two constants, we can first try to recursively compare the syntax trees of their types.

Abstract Syntax Trees. We define **abstract syntax trees** as pairs (t, s) where t is subject to the grammar

$$t ::= C_{Nat} \mid V_{Nat} \mid t [t^+] (t^+)$$

(where Nat is a non-terminal for natural numbers) and s is a list of constant names.

We obtain an abstract syntax tree from an MMT expression E by (i) switching to de-Bruijn representation of bound variables and (ii) replacing all occurrences of constants with C_i in such a way that every C_i refers to the i -th element of s .

Abstract syntax trees have the nice property that they commute with the application of simple views σ : If (t, s) represents E , then $\sigma(E)$ is represented by (t, s') where s' arises from s by replacing every constant with its σ -assignment.

The above does not completely specify i and s yet, and there are several possible canonical choices among the abstract syntax trees representing the same

expression. The trade-off is subtle because we want to make it easy to both identify and check views later on. We call (t, s) the **long** abstract syntax tree for E if C_i replaces the i -th occurrence of a constant in E when E is read in left-to-right order. In particular, the long tree does not merge duplicate occurrences of the same constant into the same number. The **short** abstract syntax tree for E arises from the long one by removing all duplicates from s and replacing the C_i accordingly.

Example 1. Consider again the axiom $\forall x : \mathbf{set}, y : \mathbf{set}. \mathbf{beautiful}(x) \wedge y \subseteq x \Rightarrow \mathbf{beautiful}(y)$ with internal representation

$$\forall [x : \mathbf{set}, y : \mathbf{set}] (\Rightarrow (\wedge (\mathbf{beautiful}(x), \subseteq (y, x)), \mathbf{beautiful}(y))).$$

The *short* syntax tree and list of constants associated with this term would be:

$$\begin{aligned} t &= C_1 [C_2, C_2] (C_3 (C_4 (C_5 (V_2), C_6 (V_1, V_2)), C_5 (V_1))) \\ s &= (\forall, \mathbf{set}, \Rightarrow, \wedge, \mathbf{beautiful}, \subseteq) \end{aligned}$$

The corresponding long syntax tree is:

$$\begin{aligned} t &= C_1 [C_2, C_3] (C_4 (C_5 (C_6 (V_2), C_7 (V_1, V_2)), C_8 (V_1))) \\ s &= (\forall, \mathbf{set}, \mathbf{set}, \Rightarrow, \wedge, \mathbf{beautiful}, \subseteq, \mathbf{beautiful}) \end{aligned}$$

For our algorithm, we pick the *long* abstract syntax tree, which may appear surprising. The reason is that shortness is not preserved when applying a simple view: whenever a view maps two different constants to the same constant, the resulting tree is not short anymore. Length, on the other hand, is preserved. The disadvantage that long trees take more time to traverse is outweighed by the advantage that we never have to renormalize the trees.

3.2 Search

Consider two constants $c : E$ and $c' : E'$, where E and E' are preprocessed into long abstract syntax trees (t, s) and (t', s') . It is now straightforward to show the following Lemma:

Lemma 1. *The assignment $c \mapsto c'$ is well-typed in a view σ if $t = t'$ (in which case s and s' must have the same length l) and σ also contains $s_i \mapsto s'_i$ for $i = 1, \dots, l$.*

Of course, the condition about $s_i \mapsto s'_i$ may be redundant if s contain duplicates; but because s has to be traversed anyway, it is cheap to skip all duplicates. We call the set of assignments $s_i \mapsto s'_i$ the **prerequisites** of $c \mapsto c'$.

This lemma is the center of our search algorithm explained in

Lemma 2 (Core Algorithm). *Consider two constant declarations c and c' in theories Σ and Σ' . We define a view by starting with $\sigma = c \mapsto c'$ and recursively adding all prerequisites to σ until*

- either the recursion terminates
- or σ contains two different assignments for the same constant, in which case we fail.

If the above algorithm succeeds, then σ is a well-typed partial simple view from Σ to Σ' .

Example 2. Consider two constants c and c' with types $\forall x : \mathbf{set}, y : \mathbf{set}. \mathbf{beautiful}(x) \wedge y \subseteq x \Rightarrow \mathbf{beautiful}(y)$ and $\forall x : \mathbf{powerset}, y : \mathbf{powerset}. \mathbf{finite}(x) \wedge y \subseteq x \Rightarrow \mathbf{finite}(y)$. Their syntax trees are

$$\begin{aligned}
 t = t' &= C_1 [C_2, C_3] (C_4 (C_5 (C_6 (V_2), C_7 (V_1, V_2)), C_8 (V_1))) \\
 s &= (\forall, \mathbf{set}, \mathbf{set}, \Rightarrow, \wedge, \mathbf{beautiful}, \subseteq, \mathbf{beautiful}) \\
 s' &= (\forall, \mathbf{powerset}, \mathbf{powerset}, \Rightarrow, \wedge, \mathbf{finite}, \subseteq, \mathbf{finite})
 \end{aligned}$$

Since $t = t'$, we set $c \mapsto c'$ and compare s with s' , meaning we check (ignoring duplicates) that $\forall \mapsto \forall, \mathbf{set} \mapsto \mathbf{powerset}, \Rightarrow \mapsto \Rightarrow, \wedge \mapsto \wedge, \mathbf{beautiful} \mapsto \mathbf{finite}$ and $\subseteq \mapsto \subseteq$ are all valid.

To find all views from Σ to Σ' , we first run the core algorithm on every pair of Σ -constants and Σ' -constants. This usually does not yield big views yet. For example, consider the typical case where theories contain some symbol declarations and some axioms, in which the symbols occur. Then the core algorithm will only find views that map at most one axiom.

Depending on what we intend to do with the results, we might prefer to consider them individually (e.g. to yield *alignments* in the sense of [Kal+16]). But we can also use these small views as building blocks to construct larger, possibly total ones:

Lemma 3 (Amalgamating Views). *We call two partial views **compatible** if they agree on all constants for which both provide an assignment.*

The union of compatible well-typed views is again well-typed.

Example 3. Consider the partial view from Example 2 and imagine a second partial view for the axioms $\mathbf{beautiful}(\emptyset)$ and $\mathbf{finite}(\emptyset)$. The former has the requirements

$$\forall \mapsto \forall, \quad \mathbf{set} \mapsto \mathbf{powerset} \quad \Rightarrow \mapsto \Rightarrow \quad \wedge \mapsto \wedge \quad \mathbf{beautiful} \mapsto \mathbf{finite} \quad \subseteq \mapsto \subseteq$$

The latter requires only $\mathbf{set} \mapsto \mathbf{powerset}$ and $\emptyset \mapsto \emptyset$. Since both views agree on all assignments, we can merge all of them into a single view, mapping both axioms and all requirements of both.

3.3 Optimizations

The above presentation is intentionally simple to convey the general idea. We now describe a few advanced features of our implementation to enhance scalability.

Caching Preprocessing Results. Because the preprocessing performs normalization, it can be time-consuming. Therefore, we allow for storing the preprocessing results to disk and reloading them in a later run.

Fixing the Meta-Theory. We improve the preprocessing in a way that exploits the common meta-theory, which is meant to be fixed by every view. All we have to do is, when building the abstract syntax trees (t, s) , to retain all references to constants of the meta-theory in t instead of replacing them with numbers. With this change, s will never contain meta-theory constants, and the core algorithm will only find views that fix all meta-theory constants. Because s is much shorter now, the view search is much faster.

It is worth pointing out that the meta-theory is not always as fixed as one might think. Often we want to consider to be part of the meta-theory certain constants that are defined early on in the library and then used widely. In PVS, this makes sense, e.g., for all operations define in the Prelude library (the small library shipped with PVS). Note that we still only have to cache one set of preprocessing results for each library: changes to the meta-theory only require minor adjustments to the abstract syntax trees without redoing the entire normalization.

Biasing the Core Algorithm. The core algorithm starts with an assignment $c \mapsto c'$ and then recurses into constant that occur in the declarations of c and c' . This occurs-in relation typically splits the constants into layers. A typical theory declares types, which then occur in the declarations of function symbols, which then occur in axioms. Because views that only map type and function symbols are rarely interesting (because they do not allow transporting non-trivial theorems), we always start with assignments where c is an axiom.

Exploiting Theory Structure. Libraries are usually highly structured using imports between theories. If Σ is imported into Σ' , then the set of partial views out of Σ' is a superset of the set of partial views out of Σ . If implemented naively, that would yield a quadratic blow-up in the number of views to consider.

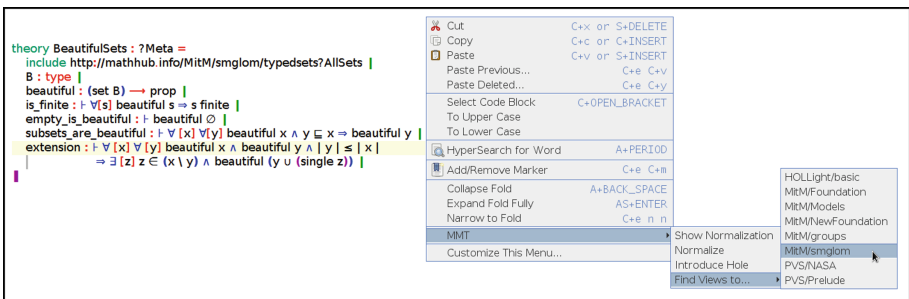


Fig. 2. “Beautiful sets” in MMT surface syntax

Instead, when running our algorithm on an entire library, we only consider views between theories that are not imported into other theories. In an additional postprocessing phase, the domain and codomain of each found partial view σ are adjusted to the minimal theories that make σ well-typed.

3.4 Implementation

We have implemented our view finder algorithm in the MMT system. A screenshot of Jane’s theory of beautiful sets is given in Fig. 2. Right-clicking anywhere within the theory allows Jane to select MMT \rightarrow Find Views to... \rightarrow MitM/smgglom. The latter menu offers a choice of known libraries in which the view finder should look for codomain theories; MitM/smgglom is the Math-in-the-Middle library based that we have developed [Deh+16] to describe the common knowledge used in various CICM systems.



Fig. 3. Views found for “beautiful sets”

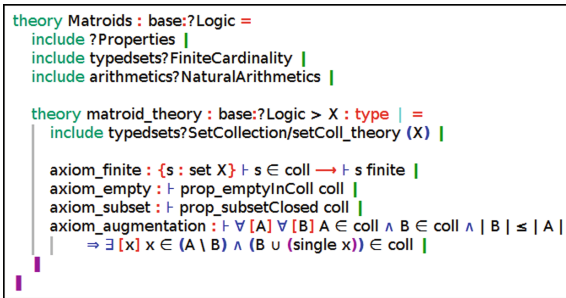


Fig. 4. The theory of matroids in the MitM library

After choosing MitM/smgglom, the view finder finds two views (within less than one second) and shows them (Fig. 3). The first of these (View1) has a theory for matroids as its codomain, which is given in Fig. 4. Inspecting that theory and the assignments in the view, we see that it indeed represents the well-known correspondence between beautiful sets and matroids.

4 Inter-Library View Finding

We now generalize to view finding to different libraries written in different logics. Intuitively, the key idea is that we now have two fixed meta-theories M and M' and a fixed meta-view $m : M \rightarrow M'$. However, due to the various idiosyncrasies of logics, tools’ library structuring features, individual library conventions, this problem is significantly more difficult than *intra*-library view finding. For example, unless the logics are closely related, meta-views usually do not even exist and

must be approximated. Therefore, a lot of tweaking is typically necessary, and it is possible that multiple runs with different trade-offs give different interesting results.

As an example, we present a large case study where we find views from the MitM library used in the running example so far into the PVS/NASA library.

4.1 The PVS/NASA Library

PVS [ORS92] is a proof assistant under active development based on a higher-order logic with a number of advanced features. In addition to the *Prelude* library, which contains the most common domains of mathematical discourse and is shipped with PVS itself, there is a large library of formal mathematics developed and maintained by NASA [PVS]. In [Koh+17], we represent PVS as a meta-theory in MMT and implemented a translator that transforms both libraries into MMT format. We use a meta-view that embeds MitM’s higher-order logic into PVS’s higher-order logic and make sure that we normalize PVS-formulas in the same way as MitM-formulas.

Theory Structure Normalization. PVS’s complex and prevalently used parametric theories critically affect view finding because they affect the structure of theories. For example, the theory of groups `group_def` in the NASA library has three theory parameters (`T`, `*`, `one`) for the signature of groups, and includes the theory `monoid_def` with the same parameters, and then declares the axioms for a group in terms of these parameters. Without special treatment, we could only find views from/into libraries that use the same theory structure.

We have investigated three approaches of handling parametric theories:

1. *Simple treatment:* We drop theory parameters and interpret references to them as free variables that match anything. This is of course not sound so that all found views must be double-checked. However, because practical search problems often do not require exact results, even returning all potential views can be useful.
2. *Covariant elimination:* We treat theory parameters as if they were constants declared in the body. In the above mentioned theory `group_def`, we would hence add three new constants `T`, `*` and `one` with their corresponding types. This works well in the common case where a parametric theory is not used with two different instantiations in the same context.
3. *Contravariant elimination:* The theory parameters are treated as if they were bound separately for every constant in the body of the theory. In the above mentioned theory `group_def`, we would change e.g. the unary predicate `inverse.exists?` with type $T \rightarrow \text{bool}$ to a function with type $(T : \text{pvstype}) \rightarrow (* : T \rightarrow T \rightarrow T) \rightarrow (\text{one} : T) \rightarrow (T \rightarrow \text{bool})$. This is closest to the actual semantics of the PVS module system. But it makes finding interesting views the least likely because it is the most sensitive to the modular structure of individual theories.

We have implemented the first two approaches. The first is the most straightforward but it leads to many false positives and false negatives. We have found the second approach to be the most useful for inter-library search since it most closely corresponds to simple formalizations of abstract theories in other libraries. The third approach will be our method of choice when investigating *intra*-library views of PVS/NASA in future work.

4.2 Implementation

As a first use case, we can write down a theory for a commutative binary operator using the MitM foundation, while targeting the PVS Prelude library – allowing us to find all commutative operators, as in Fig. 5 (using the simple approach to theory parameters).

```

theory CommTest : mitm:?Logic =
  A : type |
  op : A → A → A |
  comm : ⊢ ∀ [x] ∀ [y] op x y = op y x |

```

From: <http://cds.omdoc.org/testcases?CommTest>
 To: PVS/Prelude
 4 Results found:

View3 : CommTest -> finite_sets_sum
 A = finite_sets_sum?Parameter/R
 op = finite_sets_sum?Parameter/+
 comm = finite_sets_sum?plus_comm

View2 : CommTest -> finite_sets_product
 A = finite_sets_product?Parameter/R
 op = finite_sets_product?Parameter/*
 comm = finite_sets_product?mult_comm

Fig. 5. Searching for commutative operators in PVS

This example also hints at a way to iteratively improve the results of the view finder: since we can find properties like commutativity and associativity, we can use the results to in turn inform a better normalization of the theory by exploiting these properties. This in turn would potentially allow for finding more views.

To evaluate the approaches to theory parameters we used a simple theory of monoids in the MitM foundation and the theory of monoids in the NASA library as domains for viewfinding with the whole NASA library as target using simple and covariant approaches. The results are summarized in Fig. 6.

Domain	Normalization	Simple Views	Aggregated
NASA/monoid	simple	388	154
MitM/monoid	simple	32	17
NASA/monoid	covariant	1026	566
MitM/monoid	covariant	22	6

Fig. 6. Results of inter- and intra-library view finding in the PVS NASA library

Most of the results in the simple MitM \rightarrow NASA case are artifacts of the theory parameter treatment and view amalgamation – in fact only two of the 17 results are meaningful (to operations on sets and the theory of number fields). In the covariant case, the additional requirements lead to fuller (one total) and less spurious views. With a theory from the NASA library as domain, the results are already too many to be properly evaluated by hand. With the simple approach to theory parameters, most results can be considered artifacts; in the covariant case, the most promising results yield (partial) views into the theories of semigroups, rings (both the multiplicative and additive parts) and most extensions thereof (due to the duplication of theory parameters as constants).

5 Conclusion

We present a general MKM utility that given a MMT theory and an MMT library \mathcal{L} finds partial and total views into \mathcal{L} . Such a view finder can be used to drive various MKM applications ranging from theory classification to library merging and refactoring. The theory discovery use case described in Sect. 3.4 is mostly desirable in a setting where a user is actively writing or editing a theory, so the integration in jEdit is sensible. However, the inter-library view finding would be a lot more useful in a theory exploration setting, such as when browsing available archives on MathHub [Ian+14] or in the graph viewer integrated in MMT [RKM17].

Future Work. The current view finder is already efficient enough for the limited libraries we used for testing. To increase efficiency, we plan to explore term indexing techniques [Gra96] that support $1 : n$ and even $n : m$ matching and unification queries. The latter will be important for the library refactoring and merging applications which look for all possible (partial and total) views in one or between two libraries. As such library-scale operations will have to be run together with theory flattening to a fixed point and re-run upon every addition to the library, it will be important to integrate them with the MMT build system and change management processes [AM10, Ian12].

Enabled Applications. Our work enables a number of advanced applications. Maybe surprisingly, a major bottleneck here concerns less the algorithm or software design challenges but user interfaces and determining the right application context.

- **Model-/Countermodel Finding:** If the codomain of a view is a theory representing a specific model, it would tell Jane that those are *examples* of her abstract theory. Furthermore, partial views – especially those that are total on some included theory – could lead to insightful *counterexamples*.
- **Library Refactoring:** Given that the view finder looks for *partial* views, we can use it to find natural extensions of a starting theory. Imagine Jane removing the last of her axioms for “beautiful sets” – the other axioms (disregarding finiteness of her sets) would allow her to find e.g. both Matroids and *Ideals*, which would suggest to her to possibly refactor her library such that both extend her new theory. Additionally, *surjective* partial views would inform her, that her theory would probably better be refactored as an extension of the codomain, which would allow her to use all theorems and definitions therein.
- **Theory Generalization:** If we additionally consider views into and out of the theories found, this can make theory discovery even more attractive. For example, a view from a theory of vector spaces into matroids could inform Jane additionally, that her beautiful sets, being matroids, form a generalization of the notion of linear independence in linear algebra.
- **Folklore-based Conjecturing:** If we have theory T describing (the properties of) a class O of objects under consideration and a view $v : S \rightsquigarrow T$, then we can use extensions of S' in \mathcal{L} with $\iota : S \hookrightarrow S'$ for making conjectures about O : The v -images of the local axioms of S' would make useful properties to establish about O , since they allow pushing out v over ι to a view $v' : S' \rightsquigarrow T'$ (where T' extends T by the newly imported properties) and gain $v'(S')$ as properties of O . Note that we would need to keep book on our transformations during preprocessing and normalization, so that we could use the found views for translating both into the codomain as well as back from there into our starting theory. A useful interface might specifically prioritize views into theories on top of which there are many theorems and definitions that have been discovered.

Note that even though the algorithm is in principle symmetric, some aspects often depend on the direction—e.g. how we preprocess the theories, which constants we use as starting points or how we aggregate and evaluate the resulting (partial) views (see Sects. 3.3, 3.1 and 4.1).

Acknowledgments. The authors gratefully acknowledge financial support from the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541) and the DFG-funded project OAF: An Open Archive for Formalizations (KO 2428/13-1).

References

- [AM10] Autexier, S., Müller, N.: Semantics-based change impact analysis for heterogeneous collections of documents. In: Gormish, M., Ingold, R. (eds.) Proceedings of the 10th ACM Symposium on Document Engineering, DocEng 2010, Manchester, United Kingdom, pp. 97–106. ACM (2010). <https://doi.org/10.1145/1860559.1860580>
- [Deh+16] Dehaye, P.-O., et al.: Interoperability in the OpenDreamKit project: the math-in-the-middle approach. In: Kohlhase, Michael, Johansson, Moa, Miller, Bruce, de Moura, Leonardo, Tompa, Frank (eds.) CICM 2016. LNCS (LNAI), vol. 9791, pp. 117–131. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42547-4_9
- [FGT93] Farmer, W., Guttman, J., Thayer, F.: IMPS: an interactive mathematical proof system. *J. Autom. Reas.* **11**(2), 213–248 (1993)
- [GK14] Gauthier, T., Kaliszyk, C.: Matching concepts across HOL libraries. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) CICM 2014. LNCS (LNAI), vol. 8543, pp. 267–281. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08434-3_20
- [Gra96] Graf, P. (ed.): Term Indexing. LNCS, vol. 1053. Springer, Heidelberg (1995). <https://doi.org/10.1007/3-540-61040-5>
- [Ian+14] Iancu, M., Jucovschi, C., Kohlhase, M., Wiesing, T.: System description: MathHub.info. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) CICM 2014. LNCS (LNAI), vol. 8543, pp. 431–434. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08434-3_33
- [Ian12] Iancu, M.: Management of change in declarative languages. MA thesis. Jacobs University Bremen (2012)
- [Kal+16] Kaliszyk, C., Kohlhase, M., Müller, D., Rabe, F.: A standard for aligning mathematical concepts. In: Kohlhase, A., Kohlhase, M., Libbrecht, P., Miller, B., Tompa, F., Naummowicz, A., Neuper, W., Quaresma, P., Suda, M. (eds.) Work in Progress at CICM 2016, pp. 229–244. CEUR-WS.org (2016)
- [Koh+17] Kohlhase, M., Müller, D., Owre, S., Rabe, F.: Making PVS accessible to generic services by interpretation in a universal format. In: Ayala-Rincón, M., Muñoz, C.A. (eds.) ITP 2017. LNCS, vol. 10499, pp. 319–335. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66107-0_21
- [Koh06] Kohlhase, M.: OMDoc An Open Markup Format for Mathematical Documents (Version 1.2). LNAI, vol. 4180. Springer, Heidelberg (2006). <https://doi.org/10.1007/11826095>
- [MRK] Müller, D., Rabe, F., Kohlhase, M.: Automatically Finding Theory Morphisms for Knowledge Management. <http://kwarc.info/kohlhase/submit/viewfinder-report.pdf>
- [MWP] Matroid – Wikipedia, The Free Encyclopedia. <https://en.wikipedia.org/w/index.php?title=Matroid>, Accessed 04 Apr 2018
- [NK07] Normann, I., Kohlhase, M.: Extended formula normalization for ε -retrieval and sharing of mathematical knowledge. In: Kauer, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) Calculemus/MKM -2007. LNCS (LNAI), vol. 4573, pp. 356–370. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73086-6_28
- [ORS92] Owre, S., Rushby, J.M., Shankar, N.: PVS: a prototype verification system. In: Kapur, D. (ed.) CADE 1992. LNCS, vol. 607, pp. 748–752. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55602-8_217

- [Pau94] Paulson, L.C. (ed.): Isabelle: A Generic Theorem Prover. LNCS, vol. 828. Springer, Heidelberg (1994). <https://doi.org/10.1007/BFb0030541>
- [PVS] NASA PVS Library. <http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/>. Accessed 17 Dec 2014
- [RK13] Rabe, F., Kohlhase, M.: A scalable module system. *Inf. Comput.* **230**(1), 1–54 (2013)
- [RKM17] Rupprecht, M., Kohlhase, M., Müller, D.: A flexible, interactive theory-graph viewer. In: Kohlhase, A., Pollanen, M. (eds.) *The 12th Workshop on Mathematical User Interfaces, MathUI 2017* (2017). <http://kwarc.info/kohlhase/papers/mathui17-tgview.pdf>



Goal-Oriented Conjecturing for Isabelle/HOL

Yutaka Nagashima^{1,2} and Julian Parsert²(✉)

¹ CIIRC, Czech Technical University in Prague, Prague, Czech Republic

² Department of Computer Science, University of Innsbruck, Innsbruck, Austria
{yutaka.nagashima,julian.parsert}@uibk.ac.at

Abstract. We present PGT, a Proof Goal Transformer for Isabelle/HOL. Given a proof goal and its background context, PGT attempts to generate conjectures from the original goal by transforming the original proof goal. These conjectures should be weak enough to be provable by automation but sufficiently strong to identify and prove the original goal. By incorporating PGT into the pre-existing PSL framework, we exploit Isabelle's strong automation to identify and prove such conjectures.

1 Introduction

Consider the following two reverse functions defined in literature [9]:

```
primrec itrev :: "'a list 'a list 'a list" where
  "itrev [] ys = ys" | "itrev (x#xs) ys = itrev xs (x#ys)"
primrec rev :: "'a list 'a list" where
  "rev [] = []" | "rev (x # xs) = rev xs @ [x]"
```

How would you prove their equivalence "itrev xs [] = rev xs"? Induction comes to mind. However, it turns out that Isabelle's default proof methods, `induct` and `induct_tac`, are unable to handle this proof goal effectively.

Previously, we developed PSL [8], a programmable, meta-tool framework for Isabelle/HOL. With PSL one can write the following strategy for induction:

```
strategy DInd = Thens [Dynamic (Induct), Auto, IsSolved]
```

PSL's `Dynamic` keyword creates variations of the `induct` method by specifying different combinations of promising arguments found in the proof goal and its background proof context. Then, `DInd` combines these induction methods with the general purpose proof method, `auto`, and `is_solved`, which checks if there is

Y. Nagashima—Supported by the European Regional Development Fund under the project AI & Reasoning (reg. no.CZ.02.1.01/0.0/0.0/15_003/0000466)

J. Parsert—Supported by the European Research Council (ERC) grant no 714034 SMART.

any proof goal left after applying `auto`. As shown in Fig. 1a, PSL keeps applying the combination of a specialization of `induct` method and `auto`, until either `auto` discharges all remaining sub-goals or `DInd` runs out of the variations of `induct` methods as shown in Fig. 1a.

This approach works well only if the resulting sub-goals after applying some `induct` are easy enough for Isabelle’s automated tools (such as `auto` in `DInd`) to prove. When proof goals are presented in an automation-unfriendly way, however, it is not enough to set a certain combination of arguments to the `induct` method. In such cases engineers have to investigate the original goal and come up with auxiliary lemmas, from which they can derive the original goal.

In this paper, we present `PGT`, a novel design and prototype implementation¹ of a conjecturing tool for Isabelle/HOL. We provide `PGT` as an extension to `PSL` to facilitate the seamless integration with other Isabelle sub-tools. Given a proof goal, `PGT` produces a series of conjectures that might be useful in discharging the original goal, and `PSL` attempts to identify the right one while searching for a proof of the original goal using those conjectures.

2 System Description

2.1 Identifying Valuable Conjectures via Proof Search

To automate conjecturing, we added the new language primitive, `Conjecture` to `PSL`. Given a proof goal, `Conjecture` first produces a series of conjectures that might be useful in proving the original theorem, following the process described in Sect. 2.2. For each conjecture, `PGT` creates a `subgoal_tac` method and inserts the conjecture as the premise of the original goal. When applied to `"itrev xs [] = rev xs"`, for example, `Conjecture` generates the following proof method along with 130 other variations of the `subgoal_tac` method:

```
apply (subgoal_tac "!!Nil. itrev xs Nil = rev xs @ Nil")
```

where `!!` stands for the universal quantifier in Isabelle’s meta-logic. Namely, `Conjecture` introduced a variable of name `Nil` for the constant `[]`. Applying this method to the goal results in the following two new sub-goals:

1. `(!!Nil. itrev xs Nil = rev xs @ Nil) ==> itrev xs [] = rev xs`
2. `!!Nil. itrev xs Nil = rev xs @ Nil`

`Conjecture` alone cannot determine which conjecture is useful for the original goal. In fact, some of the generated statements are not even true or provable. To discard these non-theorems and to reduce the size of `PSL`’s search space, we combine `Conjecture` with `Fastforce` (corresponding to the `fastforce` method) and `Quickcheck` (corresponding to Isabelle’s sub-tool `quickcheck` [3]) sequentially as well as `DInd` as follows:

¹ Available at Github <https://github.com/data61/PSL/releases/tag/v0.1.1>. The example of this paper appears in `PSL/PGT/Example.thy`.

```
strategy CDInd = Thens [Conjecture, Fastforce, Quickcheck, DInd]
```

Importantly, `fastforce` does not return an intermediate proof goal: it either discharges the first sub-goal completely or fails by returning an empty sequence. Therefore, whenever `fastforce` returns a new proof goal to a sub-goal resulting from `subgoal_tac`, it guarantees that the conjecture inserted as a premise is strong enough for Isabelle to prove the original goal. In our example, the application of `fastforce` to the aforementioned first sub-goal succeeds, changing the remaining sub-goals to the following:

```
1. !!Nil. itrev xs Nil = rev xs @ Nil
```

However, PSL still has to deal with many non-theorems: non-theorems are often strong enough to imply the original goal due to the principle of explosion. Therefore, `CDInd` applies `Quickcheck` to discard easily refutable non-theorems. The atomic strategy `Quickcheck` returns the same sub-goal only if Isabelle's sub-tool `quickcheck` does not find a counter example, but returns an empty sequence otherwise.

Now we know that the remaining conjectured goals are strong enough to imply the original goal and that they are not easily refutable. Therefore, `CDInd` applies its sub-strategy `DInd` to the remaining sub-goals and it stops its proof search as soon as it finds the following proof script, which will be printed in Isabelle/jEdit's output panel.

```
apply (subgoal_tac "!!Nil. itrev xs Nil = rev xs @ Nil")
apply fastforce apply (induct xs) apply auto done
```

Figure 1b shows how `CDInd` narrows its search space in a top-down manner. Note that PSL lets you use other Isabelle sub-tools to prune conjectures. For example, you can use both `nitpick` [1] and `quickcheck: Thens [Quickcheck, Nitpick]` in `CDInd`. It also let you combine `DInd` and `CDInd` into one: `Ors [DInd, CDInd]`.

2.2 Conjecturing

Section 2.1 has described how we identify useful conjectures. Now, we will focus on how PGT creates conjectures in the first place. PGT introduced both automatic conjecturing (`Conjecture`) and automatic generalization (`Generalize`). Since the conjecturing functionality uses generalization, we will only describe the former. We now walk through the main steps that lead from a user defined goal to a set of potentially *useful* conjectures, as illustrated in Fig. 2. We start with the extraction of constants and sub-terms, continue with generalization, goal oriented conjecturing, and finally describe how the resulting terms are sanitized.

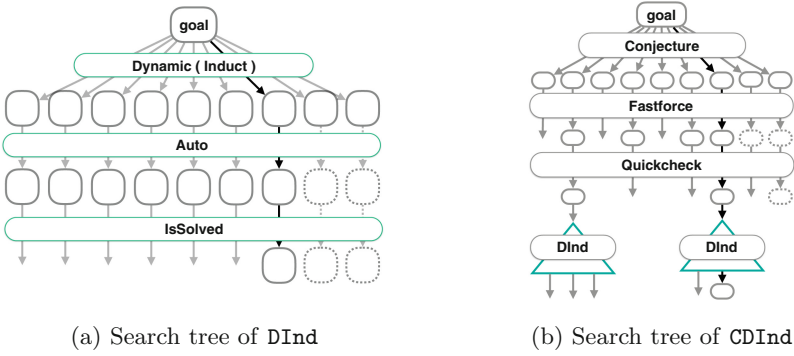


Fig. 1. PSL’s proof search with/without PGT.

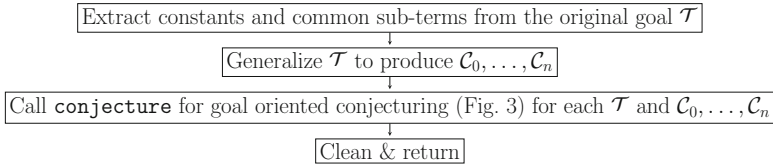


Fig. 2. The overall workflow of Conjecture.

Extraction of Constants and Common Sub-terms. Given a term representation \mathcal{T} of the original goal, PGT extracts the constants and sub-terms that appear multiple times in \mathcal{T} . In the example from Sect. 1, PGT collects the constants `rev`, `itrev`, and `[]`.

Generalization. Now, PGT tries to generalize the goal \mathcal{T} . Here, PGT alone cannot determine over which constant or sub-terms it should generalize \mathcal{T} . Hence, it creates a generalized version of \mathcal{T} for each constant and sub-term collected in the previous step. For `[]` in the running example, PGT creates the following generalized version of \mathcal{T} : `!!Nil. itrev xs Nil = rev xs`.

Goal Oriented Conjecturing. This step calls the function `conjecture`, illustrated in Fig. 3, with the original goal \mathcal{T} and each of the generalized versions of \mathcal{T} from the previous step ($\mathcal{C}_0, \dots, \mathcal{C}_n$). The following code snippet shows part of `conjecture`:

```

fun cnjcts t = flat (map (get_cnjct generalisedT t) consts)
fun conj (trm as Abs (_,_,subtrm)) = cnjcts trm @ conj subtrm
  | conj (trm as App (t1,t2)) = cnjcts trm @ conj t1 @ conj t2
  | conj trm = cnjcts trm

```

For each \mathcal{T} and \mathcal{C}_i for $0 \leq i \leq n$, `conjecture` first calls `conj`, which traverses the term structure of each \mathcal{T} or \mathcal{C}_i in a top-down manner. In the running

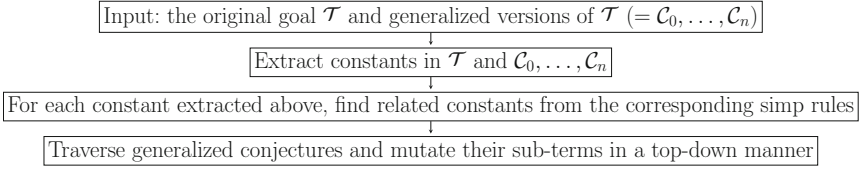


Fig. 3. The workflow of the `conjecture` function.

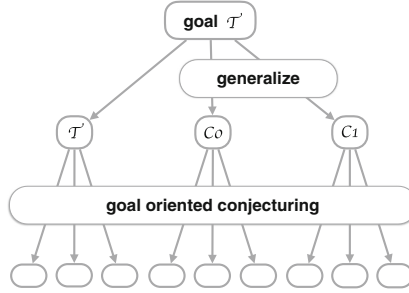


Fig. 4. PSL’s sequential generalization and goal oriented conjecturing.

example, PGT takes some \mathcal{C}_k , say `!!Nil. itrev xs Nil = rev xs`, as an input and applies `conj` to it.

For each sub-term the function `get_cnjct` in `cnjcts` creates new conjectures by replacing the sub-term (`t` in `cnjcts`) in \mathcal{T} or \mathcal{C}_i (`generalisedT`) with a new term. This term is generated from the sub-term (`t`) and the constants (`consts`). These are obtained from simplification rules that are automatically derived from the definition of a constant that appears in the corresponding \mathcal{T} or \mathcal{C}_i .

In the example, PGT first finds the constant `rev` within \mathcal{C}_k . Then, PGT finds the simp-rule (`rev.simps(2)`) relevant to `rev` which states, `rev (?x # ?xs) = rev ?xs @ [?x]`, in the background context. Since `rev.simps(2)` uses the constant `@`, PGT attempts to create new sub-terms using `@` while traversing in the syntax tree of `!!Nil. itrev xs Nil = rev xs` in a top-down manner.

When `conj` reaches the sub-term `rev xs`, `get_cnjct` creates new sub-terms using this sub-term, `@` (an element in `consts`), and the universally quantified variable `Nil`. One of these new sub-terms would be `rev xs @ Nil`². Finally, `get_cnjct` replaces the original sub-term `rev xs` with this new sub-term in \mathcal{C}_k , producing the conjecture: `!!Nil. itrev xs Nil = rev xs @ Nil`.

Note that this conjecture is not the only conjecture produced in this step: PGT, for example, also produces `!!Nil. itrev xs Nil = Nil @ rev xs`, by replacing `rev xs` with `Nil @ rev xs`, even though this conjecture is a non-theorem. Figure 4 illustrates the sequential application of generalization in the previous paragraph and goal oriented conjecturing described in this paragraph.

² Note that `Nil` is a universally quantified variable here.

Clean and Return. Most produced conjectures do not even type check. This step removes them as well as duplicates before passing the results to the following sub-strategy (Then [Fastforce, Quickcheck, DInd] in the example).

3 Conclusion

We presented an automatic conjecturing tool PGT and its integration into PSL. Currently, PGT tries to generate conjectures using previously derived simplification rules as hints. We plan to include more heuristics to prioritize conjectures before passing them to subsequent strategies.

Most conjecturing tools for Isabelle, such as *IsaCoSy* [6] and *Hipster* [7], are based on the bottom-up approach called *theory exploration* [2]. The drawback is that they tend to produce uninteresting conjectures. In the case of *IsaCoSy* the user is tasked with pruning these by hand. *Hipster* uses the difficulty of a conjecture's proof to determine or measure its usefulness. Contrary to their approach, PGT produces conjectures by mutating original goals. Even though PGT also produces unusable conjectures internally, the integration with PSL's search framework ensures that PGT only presents conjectures that are indeed useful in proving the original goal. Unlike *Hipster*, which is based on a Haskell code base, PGT and PSL are an Isabelle theory file, which can easily be imported to any Isabelle theory. Finally, unlike *Hipster*, PGT is not limited to equational conjectures.

Gauthier and Kaliszyk described conjecturing across proof corpora [4]. While PGT creates conjectures by mutating the original goal, Gauthier *et al.* produced conjectures by using statistical analogies extracted from large formal libraries [5].

References

1. Blanchette, J.C.: Nitpick: a counterexample generator for Isabelle/HOL based on the relational model finder Kodkod. In: LPAR-17, pp. 20–25 (2010)
2. Buchberger, B.: Theory exploration with theorem4 (2000)
3. Bulwahn, L.: The new quickcheck for Isabelle. In: Hawblitzel, C., Miller, D. (eds.) CPP 2012. LNCS, vol. 7679, pp. 92–108. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35308-6_10
4. Gauthier, T., Kaliszyk, C.: Sharing HOL4 and HOL light proof knowledge. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) LPAR 2015. LNCS, vol. 9450, pp. 372–386. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48899-7_26
5. Gauthier, T., Kaliszyk, C., Urban, J.: Initial experiments with statistical conjecturing over large formal corpora, pp. 219–228 (2016). <http://ceur-ws.org/Vol-1785/W23.pdf>
6. Johansson, M., Dixon, L., Bundy, A.: Conjecture synthesis for inductive theories. *J. Autom. Reason.* **47**(3), 251–289 (2011)

7. Johansson, M., Rosén, D., Smallbone, N., Claessen, K.: Hipster: integrating theory exploration in a proof assistant. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) CICM 2014. LNCS (LNAI), vol. 8543, pp. 108–122. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08434-3_9
8. Nagashima, Y., Kumar, R.: A proof strategy language and proof script generation for Isabelle/HOL. In: de Moura, L. (ed.) CADE 2017. LNCS (LNAI), vol. 10395, pp. 528–545. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63046-5_32
9. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002). <https://doi.org/10.1007/3-540-45949-9>



Knowledge Amalgamation for Computational Science and Engineering

Theresa Pollinger^(✉), Michael Kohlhase, and Harald Köstler

Computer Science, FAU Erlangen-Nürnberg, Erlangen, Germany
theresa.pollinger@fau.de

Abstract. This paper addresses a *knowledge gap* that is commonly encountered in computational science and engineering: To set up a simulation, we need to combine domain knowledge (usually in terms of physical principles), model knowledge (e.g. about suitable partial differential equations) with simulation (i.e. numerics/computing) knowledge. In current practice, this is resolved by intense collaboration between experts, which incurs non-trivial translation and communication overheads. We propose an alternate solution, based on *mathematical knowledge management (MKM)* techniques, specifically *theory graphs* and *active documents*: Given a theory graph representation of the domain, model, and background mathematics, we can derive a targeted knowledge acquisition dialogue that supports the formalization of domain knowledge, combines it with simulation knowledge and – in the end – drives a simulation run – a process we call MOSIS (“Models-to-Simulations Interface System”). We present the MOSIS prototype that implements this process based on a custom Jupyter kernel for the user interface and the theory-graph-based MMT knowledge management system as an MKM backend.

1 Introduction and Motivation

Computational science and engineering (CSE) deals with the development and application of computational models and simulations, often coupled with high-performance computing, to solve complex physical problems arising in engineering analysis and design (computational engineering) as well as natural phenomena (computational science). CSE has been described as the “third mode of discovery” (next to theory and experimentation). Computer simulation provides the capability to enter fields that are either inaccessible to traditional experimentation or where carrying out traditional empirical inquiries is prohibitively expensive.

However, CSE as an interdisciplinary field requires a mixture of three fields of expertise, a skillset that is difficult to acquire and for which university programs are only now being established [Rü+16]. Thus at the heart of CSE resides a *knowledge management problem* where

1. **domain knowledge** – information and intuition about real-world processes
– has to be combined with

2. **model knowledge** – i.e. how to express and describe the underlying relationships and processes in the domain with mathematical constructs, e.g. partial differential equations, and
3. **simulations knowledge** on how to get accurate and efficient numerical approximations.

The disciplinary boundaries in CSE – see the clover leaf in Fig. 1 for an illustration – do not fully align with these knowledge categories. Indeed, for many problems, CSE practitioners¹ will act as interpreters for application domain experts who usually know the problem to be simulated well and are mathematically literate. They coordinate with the domain experts on and via the models and provide the simulations expertise needed for the particular application.

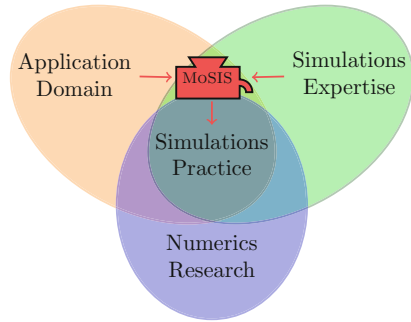


Fig. 1. Disciplines in simulations practice

We address the mathematical knowledge management (MKM) problem involved with “Mathematical Modeling and Simulation” (MMS) – a synonym for CSE that puts more emphasis on the modeling part. Following the MaMoReD (Math Models as Research Data) approach [KKMT17], we meta-model CSE/MMS knowledge using logical constructions such as theory graphs and CSE/MMS practices like knowledge application as actions (e.g. pushout constructions).

Contribution. In this paper, we show how existing MaMoReD theory graphs (see Sect. 2.1) can be utilized to solve the knowledge amalgamation problem inherent in the collaboration of domain experts and computational engineers. We discuss how theory graph structure can be exploited to automate the application knowledge acquisition processes (“Models-to-Simulations Interface System”) to drive simulation tools directly – which is why MoSIS is situated at exactly the intersection of Application and Simulations Expertise in Fig. 1. We distinguish persistent background knowledge that can be collected and curated in knowledge bases by the wider CSE community from ephemeral, application-specific knowledge that does not transfer to other situations.

To show the feasibility of the MoSIS approach, we implemented MoSIS 1.0, a simple interactive interface to the domain-specific language ExaSlang for the specification of PDEs and stencil-based algorithms. MoSIS 1.0 allows the user to generate both a flexiformal representation of their PDE model as well as its numerical solution. Furthermore, we explore other possible applications of knowledge management tools in mathematical modeling.

¹ For this paper we disregard numerics research, which is advancing the available methods, as this largely is an off-line process that is motivated by concrete applications, but not invoked on a per-problem basis.

Overview. This paper is a refined and condensed version of [Pol17], to which we refer for details and code. Section 2 sets up the running example, introduces the MaMoReD paradigm of “Mathematical Models as Research Data”, and discusses the knowledge gap encountered with the solver ExaStencils. Section 3 presents a theory graph for the knowledge of our running example discusses the MOSIS prototype. Section 4 wraps up our findings and provides an outlook.

2 Meta-Modeling and Simulation

2.1 The MaMoReD Approach

We follow the MaMoReD approach – “Mathematical Models as Research Data” [KKMT17, Kop+18] – of explicitly representing mathematical models to enable computer support of CSE practices. Concretely, the model, all its assumptions, and the mathematical background in whose terms the model is defined, are represented as a flexiformal theory graph (see Sect. 2.3). This can serve as a flexiformal documentation of the ideas and maths used in a project and can also be processed and compared computationally. However, it is fair to ask whether the benefits really outweigh the costs of creating the theory graph representation in practice. We work towards an affirmative answer by showing that MaMoReD theory graphs support added-value services for CSE: bridging the gap between problems and simulations.

To fortify our intuition, consider the following problem, which we will use as a running example throughout the paper:

Running Example. (One-Dimensional Heat Conduction Problem) Jen, an engineer, would like to simulate the heat conduction through the walls of her house. Jen knows basic physics, and in particular that the heat distribution throughout a heated wall with constant surrounding temperatures is described by Poisson’s equation when simplified to the static case. In the illustration, Fig. 2, we see heating pipes going through the wall – which is made from materials of varying thermal conductivity k_1, k_2 – as well as the warm air on its side a , and cold on b . Jen has not worked with (thermal) simulations so far, but is aware of the pitfalls involved. Luckily she has a friend James who is a computational engineer. They discuss the model (see Sect. 2.2), James inquires about the specific parameters, and eventually uses them to set up an ExaStencils script, runs the simulation, and together they interpret the results.

The problem can be extended to include time or coupled systems later on, but Jen restricts herself to the static problem for the moment.

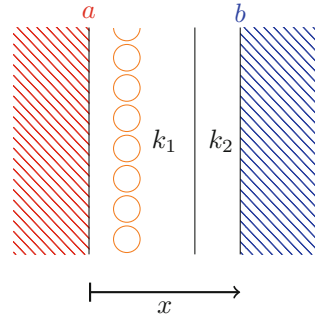


Fig. 2. Wall schematics

2.2 Poisson’s Equation

Poisson’s equation occurs in nature e.g. in electrostatics or in static heat conduction problems. Usually defining an unknown scalar function $u : \Omega \mapsto \mathbb{R}$ and an integrable source term $f : \Omega \mapsto \mathbb{R}$ on the domain $\Omega \subset \mathbb{R}^n$, it can be represented as

$$-\Delta u = f \text{ in } \Omega, \tag{1}$$

where the second-order differential operator Δ , the **Laplace operator**, is defined as the sum of the second order partial derivatives

$$\Delta = \partial_{x_1x_1} + \partial_{x_2x_2} + \dots + \partial_{x_dx_d} \tag{2}$$

in d dimensions. As we will visualize in the next paragraph, u is the unknown temperature in our running example, and f is given as the heat transported into the wall divided by the thermal conductivity.

To allow the Poisson equation to be a (uniquely) solvable **boundary value problem**, we need to add **boundary conditions** imposed on u . For example, u may need to be equal to some other function b

$$u = b \text{ in } \partial\Omega, \tag{3}$$

as a **Dirichlet boundary condition** on all of the boundary $\partial\Omega$. We can then uniquely determine u as an element of the Sobolev space $H_b^1(\Omega)$.

2.3 Theory Graphs

Theory graphs [RK13] are a conceptual format for representing mathematical knowledge in a structured, modular form. They consist of **theories**, small units of knowledge, and **morphisms** – truth-preserving mappings – between them.

The most important theory morphisms for this paper are the **inclusion** (\hookrightarrow) and **view** (\rightsquigarrow) relations. **Inclusions** are equivalent to unnamed imports in many programming languages, allowing to re-use symbols exactly as they were defined. **Views**, however, map symbols to apply different concepts of reasoning to the given situation, which in programming would be equivalent to Python-style duck typing, a common example being the concept of “example” itself.

For instance, in Fig. 3 the **Wall cross-section over time** can be viewed as a **Spatial Domain**, as all necessary symbols are mapped in the view e, effectively making **Wall cross-section** an example of a **Spatial Domain**. Subsequently, every item of knowledge that can be generally formulated in terms of the one can then be applied to the other in particular, for example the **Differential Operator** Δ . This mode of knowledge generation is called a **pushout** and is ubiquitous in mathematical reasoning. In Fig. 3, we see the pushout for **Differential operators on wall cross-section** denoted as a right angle with dot. Pushouts are a convenient construction whenever they occur, as they can be

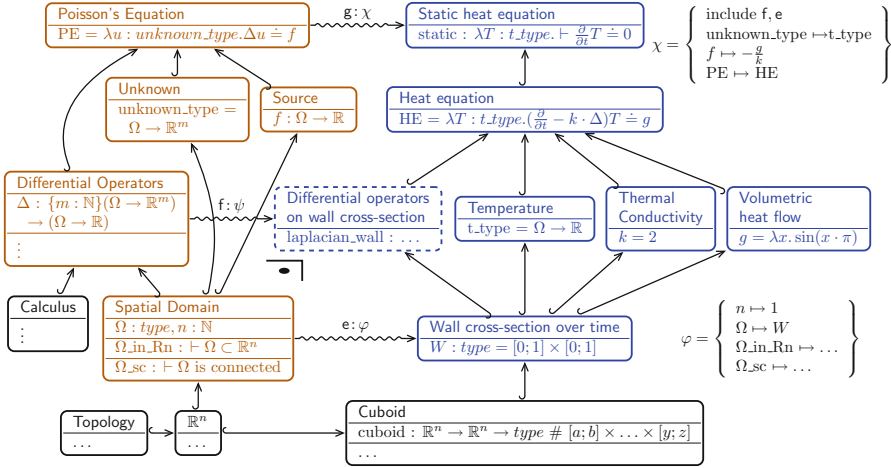


Fig. 3. A theory graph example: the static heat equation (for Poisson’s equation)

automatically generated [CMR17]. This allows us to walk up the graph further, such that finally a view g can be applied to see the **Static heat equation** as an instance of **Poisson’s Equation**, which includes the views established earlier on.

To make theory graphs computationally usable, we express them in the MMT language (Meta Meta Toolset) [MMTa]. Concretely we base our formalizations on the **Math-in-the-Middle** (MitM) foundation, a feature-rich higher-order logic with dependent function and record types as well as predicate subtypes and general subtyping. This choice of primitives is geared towards ease of representation of mathematical knowledge without losing structure.

2.4 MoSIS: Creating ExaSlang Layer 0

In the ExaStencils project [EXA], the external, multi-layered domain-specific language (DSL) ExaSlang [Len+14, Kös+17] is developed in order to support automatic code generation of scalable and efficient numerical PDE solvers. The code generator is implemented in Scala and outputs e.g. parallel C++ or CUDA code [KK16]. Algorithms that can be expressed as stencils include most finite difference methods (FDM), which can be used to numerically solve e.g. Poisson’s

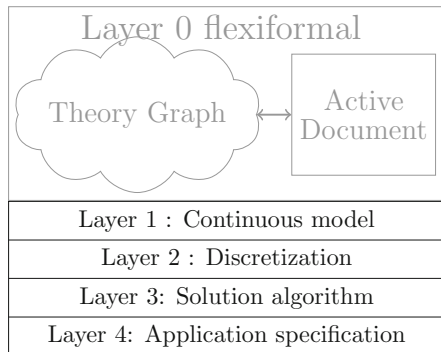


Fig. 4. MoSIS as ExaSlang Layer 0

equation on a structured grid – computational grids are restricted to structured ones that lead to highly performant stencil codes.

ExaSlang is built from different layers, cf. Fig. 4, which represent a transformation starting from an abstract continuous mathematical model (in Layer 1) down to the specifics of the application to be generated (in Layer 4). Especially Layer 1 is aimed at providing an intuitive interface for the mathematically versed user.

But there are two obstacles for ExaSlang to be an interface for MMS practitioners. Firstly, there is no interactive feedback when entering the PDE model; problems are only encountered once everything is set up and the user tries to compile and run the configuration.

Secondly, there is no input checking to ensure that the model is consistent in itself and neither under- nor over-specified as to allow for a unique solution of the PDE in the weak sense – or whether the (non-)existence of a unique solution may even be provable. This kind of consideration is part of the typical CSE/math knowledge and is not easily represented on a low level of abstraction, such as C++ code.

The underlying problem in both cases is that the transformation process from – explicit or tacit – knowledge in the experts to program code is not structure-preserving and not easily reversible; a problem that the language hierarchy in ExaSlang is designed to mitigate. But practical experience shows that (at least) another stepping stone is needed.

3 MoSIS: Combining MaMoReD and ExaStencils

We propose a “Layer 0” for ExaSlang that uses an **active document** [Koh+11] for user interaction. It can use the **theory graph** containing background knowledge to guide the user through the model description process. The resulting program can then translate the abstract model to ExaSlang Layer 1 code, which is still human-readable and can be translated to highly performant solver code via the ExaStencils tool chain. As a “by-product”, a high-level representation of the mathematical model under consideration is generated.

In the following, we will call this idea MoSIS, the “Models-to-Simulations Interface System”.

The implementation we report on in this paper uses a MMT-based **theory graph** and the **Jupyter notebook** [JN] as basic active document format. Building on the MaMoReD approach, we meta-model the process of establishing a mathematical model as an actual dialog carried out between a domain expert and a simulations expert. The main prerequisites for this are as follows.

3.1 A Theory Graph for PDE Knowledge

To evaluate MoSIS in our running example we MMT-formalized a simple meta-model of PDEs into a theory graph, cf. Fig. 5.

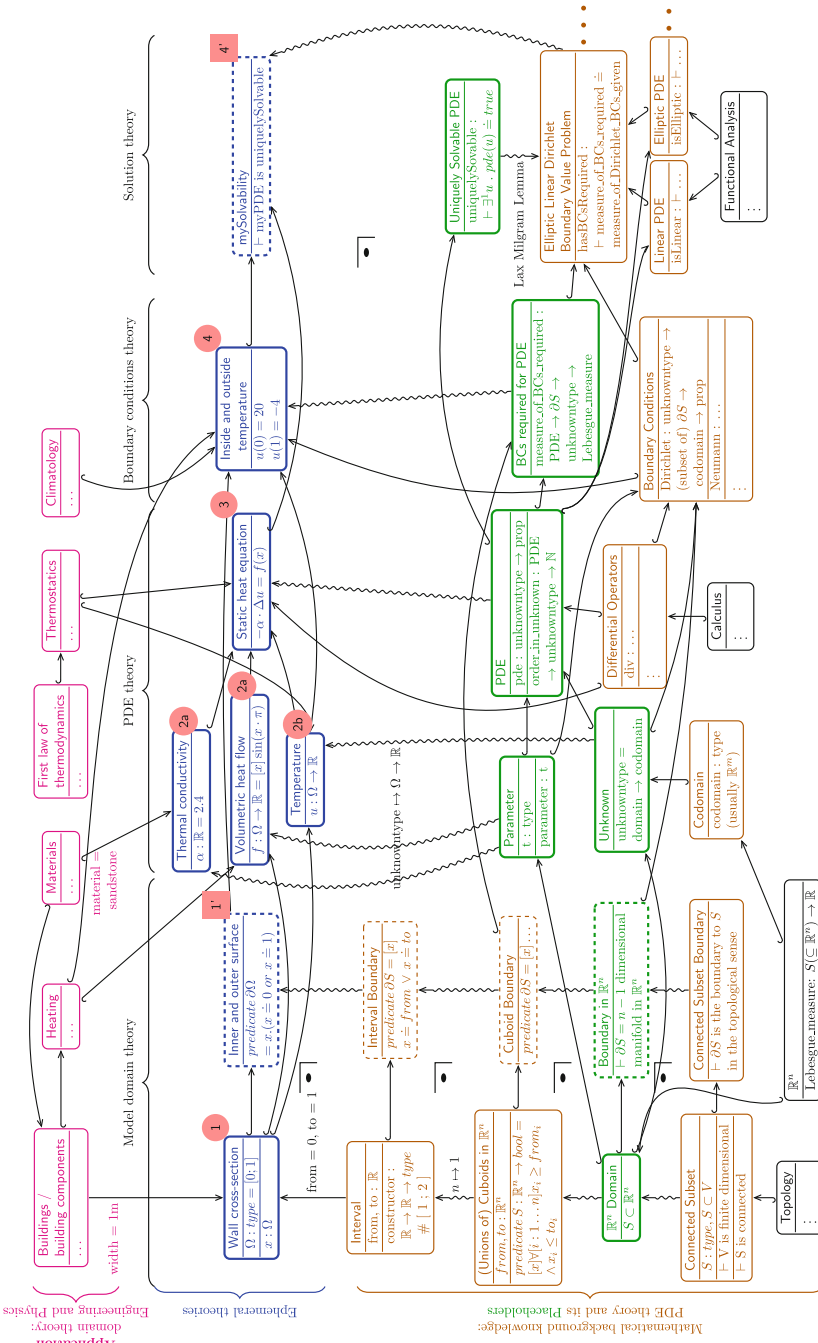


Fig. 5. Theory graph for PDEs, applied to 1-D heat conduction problem: the ephemeral (blue) subgraph is forged between the background and application knowledge and provides an outline for the interview (Color figure online)

At the base we see the more general mathematical concepts, such as the Euclidean space \mathbb{R}^n , arithmetics, calculus, etc. The specifics of PDEs come above that, answering questions like “*What do we expect of a domain?*” and “*What are the types of unknowns and the PDE itself?*”. The lower right corner contains theories about combinations of properties that make the PDE solvable², here only showing the parts needed to understand Poisson’s equation.

The knowledge encoded in the lower part of the graph is background knowledge that can be used to generally describe a PDE model and acts as an immutable common ground for multiple modeling/simulation tasks. For one particular such task (e.g. Jen’s Wall), we need to generate application-specific theories, which we call **ephemeral**, since they can be discarded after the simulation. Ephemeral theories are not universally true – and as such need not become part of the **persistent** (i.e. non-ephemeral) knowledge base – but they are true for Jen’s model.

Shown as a “sandwiched” layer in the theory graph, the ephemeral theories are always connected to their background counterpart by way of views. The concrete ephemeral subgraph in Fig. 5 is chosen for our running example: the static temperature distribution throughout the walls of a house, with fixed inside and outside temperatures. This can be mapped (also in more dimensions) to be an **elliptic linear Dirichlet boundary value problem**, and is therefore provably uniquely solvable in the weak sense.

The domain theory layer above the ephemeral theories is persistent, but not part of the meta-model; we assume this to be Jen’s knowledge about engineering and physics, based on which she would like to set up the simulation.

Figure 5 only shows the part of the MitM Ontology [Mitb] that is relevant to our – didactically chosen and thus relatively simple – running example. MitM itself is under constant development and covers other modeling case studies as well, e.g. the van Roosbroeck models discussed in [KKMT17, Kop+18]. It is our experience that the general topology of the theory graph – which is the only part that is actually used in MOSIS – does not fundamentally change for more complex examples, e.g. in higher dimensions.

Finally, note that the possible notation “ ∂ ” to denote the boundary is the same that is often used to describe the partial derivative. In practice, this only rarely causes confusion; which one is meant usually becomes clear from both the context and the type of object that the operator is applied to. The same is true for the theory graph representation here. In addition, a production MOSIS interface should allow for reformulation of notations – as long as no ambiguities are introduced – in order to support various mathematical traditions (e.g. f'_x , f_x , $\partial_x f$, $D_x f$, $D_1 f$, $\frac{\partial}{\partial x} f$, or $\frac{\partial f}{\partial x}$ as notations for the partial derivative used in different communities).

² A lot of mathematical insight about weak solutions is represented by the view labeled “Lax-Milgram Lemma”.

3.2 Automating Model Knowledge Amalgamation

The knowledge gap we discussed in the introduction is a tangible one: People – that are competent in their own domain – set up numerical tool kits themselves by diving into the technical documentation, just to end up with the wrong results (possibly without noticing and understanding why).

Luckily, our engineer Jen knows about the pitfalls and decides to talk to her friend James, a computational engineer. James asks her about the property she wants to determine – the temperature curve in the wall – and the situation at hand: concretely he needs the *(i)* coordinates of the wall cross-section, *(ii)* materials and their thermal properties, *(iii)* the layout of heat sources in the wall, *(iv)* physics of the wall (here given by the static heat equation), *(v)* boundary conditions (inside and outside temperatures).

In the course of the discussion – we think of this as the **knowledge acquisition process** to be solved by James – Jen writes down some assignments and formulas. Only when everything is defined, and the meaning is perfectly clear and plausible to both of them, James tells Jen that he thinks the PDE problem is solvable – in this case even uniquely – and starts a suitable simulation (which may further be influenced by Jen’s requirement on accuracy and time) and proudly presents Jen with the results: a nice visualization of the temperature distribution. But not every domain specialist has a friend or colleague who is a computational engineer; so automation of knowledge acquisition – and more generally of knowledge amalgamation, like the interaction between Jen and James – is desirable.

Fortunately, the sequence of interactions necessary to be able to fully specify the simulation can be read off the theory graph in Fig. 5. Essentially, the shape of the green part is a template to the (blue) subgraph of ephemeral theories that forces the interview: We would require that at least one item is defined that can be understood as a **Domain**. We can allow users to use all constants defined in theories that can themselves be viewed as a **Domain**, such as the **Interval** constructor. Establishing a (transitive) view between **Domain** and **Wall cross-section** as a 1st user input immediately returns knowledge about the domain boundary, denoted by **I**.

Next, we would like to know about (optional) **Parameters** and at least one **Unknown**. Based on the dependencies given by the include structure, we notice that the order between the inputs 2a and 2b is arbitrary. What is peculiar at this point is that only the type but no concrete definition must be given for the **Unknown**, as otherwise it would in fact be known!

The inputs 3 and 4 define the **PDE** and **Boundary Conditions** as the central part of the model specification. If we now push out³ 3 and 4 with theories **PDE** and **Elliptic Linear Dirichlet Boundary Value Problems**, we get the output theory **mySolvability**, which states that a solution $u : \Omega \rightarrow \mathbb{R}$ can be computed.

³ The category of MMT theories and morphisms has co-limits – and thus pushouts, which can be calculated canonically [CMR17] in the MMT system.

The theory graph in Fig. 5 stops at “mathematical solvability theory”, but could be extended to include numerical solvability by discretization on a grid and even solvability in particular simulation methods. Such extensions would greatly enhance practical coverage, but are beyond the scope of this paper.

Note that the ephemeral theories (in blue in Fig. 5) necessarily combine information about the background mathematical (in orange) with knowledge of the physics of the problem (theories at the top in magenta). For specifics of this combination we refer the reader to [KKMT17, Kop+18], where we have elaborated on this in the case of the “van Roosbroeck Model” from quantum electronics. Here we focus on the application of such models to a specific situation, which we had previously only touched upon.

Note furthermore that we have abbreviated the ephemeral theories and their provenance from physics; we give their constants concrete values by using full MMT declarations of the form by $c : \tau = \delta$, where c is the constant name, τ its type, and δ its definition. Actually, in the pure MaMoReD approach we would have divided them into a physical background theory with “undefined” constant $c : \tau$ and only instantiating c to δ in the corresponding ephemeral theory. In Fig. 5 we have not elaborated the physics layer, because we have a different – but related – way of automating, which we discuss now.

The green theories (thick border) in the persistent part of the graph in Fig. 5, e.g. **Boundary in \mathbb{R}^n** and **PDE** are exactly the ones we want to map ephemerally in order to get the simulation information in the steps 1 to 4. We call them **placeholder** theories. The placeholder theories can be determined by looking for those constants in the persistent graph that do not have a definition (yet).

The order in which the placeholders needs to be filled in can be obtained by the persistent theory morphisms. Consequently, the ephemeral theories can be generated as fill-ins to the placeholder “form”, carrying the same mirrored theory morphisms, and the exact same outer dependencies. This instantiation is the heart of the knowledge acquisition process.

Now one might say that it would be possible to just generate all ephemeral theories in the beginning and have all the constants assigned by the user. This is true if we generate the dialogue from a known model template as given in the **application domain theory**. The notable difference here is that we do not know how often Jen is going to fill certain parts into the generalized “model”. For instance, she may define arbitrarily many parameters – **Thermal conductivity** and **Volumetric heat flow** in our running example – several unknowns and exactly as many determining equations, here one, but we still need only one solvability theory for the whole problem.

A slightly modified approach to generate interviews is offering a keyword for each placeholder. For instance, Jen might want to define a new constant by typing `parameter my_favourite_room_temperature = 25` at any given time. The dependencies in the theory graph should then make sure that this happens only when appropriate, i. e. after a domain for x was defined. This feature would correspond to the user’s habit of introducing parameter functions e.g. to keep equations more readable, or to extend the model to coupled physics, e.g. if Jen wanted to find out how much the wall will expand when heated. To this end,

there is a lot that can be done with simplifications such as in-situ computations [ODK17] that simplify the model in real time, and the visualization of these changes. Adding keywords to the sequential interview effectively mimics mathematical model amalgamation as a mixed-initiative dialogue.

3.3 Implementation: Realizing MoSIS via Jupyter and Mmt

Building on the ideas introduced in the last section, we have implemented MoSIS 1.0 in Jupyter [Jup]. This is an open-source web application that plays **Jupyter notebooks**: interactive documents that contain live code: equations, visualizations and narrative text. The code is executed in a backend kernel process, here running the MMT system. In contrast to normal Jupyter notebooks that use a fixed sequence of (documented) instructions in a read-eval-print-loop (REPL), MoSIS implements an ephemeral dialogue using a simple state machine with states for each part (1-4) of the theory graph to be set up. MoSIS uses a MMT kernel and communicates through the HTTP API of the MMT server. We have extended the MMT server to support the creation (and if needed the storage) of ephemeral theories (see [MMTb]).

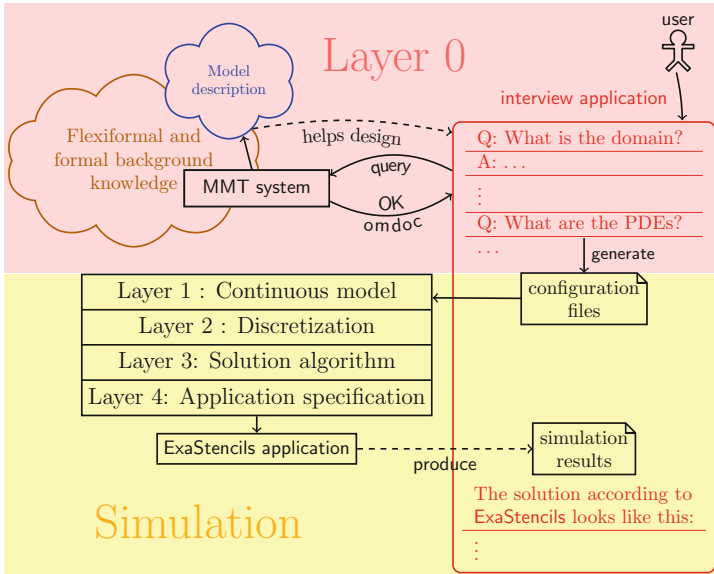


Fig. 6. MoSIS system architecture

Figure 6 shows the MoSIS architecture. The left side is a concretization of the “Layer-0-design” from Fig. 4, where the MMT system takes the place of the theory graph interface. The right hand side shows the interview – a Jupyter notebook – as the active document and how it interacts with the kernel level. In particular, the user only sees the notebook; answers the knowledge acquisition questions presented by MoSIS, until MoSIS can generate a ExaStencils

configuration file to be shipped to ExaStencils, which transforms it into efficient code through the ExaSlang layers, computes the results and visualizations, that MoSIS in turn incorporates into the notebook. Note that the user (in our example the domain expert Jen) can run simulations without having to interact with the simulation system and learn its peculiarities (or consulting James).

On the front end, MoSIS 1.0 inherits the communication and representation capabilities from the Jupyter web application and notebook format. Questions and state information required by MoSIS are presented as Markdown cells. The user’s replies can be written as “maths” encoded in MMT surface syntax employing MMT notations that were defined in the background knowledge theory graph beforehand. Following scientific practice, L^AT_EX representation can be used for non-standard Unicode characters. A side benefit of passing user inputs through MMT is that these are type-checked by the system. Already in our simple example this eliminates a considerable source of errors before they ever reach ExaStencils.

Hello, Jen! I am MoSIS 1.0, your partial differential equations and simulations expert. Let's set up a model and simulation together.

To get explanations, enter `explain <optional keyword>`. To see a recap of what we know so far, enter `recap <optional keyword>`. To interactively visualize the current theory graph, enter `tgview` or `tgview mpd`. Otherwise, you can always answer with LaTeX-type input.

Modeling

How many dimensions does your model have?

I am just assuming it's 1, since that is all we can currently handle.

What is the domain in your model? Ω : type | = [?;?], e.g. `\Omega = [0.0;1.0]`

In [1]: `\Omega = [0.0;1.0]`

we will just assume that the variable is called x for now.

Which variable(s) are you looking for? / What are the unknowns in your model? $u : \Omega \rightarrow ??$, e.g., $u : \Omega \rightarrow \mathbb{R}$?

In [2]: `u : \Omega \rightarrow \mathbb{R}`

Ok, $u : \Omega \rightarrow \mathbb{R}$

Would you like to name additional parameters like constants or functions (that are independent of your unknowns)? $c : \mathbb{R} = ?$ or $f : \Omega \rightarrow \mathbb{R} = ?$

In [3]: `f = sin(x)`

Ok, $f = [x : \Omega] \sin(x)$

Fig. 7. Beginning of a dialogue in MoSIS

For the translation into ExaSlang Layer 1 code, we use the views into the ephemeral theories, e.g. from **Domain** to **Wall cross-section** – theory graph morphisms can be composed and transform into Domain syntax via MMT notation definitions. This is also where a central benefit of the theory graph representation comes into play: The theory graph always contains the same or more information than any specific programming language representation of the same model, and can therefore be used to translate to different changing formats once a translation is available. This is especially interesting in the context of the **Math-in-the-Middle approach** [Deh+16] of aligning mathematical open source tools through a common flexiformal mathematical reference point: the MitM Ontology [Koh+17].

MOSIS uses ExaStencils to generate the solver code for this particular problem and runs it – and the user is presented with a Bokeh [BO12] visualization of the solution. But what is more, they now also have a re-usable representation of how they got to the result, both as a Jupyter notebook (for narration) and an OMDOC file (for further computation). Figure 7 shows (the beginning of) an actual MOSIS dialogue for our running example; see [MoS] for a live demo.

In addition to the dialogue turns, the user can orient herself using special MOSIS actions: getting a recap of all the ephemeral information stored so far, undoing the last step(s) and having a theory graph or Model Pathway Diagram (MPD) [KKMT17] of the current working state displayed back through the theory graph viewer TGView [RKM17], cf. Fig. 8. Our user can discover, inspect and “debug” the structure of the model captured. Given a Model/MPD graph following [KKMT17, Kop+18], we can use the MPD view as a more intuitive user interface for inspecting the PDE.

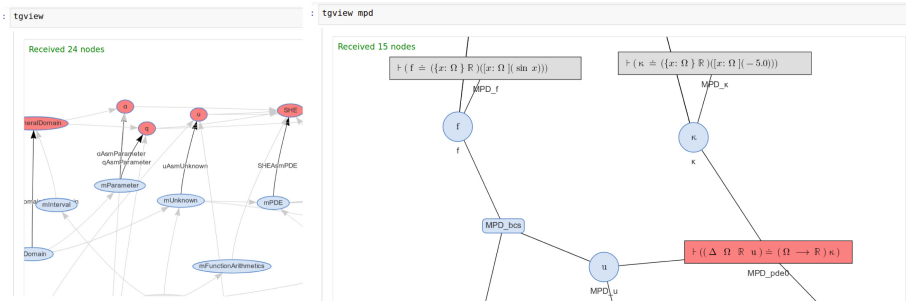


Fig. 8. The theory graph viewer TGView embedded into Jupyter

The code for MOSIS kernel can be found in [Pol]; see [Mita] for formalizations. In addition, the MOSIS 1.0 kernel is set up in a JupyterHub environment under [MoS]. JupyterHub [JHub] is a Jupyter notebook server that is accessible to many users who can independently work in their own notebooks in the browser and execute code on the server, such that no software needs to be installed locally.

4 Conclusion and Future Work

This paper addresses the knowledge gap in modeling and simulations practice: People who want to derive or work with new application models know that at the end, a PDE has to be solved, but usually no background or interest in the required computation process. But an efficient solver depends on the PDE and thus detailed system knowledge or intense discussions with simulation experts are necessary, or else errors are bound to occur.

In the *ExaStencils* project, this problem has so far been approached with the development of a dedicated domain-specific language, *ExaSlang*, cf. Sect. 2.4. The knowledge gap discussed above corresponds to the problem of specifying the model on *ExaSlang*'s most abstract layer, Layer 1.

MOSIS fills the gap by providing an architecture and components for it by combining represented knowledge with an active document that the user can interact with. We established the feasibility of MOSIS by implementing MOSIS 1.0 based on existing components: MMT for the representations of formal (and flexiformal) knowledge as a theory graph (cf. Sect. 2.3) and an interactive Jupyter kernel with a MMT backend through which it can access PDE knowledge.

We currently have only developed the theory graph to the extent necessary for our running example. For more realistic simulations we will need to extend it to n -dimensional calculus and *partial* differential equations. For that we will need to extend MMT and the MitM foundation from a purely logical system to one that can also “understand” equations as **quotations**. In our models, the names of the unknowns and variables in the equation actually matter, such that e.g. alpha-renaming these parts is not desirable (even though they are not formally bound in the model). One possible approach of dealing with this could be the internal enumeration of coordinates and variables.

To enhance the interactivity of the MOSIS front-end we are working on integrating Jupyter widgets [IPyWid15] into MOSIS, e.g. for selection inputs.

The technical terms used in the Markdown content can come with hoverable or clickable explanations as we already know them from the semantic glossary SMGloM [Gin+16]. Co-highlighting the aforementioned terms together with all the corresponding ephemeral and persistent mathematical symbols – in the best case also in the theory graph viewer and other possible tools – would greatly support the effect of visualizing what belongs together.

Acknowledgments. The authors acknowledge financial support from the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541); Project *ExaStencils* received funding within the DFG priority programme 1648 SPPEXA. We gratefully acknowledge fruitful discussions with the KWARC group, especially Dennis Müller and Florian Rabe for support with the MMT implementation, to Thomas Koprucki and Karsten Tabelow at WIAS Berlin; and to Sebastian Kuckuk (LSS chair, FAU). Last but not least, Kai Amman and Tom Wiesing have helped with the Jupyter frontend and deployment of MOSIS on JupyterHub.

References

- [BO12] bokeh: Interactive Web Plotting for Python, 26 March 2012. <https://github.com/bokeh/bokeh>. Accessed 18 Apr 2018
- [CMR17] Mossakowski, T., Rabe, F., Codescu, M.: Canonical selection of colimits. In: James, P., Roggenbach, M. (eds.) WADT 2016. LNCS, vol. 10644, pp. 170–188. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-72044-9_12
- [Deh+16] Dehaye, P.-O., Iancu, M., Kohlhase, M., Kononov, A., Lelièvre, S., Müller, D., Pfeiffer, M., Rabe, F., Thiéry, N.M., Wiesing, T.: Interoperability in the OpenDreamKit project: the math-in-the-middle approach. In: Kohlhase, M., Johansson, M., Miller, B., de Moura, L., Tompa, F. (eds.) CICM 2016. LNCS (LNAI), vol. 9791, pp. 117–131. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42547-4_9
- [EXA] Advanced Stencil-Code Engineering (ExaStencils). <http://exastencils.org>. Accessed 25 Apr 2018
- [Gin+16] Ginev, D., et al.: The SMGloM project and system: towards a terminology and ontology for mathematics. In: Greuel, G.-M., Koch, T., Paule, P., Sommese, A. (eds.) ICMS 2016. LNCS, vol. 9725, pp. 451–457. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42432-3_58
- [IPyWid15] ipywidgets: Interactive widgets for the Jupyter Notebook, 17 April 2015. <https://github.com/jupyter-widgets/ipywidgets>. Accessed 18 Apr 2018
- [JHub] JupyterHub – JupyterHub documentation. <https://jupyterhub.readthedocs.io/en/latest/>. Accessed 18 Apr 2018
- [JN] Jupyter Notebook. <http://jupyter-notebook.readthedocs.org/en/latest/notebook.html#notebook-documents>. Accessed 22 Aug 2017
- [Jup] Project Jupyter. <http://www.jupyter.org>. Accessed 22 Aug 2017
- [KK16] Kuckuk, S., Köstler, H.: Automatic generation of massively parallel codes from ExaSlang”. In: *Computation* 4.3, 4 August 2016, p. 27 (2016). <https://doi.org/10.3390/computation4030027>. Accessed 09 Oct 2017. ISSN 2079–3197
- [KKMT17] Kohlhase, M., Koprucki, T., Müller, D., Tabelow, K.: Mathematical models as research data via flexiformal theory graphs. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) CICM 2017. LNCS (LNAI), vol. 10383, pp. 224–238. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_16
- [Koh+11] Kohlhase, M., et al.: The planetary system: web 3.0 & active documents for STEM. *Procedia Comput. Sci.* 4, 598–607 (2011). <https://doi.org/10.1016/j.procs.2011.04.063>. In: Sato, M., et al. (eds.) Special Issue: Proceedings of the International Conference on Computational Science (ICCS). Finalist at the Executable Paper Grand Challenge
- [Koh+17] Kohlhase, M., De Feo, L., Müller, D., Pfeiffer, M., Rabe, F., Thiéry, N.M., Vasilyev, V., Wiesing, T.: Knowledge-based interoperability for mathematical software systems. In: Blömer, J., Kotsireas, I.S., Kutsia, T., Simos, D.E. (eds.) MACIS 2017. LNCS, vol. 10693, pp. 195–210. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-72453-9_14
- [Kop+18] Koprucki, T., et al.: Model pathway diagrams for the representation of mathematical models. *J. Opt. Quantum Electron.* 50(2), 70 (2018). <https://doi.org/10.1007/s11082-018-1321-7>

- [Kös+17] Köstler, H., et al.: A Scala prototype to generate multigrid solver implementations for different problems and target multi-core platforms. *Int. J. Comput. Sci. Eng.* **14**(2), 150–163 (2017)
- [Len+14] Lengauer, C., et al.: ExaStencils: advanced stencil-code engineering. In: Lopes, L., et al. (eds.) *Euro-Par 2014*. LNCS, vol. 8806, pp. 553–564. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14313-2_47
- [Mita] MitM/MoSIS. <https://gl.mathhub.info/MitM/MoSIS>. Accessed 18 Apr 2018
- [Mitb] MitM: The Math-in-the-Middle Ontology. <https://mathhub.info/MitM>. Accessed 05 Feb 2017
- [MMTa] MMT - Language and System for the Uniform Representation of Knowledge. Project web site. <https://uniformal.github.io/>. Accessed 30 Aug 2016
- [MMTb] UniFormal/MMT - The MMT Language and System. <https://github.com/UniFormal/MMT>. Accessed 24 Oct 10 2017
- [MoS] JupyterHub - MoSIS Demo. <http://mosis.mathhub.info>. Accessed 15 Apr 2018
- [ODK17] Kohlhase, M., Wiesing, T.: In-place computation in active documents (context/computation). Deliverable D4.9. OpenDreamKit (2017). <https://github.com/OpenDreamKit/OpenDreamKit/raw/master/WP4/D4.9/-final.pdf>
- [Pol] Pollinger, T.: *interview_kernel*. https://gl.kwarc.info/theresa_pollinger/MoSIS. Accessed 11 Apr 2018
- [Pol17] Pollinger, T.: Knowledge representation for modeling and simulation - bridging the gap between informal PDE theory and simulations practice. Master's thesis. Informatik, FAU Erlangen-Nürnberg (2017). <https://gl.kwarc.info/supervision/MSc-archive/blob/master/2017/tpollinger/thesis.pdf>
- [RK13] Rabe, F., Kohlhase, M.: A scalable module system. *Inf. Comput.* **230**, 1–54 (2013). <http://kwarc.info/frabe/Research/mmt.pdf>
- [RKM17] Rupperecht, M., Kohlhase, M., Müller, D.: A flexible, interactive theory-graph viewer. In: Kohlhase, A., Pollanen, M. (eds.) *MathUI 2017: The 12th Workshop on Mathematical User Interfaces* (2017). <http://kwarc.info/kohlhase/papers/mathui17-tgview.pdf>
- [Rü+16] Rude, U., et al.: Research and education in computational science and engineering. In: [arXiv:1610.02608](https://arxiv.org/abs/1610.02608) [cs, math, stat], 8 October 2016. [arXiv:1610.02608](https://arxiv.org/abs/1610.02608). <http://arxiv.org/abs/1610.02608>. Accessed 23 Feb 2018



Validating Mathematical Theorems and Algorithms with RISCAL

Wolfgang Schreiner^(✉) 

Research Institute for Symbolic Computation (RISC), Johannes Kepler University,
Linz, Austria

Wolfgang.Schreiner@risc.jku.at

<https://www.risc.jku.at>

Abstract. RISCAL is a language for describing mathematical algorithms and formally specifying their behavior with respect to user-defined theories in first-order logic. This language is based on a type system that constrains the size of all types by formal parameters; thus a RISCAL specification denotes an infinite class of models of which every instance has finite size. This allows the RISCAL software to fully automatically check in small instances the validity of theorems and the correctness of algorithms. Our goal is to quickly detect errors respectively inadequacies in the formalization by falsification in small model instances before attempting actual correctness proofs for the whole model class.

Keywords: Formal specification · Falsification · Model checking

1 Introduction

The application of formal methods in computer science and computer mathematics is hampered by the fact that typically most time and effort in proving theorems (such as the verification conditions of a computer program) are wasted in vain: because of errors in the formalizations, proof attempts are often a priori doomed to fail (because the conditions do not hold) or pointless (because the conditions do not express the required intentions).

The RISC Algorithm Language (RISCAL) [3, 5] attempts to detect such errors quickly by making algorithms and theorems fully checkable. For this purpose, all (program and logical) variables are restricted to finite types whose sizes are parameterized by formal parameters. For each assignment of concrete values to these parameters, the state space in which algorithms execute and the domain in which first-order formulas are interpreted are finite; this allows to check the correctness of algorithms and the validity of formulas. In RISCAL algorithms may be formulated in a very abstract way including non-deterministic constructions (choices of values satisfying certain conditions). The RISCAL software implements the denotational semantics of programs and formulas in order

Supported by the Johannes Kepler University, Linz Institute of Technology (LIT), project LOGTECHEDU, and by the OEAD WTZ project SK 14/2018 SemTech.

to perform the checks; in particular, formulas are proved/disproved by “brute-force” evaluation to their truth values. We may thus quickly validate/falsify the correctness of theorems and algorithms in small model instances before turning to their proof-based verification for the full class of models. While [5] gives an extensive overview on (an earlier version) of RISCAL, this paper focuses on its practical use for the purpose of computer mathematics.

Various other modeling languages support checking respectively counterexample generation [2], differing from RISCAL mainly in the application domain of the language and/or the exhaustiveness of the analysis. For instance, the specification language of Alloy is based on a relational logic designed for modeling the states of abstract systems and the executions of such systems; given bounds for state sizes and execution lengths, the Alloy Analyzer finds all executions leading to states satisfying/violating given properties. On the other hand, the counterexample generator Nitpick [1] for the proof assistant Isabelle/HOL supports classical predicate logic but may (due to the presence of unbounded quantifiers) fail to find counterexamples; in an “unsound mode” quantifiers are artificially bounded, but then invalid counterexamples may be reported.

While Alloy and Nitpick are based on SAT-solving techniques, RISCAL’s implementation is actually closer to that of the test case generator Smallcheck [4]. This system generates for the parameters of a Haskell function in an exhaustive way argument values up to a certain bound; the results of the corresponding function applications may be checked against properties expressed in first-order logic (encoded as executable higher-order Haskell functions). However, unlike RISCAL, the value generation bound is specified by global constants rather than by the types of parameters and quantified variables such that separate mechanisms have to be used to restrict searches for counterexamples.

RISCAL differs from these approaches in that it combines a rich language that is suitable for specifying and implementing mathematical algorithms (including implicitly specified functions and non-deterministic choices) with a fully checkable semantics, all of which is integrated in a single easy to use specification and validation framework (see Fig. 1 for its GUI). RISCAL is intended primarily for educational purposes [6]; it is open source and freely available [3] with extensive documentation and examples.

2 Checking Theorems and Algorithms

We demonstrate the use of RISCAL by a small example adapted from [5].

A Theory. The following specification introduces a domain `Formula` of propositional formulas with n variables in conjunctive normal form:

```
val n: ℕ;
type Literal = ℤ[-n,n] with value ≠ 0;
type Clause = Set[Literal] with ∀l∈value. ¬(-l∈value);
type Formula = Set[Clause];
```

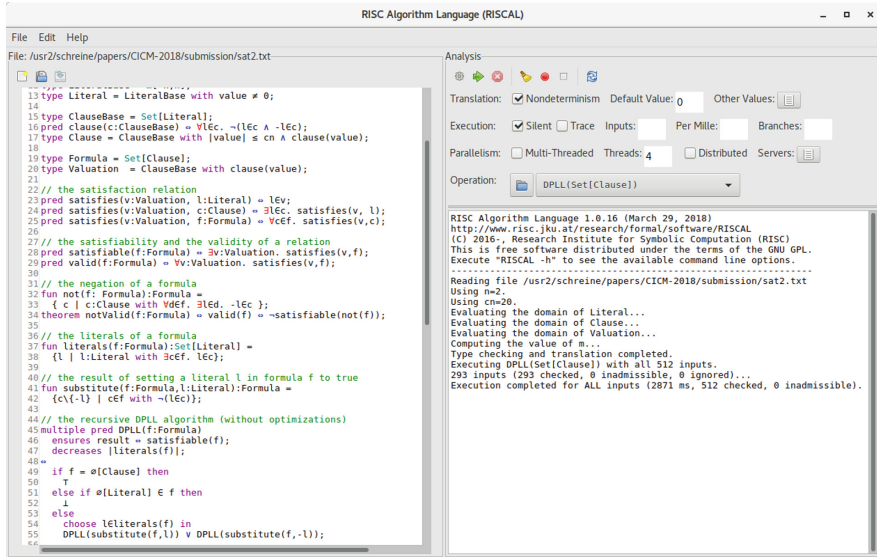


Fig. 1. The RISCAL graphical user interface

A formula is represented as a set of clauses where a clause is a set of non-conflicting literals; a literal is a non-zero integer in the interval $[-n, n]$ (in a type definition, `value` denotes the parameter of the predicate constraining the values of the type). A formula valuation has the same representation as a clause: positive literals in the clause receive the value “true” while negative literals receive the value “false”. Based on this representation, we can define a predicates `satisfies(v,f)` denoting the relation “valuation v satisfies formula f ”, `satisfiable(f)` denoting “ f is satisfiable”, and `valid(f)` denoting “ f is valid”:

```
type Valuation = Clause;
pred satisfies(v:Valuation, f:Formula) ⇔ ∀c∈f. ∃l∈c. l∈v;
pred satisfiable(f:Formula) ⇔ ∃v:Valuation. satisfies(v,f);
pred valid(f:Formula) ⇔ ∀v:Valuation. satisfies(v,f);
```

Then we introduce the negation `not(f)` of formula f and formulate a theorem that states that f is valid if and only if its negation is not satisfiable:

```
fun not(f:Formula):Formula = { c | c:Clause with ∀d∈f. ∃l∈d. -l∈c };
theorem notValid(f:Formula) ⇔ valid(f) ⇔ ¬satisfiable(not(f));
```

In RISCAL, a theorem is a predicate that shall be true for all possible arguments. We can quickly check its validity for $n = 2$ by selecting in the RISCAL user interface the operation `notValid` and pressing the “Start Execution” button:

```
Executing notValid(Set[Clause]) with all 512 inputs.
Execution completed for ALL inputs (103 ms, 512 checked, 0 inadmissible).
```

Since the number of formulas grows double-exponentially with the number of literals, we modify the definition of `Clause` (leaving all other definitions unchanged) to consider only clauses with up to cn literals:

```
val cn: ℕ;
type Clause = Set[Literal] with |value| ≤ cn ∧ ∀l∈value. ¬(¬l∈value);
```

Using $n = 3$ and $cn = 2$ and selecting the “Multi-Threaded” option in the user interface we can check the correctness of the theorem for a large number of inputs in a still quite manageable amount of time:

```
Executing notValid(Set[Clause]) with all 524288 inputs.
PARALLEL execution with 4 threads (output disabled).
...
Execution completed for ALL inputs (17445 ms, 524288 checked, 0 inadmissible).
```

However, the checking of an invalid “theorem”

```
theorem notValidDual(f:Formula) ⇔ satisfiable(f) ⇔ ¬valid(not(f));
```

produces the expected failure:

```
ERROR in execution of notValidDual({{-3},{-2},{-1}}): evaluation of
  notValidDual
at line 42 in file sat2.txt:
  theorem is not true
```

In order to demonstrate the reason of failures, formulas may be annotated such that the values of subexpressions are printed during evaluation/checking; more elaborate explanation features are planned for the future.

An Algorithm. We now consider an algorithm for determining whether a formula f is satisfiable. As a prerequisite, we introduce two operations to determine the set of literals occurring in f and to give a literal l in f the value “true”:

```
fun literals(f:Formula):Set[Literal] = { l | l:Literal with ∃c∈f. l∈c };
fun substitute(f:Formula,l:Literal):Formula = { c\{-l} | c∈f with ¬(l∈c) };
```

Now a simple recursive algorithm solving our problem can be defined as follows:

```
multiple pred DPLL(f:Formula)
  ensures result ⇔ satisfiable(f);
  decreases |literals(f)|;
⇔
  if f = ∅[Clause] then
    ⊤
  else if ∅[Literal] ∈ f then
    ⊥
  else
    choose l∈literals(f) in
      DPLL(substitute(f,l)) ∨ DPLL(substitute(f,-l));
```

This algorithm is annotated by the `ensures` clause with the postcondition that the result of the algorithm has to satisfy: it must be “true” if and only if the formula f is satisfiable. Indeed this condition will be checked in all executions and any violations will be reported. Furthermore, the `decreases` clause indicates a measure that is decreased in every recursive invocation but does not become negative: the number of literals in f . Also these constraints will be checked in all executions such that non-termination can be ruled out.

In the recursive case, the algorithm chooses an arbitrary literal l in f and calls itself recursively twice, once with setting l to “true”, once with setting l to “false”. The RISCAL language has generally a *nondeterministic* semantics: if in the user interface the option “Nondeterminism” is chosen, the algorithm will be executed with *all* possible choices of l . With $n = 2$, we thus get output:

```
Executing DPLL(Set[Clause]) with all 512 inputs.
Execution completed for ALL inputs (884 ms, 512 checked, 0 inadmissible).
```

If this option is not selected, the algorithm will be only executed for one choice, which increases for larger n the speed of the check considerably; e.g., using $n = 3$ and $cn = 2$, we get:

```
Executing DPLL(Set[Clause]) with all 524288 inputs.
PARALLEL execution with 4 threads (output disabled).
...
Execution completed for ALL inputs (196219 ms, 524288 checked, 0 inadmissible).
Not all nondeterministic branches may have been considered.
```

An algorithm may be also formulated in an imperative style as a state-based procedure, again (by the use of a `choose` command) with a non-deterministic semantics. In such procedures, loops may be annotated by invariants and termination measures; also these are checked by all executions of the algorithm. See [5] for more details on the command language for writing state-based procedures.

Validating Specifications. As shown above, we may check that the application of a RISCAL procedure p to every argument x that satisfies the procedure’s precondition $P(x)$ generates a result y that satisfies the postcondition $Q(x, y)$. However, this check is meaningless if the specification formalized by P and Q does not really express our informal intentions. As a first possibility to validate the adequacy of the formal specification, RISCAL generates from it an implicitly defined function:

```
_pSpec(x) requires P(x) = choose y with Q(x,y);
```

This function can be executed for given x by enumerating all possible values of y (in the result domain) and checking if $Q(x, y)$ holds. Its execution in non-deterministic mode thus produces for every input x constrained by $P(x)$ every value y allowed by $Q(x, y)$; the results of this execution are displayed and may be inspected to confirm the correspondence of the specification to our intentions.

Furthermore, RISCAL automatically generates from P and Q various theorems whose validity may be checked to ensure that the specification satisfies various (usually) expected properties:

- *Is precondition satisfiable?* $(\exists x. P(x))$
- *Is precondition not trivial?* $(\exists x. \neg P(x))$
- *Is postcondition always satisfiable?* $(\forall x. P(x) \Rightarrow \exists y. Q(x, y))$
- *Is postcondition always not trivial?* $(\forall x. P(x) \Rightarrow \exists y. \neg Q(x, y))$
- *Is postcondition at least sometimes not trivial?* $(\exists x. P(x) \wedge \exists y. \neg Q(x, y))$
- *Is result unique?* $(\forall x, y_1, y_2. P(x) \wedge Q(x, y_1) \wedge Q(x, y_2) \Rightarrow y_1 = y_2)$

RISCAL determines the validity of also these formulas by their evaluation.

3 Current and Further Work

For checking a procedure, a loop may be annotated with invariants whose validity is checked before and after every iteration. However, while checking may reveal that an invariant is too strong (it is violated by some loop iteration), it cannot reveal that it is too weak to carry a proof-based verification of the procedure’s correctness. Therefore, currently work is under way to generate in RISCAL also *verification conditions* whose validity (with respect to the whole model class) implies the general correctness of the procedure; these conditions can be falsified by checks (in individual instances of the class). If we are not able to falsify these conditions, our confidence in their validity is increased and we may subsequently attempt their proof-based verification (for this we later plan to integrate RISCAL with an external proof assistant).

As for its application in education, so far RISCAL has been used in a fourth year’s master course on “Formal Methods in Software Development” in order to make students familiar with the semantics and pragmatics of program specifications before exposing them to proof-based tools; this has indeed helped students to develop adequate program specifications and annotations as a basis for general verification. In the next year, we plan to use RISCAL also in a first semester’s introductory course on “Logic” to train students by small exercises in the practical use of first-order logic and in an introductory course on “Formal Modeling” to develop logic-based formal models of problem domains. Our long-term goal is to develop (with the help of students) a collection of educational resources in various areas of computer science and mathematics; we have started to cover selected topics in discrete mathematics, computer algebra, and fundamental algorithms.

References

1. Blanchette, J.C., Nipkow, T.: Nitpick: a counterexample generator for higher-order logic based on a relational model finder. In: Kaufmann, M., Paulson, L.C. (eds.) ITP 2010. LNCS, vol. 6172, pp. 131–146. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14052-5_11
2. Butler, M., et al. (eds.): Abstract State Machines, Alloy, B, TLA, VDM, and Z. LNCS, vol. 9675. Springer, Heidelberg (2016). <https://doi.org/10.1007/978-3-319-33600-8>
3. RISCAL: The RISC Algorithm Language (RISCAL), March 2017. <https://www.risc.jku.at/research/formal/software/RISCAL>

4. Runciman, C., Naylor, M., Lindblad, F.: Smallcheck and lazy smallcheck: automatic exhaustive testing for small values. In: First ACM SIGPLAN Symposium on Haskell, Haskell 2008, pp. 37–48. ACM, New York (2008). <https://doi.org/10.1145/1411286.1411292>
5. Schreiner, W.: The RISC Algorithm Language (RISCAL) – Tutorial and Reference Manual (Version 1.0). Technical report, RISC, Johannes Kepler University, Linz, Austria, March 2017. download from [3]
6. Schreiner, W., Brunhuemer, A., Fürst, C.: Teaching the formalization of mathematical theories and algorithms via the automatic checking of finite models. In: Post-Proceedings ThEdu’17, Theorem proving components for Educational software. EPTCS, vol. 267, pp. 120–139 (2018). <https://doi.org/10.4204/EPTCS.267.8>



First Experiments with Neural Translation of Informal to Formal Mathematics

Qingxiang Wang^{1,2}, Cezary Kaliszyk¹(✉) , and Josef Urban²

¹ University of Innsbruck, Innsbruck, Austria
cezary.kaliszyk@uibk.ac.at

² Czech Technical University in Prague, Prague, Czech Republic

Abstract. We report on our experiments to train deep neural networks that automatically translate informalized \LaTeX -written Mizar texts into the formal Mizar language. To the best of our knowledge, this is the first time when neural networks have been adopted in the formalization of mathematics. Using Luong et al.’s neural machine translation model (NMT), we tested our aligned informal-formal corpora against various hyperparameters and evaluated their results. Our experiments show that our best performing model configurations are able to generate correct Mizar statements on 65.73% of the inference data, with the union of all models covering 79.17%. These results indicate that formalization through artificial neural network is a promising approach for automated formalization of mathematics. We present several case studies to illustrate our results.

1 Introduction: Autoformalization

In this paper we describe our experiments with training an end-to-end translation of \LaTeX -written mathematical texts to a formal and verifiable mathematical language – in this case the Mizar language. This is the next step in our *project to automatically learn formal understanding* [11–13] of mathematics and exact sciences using large corpora of alignments between informal and formal statements. Such machine learning and statistical translation methods can additionally integrate strong semantic filtering methods such as type-checking and large-theory Automated Theorem Proving (ATP) [4, 23].

Since there are currently no large corpora that would align many pairs of human-written informal \LaTeX formulas with their corresponding formalization, we obtain the first corpus for the experiments presented here by *informalization* [12]. This is in general a process in which a formal text is turned into (more) informal one. In our previous work over Flyspeck and Mizar [11, 12] the

Q. Wang and C. Kaliszyk—Supported by ERC grant no. 714034 *SMART*.

J. Urban—Supported by the *AI4REASON* ERC Consolidator grant number 649043, and by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15.003/0000466 and the European Regional Development Fund.

main informalization method was to forget which overloaded variants and types of the mathematical symbols are used in the formal parses. Here, we additionally use a nontrivial transformation of Mizar to \LaTeX that has been developed over two decades by Bancerek [1, 3] for presenting and publishing the Mizar articles in the journal *Formalized Mathematics*.¹

Previously [11, 12], we have built and trained on the smaller aligned corpora custom translation systems based on probabilistic grammars, enhanced with semantic pruning methods such as type-checking. Here we experiment with state-of-the-art *artificial neural networks*. It has been shown recently that given enough data, neural architectures can learn to a high degree the syntactic correspondence between two languages [20]. We are interested to see to what extent the neural methods can achieve meaningful translation by training on aligned informal-formal pairs of mathematical statements. The neural machine translation (NMT) architecture that we use is Luong et al.’s implementation [15] of the sequence-to-sequence (seq2seq) model.

We will start explaining our ideas by first providing a self-contained introduction to neural translation and the seq2seq model in Sect. 2. Section 3 explains how the large corpus of aligned Mizar- \LaTeX formulas is created. Section 4 discusses preprocessing steps and application of NMT to our data, and Sect. 5 provides an exhaustive evaluation of the neural methods available in NMT. Our main result is that when trained on about 1 million aligned Mizar- \LaTeX pairs, the best method achieves perfect translation on 65.73% of about 100 thousand testing pairs. Section 7 concludes and discusses the research directions opened by this work.

2 Neural Translation

Function approximation through artificial neural network has existed in the literature since 1940s [7]. Theoretical results in late 80–90s have shown that it is possible to approximate an arbitrary measurable function by layers of compositions of linear and nonlinear mappings, with the nonlinear mappings satisfying certain mild properties [6, 10]. However, before 2010s due to limitation of computational power and lack of large training datasets, neural networks generally did not perform as well as alternative methods.

Situation changed in early 2010s when the first GPU-trained convolutional neural network outperformed all rival methods in an image classification contest [14], in which a large labeled image dataset was used as training data. Since then we have witnessed an enormous amount of successful applications of neural networks, culminating in 2016 when a professional Go player was defeated by a neural network-enabled Go-playing system [16].

Over the years many variants of neural network architectures have been invented, and easy-to-use neural frameworks have been built. We are particularly interested in the sequence-to-sequence (seq2seq) architectures [5, 20] which have achieved tremendous successes in natural language translation as well as related tasks. In particular, we have chosen Luong’s NMT framework [15] that encapsulates the Tensorflow API gracefully and the hyperparameters of the seq2seq

¹ <https://www.degruyter.com/view/j/forma>.

model are clearly exposed at command-line level. This allows us to quickly and systematically experiment with our data.

2.1 The Seq2seq Model

A seq2seq model is a network which consists of an encoder and a decoder (the left and right part in Fig. 1). During training, the encoder takes in a sentence one word at a time from the source language, and the decoder takes in the corresponding sentence from the target language. The network generates another target sentence and a loss function is computed based on the input target sentence and the generated target sentence. As each word in a sentence will be embedded into the network as a real vector, the whole network can be considered as a complicated function from a finite-dimensional real vector space to the reals. Training of the neural network amounts to conducting optimization based on this function.

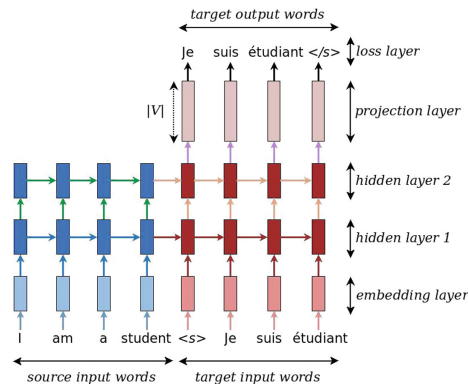


Fig. 1. Seq2seq model (adapted from Luong et al. [15])

When the training is complete, the neural network can be used to generate translations by *inferring* from (translating of) unseen source sentences. During inference, only the source sentence is provided. A target sentence is then generated word after word from the decoder by conducting greedy evaluation with the probabilistic model represented by the trained neural network (Fig. 2).

2.2 RNN and the RNN Memory Cell

The architectures of the encoder and the decoder inside the seq2seq model are similar, each of which consists of multiple layers of *recurrent neural networks* (RNNs). A typical RNN consists of one memory cell, which takes input word tokens (in vector format) and updates its parameters iteratively. An RNN cell is typically presented in literature in the *rolled-out format* (Fig. 3), though the

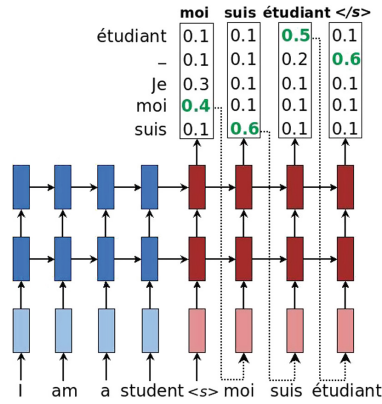


Fig. 2. Inference of seq2seq model (adapted from Luong et al. [15])

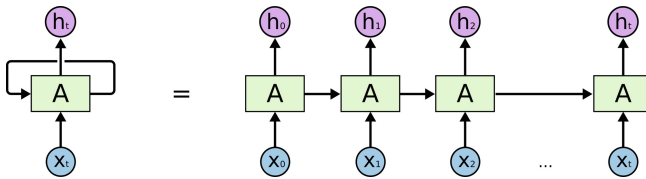


Fig. 3. RNN cell and its rolled-out format (adapted from Olah’s blog [18])

same memory cell is used and the same set of parameters are being updated during training.

Inside each memory cell there is an intertwined combination of linear and nonlinear transformations (Fig. 4). These transformations are carefully chosen to mimic the cognitive process of keeping, retaining and forgetting information.

Only differentiable functions are used to compose the memory cell, so the overall computation is also a differentiable function and gradient-based optimization can be adopted. In addition, the computation is designed in so that the derivative of the memory cell’s output with respect to its input is always close to one. This ensures that the problem of vanishing or exploding gradients is avoided when conducting differentiation using the chain rule. Several variants of memory cells exist. The most common are the *long short-term memory* (LSTM) in Fig. 4 and the *gated recurrent unit* (GRU), in Fig. 5.

2.3 Attention Mechanism

The current seq2seq model has a limitation: in the first iteration the decoder obtains all the information from the encoder, which is unnecessary as not all parts of the source sentence contribute equally to particular parts of the target sentence. The *attention* mechanism is used to overcome this limitation by adding special layers in parallel to the decoder (Fig. 6). These special layers compute

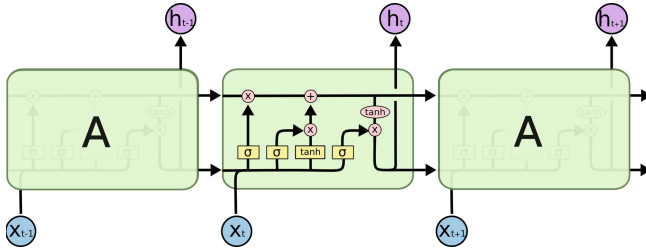


Fig. 4. Close-up look of an LSTM cell (adapted from Olah’s blog [18])

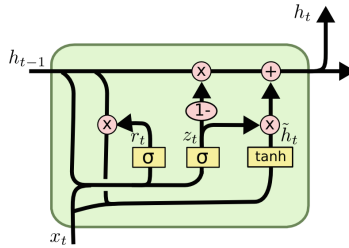


Fig. 5. Gated recurrent unit (adapted from Olah’s blog [18])

scores which can provide a weighting mechanism to let the decoder decide how much emphasis should be put on certain parts of the source sentence when translating a certain part of the target sentence. There are also several variants of the attention mechanism, depending on how the scores are computed or how the input and output are used. In our experiments, we will explore all the attention mechanisms provided by the NMT framework and evaluate their performance on the Mizar- $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ dataset.

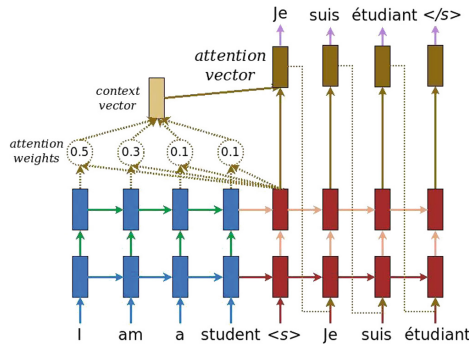


Fig. 6. Neural network with attention mechanism (adapted from Luong et al. [15])

3 The Informalized Dataset

State-of-the-art neural translation methods generally require large corpora consisting of many pairs of aligned sentences (e.g. in German and English). The lack of aligned data in our case has been a bottleneck preventing experiments with end-to-end neural translation from informal to formal mathematics. The approach that we have used so far for experimenting with non-neural translation methods is to take a large formal corpus such as Flyspeck [9] or Mizar [2] and apply various *informalization (ambiguation)* [11, 12] transformations to the formal sentences to obtain their less formal counterparts. Such transformations include e.g. forgetting which overloaded variants and types of the mathematical symbols are used in the formal parses, forgetting of explicit casting functors, bracketing, etc. These transformations result in more human-like and ambiguous sentences that (in particular in the case of Mizar) resemble the natural language style input to ITP systems, but the sentences typically do not include more complicated symbol transformations that occur naturally in \LaTeX .

There are several formalizations such as Flyspeck, the Coq proof of the Odd-Order theorem, the Mizar formalization of the Compendium of Continuous Lattices (CCL) that come with a high-level alignment of the main theorems in the corresponding (\LaTeX -written) books to the main formalized theorems. However, such mappings are so far quite sparse: e.g., there are about 500 alignments between Flyspeck and its informal book [8]. Instead, we have decided to obtain the first larger corpus of aligned \LaTeX /formal sentences again by informalization. Our requirement is that the informalization should be nontrivial, i.e., it should target a reasonably rich subset of \LaTeX and the transformations should go beyond simple symbol replacements.

The choice that we eventually made is to use the Mizar translation to \LaTeX . This translation has been developed for more than two decades by the Mizar team [21] and specifically by Bancerek [1, 3] for presenting and publishing the Mizar articles in the journal Formal Mathematics. This translation is relatively nontrivial [1]. It starts with user-defined translation patterns for different basic objects of the Mizar logic: functors, predicates, and type constructors such as adjectives, modes and structures. Quite complicated mechanisms are also used to decide on the use of brackets, the uses of singular/plural cases, regrouping of conjunctive formulas, etc. Tables 1 and 2 show examples of this translation for theorems `XBOOLE_1:1`² and `BHSP_2:3`³, together with their tokenized form used for the neural training and translation.

Since Bancerek’s technology is only able to translate Mizar formal abstracts into Latex, in order to obtain the maximum amount of data, we modified the latest experimental Mizar-to- \LaTeX XSL stylesheets that include the option to produce all the proof statements. During the translation of a Mizar article we track for every proof-internal formula its starting position (line and column) in the corresponding Mizar article, marking the formulas with these positions in

² http://grid01.ciirc.cvut.cz/~mptp/7.13.01.4.181.1147/html/xboole_1#T1.

³ http://grid01.ciirc.cvut.cz/~mptp/7.13.01.4.181.1147/html/bhsp_2#T13.

Table 1. Theorem 1 in XB00LE_1

Rendered L ^A T _E X	If $X \subseteq Y \subseteq Z$, then $X \subseteq Z$.
Mizar	$X \subseteq Y \ \& \ Y \subseteq Z$ implies $X \subseteq Z$;
Tokenized Mizar	$X \subseteq Y \ \& \ Y \subseteq Z$ implies $X \subseteq Z$;
L ^A T _E X	If $\$X \subseteq Y \subseteq Z\$,$ then $\$X \subseteq Z\$.$
Tokenized L ^A T _E X	If $\$ X \subseteq Y \subseteq Z \$,$ then $\$ X \subseteq Z \$.$

Table 2. Theorem 3 in BHSP_2

Rendered L ^A T _E X	Suppose s_8 is convergent and s_7 is convergent . Then $\lim(s_8+s_7) = \lim s_8 + \lim s_7$
Mizar	seq1 is convergent & seq2 is convergent implies $\lim(\text{seq1} + \text{seq2}) = (\lim \text{seq1}) + (\lim \text{seq2})$;
Tokenized Mizar	seq1 is convergent & seq2 is convergent implies $\lim (\text{seq1} + \text{seq2}) = (\lim \text{seq1}) + (\lim \text{seq2})$;
L ^A T _E X	Suppose $\{s_8\}$ is convergent and $\{s_7\}$ is convergent. Then $\mathop{\mathrm{lim}}\{s_8\} + \mathop{\mathrm{lim}}\{s_7\} = \mathop{\mathrm{lim}}\{s_8\} + \mathop{\mathrm{lim}}\{s_7\}$
Tokenized L ^A T _E X	Suppose $\{ s _ { 8 } \}$ is convergent and $\{ s _ { 7 } \}$ is convergent . Then $\mathop{\mathrm{lim}} \{ \text{ s } _ { 8 } \} + \mathop{\mathrm{lim}} \{ \text{ s } _ { 7 } \} = \mathop{\mathrm{lim}} \{ \text{ s } _ { 8 } \} + \mathop{\mathrm{lim}} \{ \text{ s } _ { 7 } \}$

the generated L^AT_EX file. We then extract each formula tagged with its position P from the L^AT_EX file, align it with the Mizar formulas starting at position P , and apply further data processing to them (Sect. 4.1). This results in about one million aligned pairs of L^AT_EX/Mizar sentences.

4 Applying Neural Translation to Mizar

4.1 Data Preprocessing

To adapt our data to NMT, the L^AT_EX sentences and their corresponding Mizar sentences must be properly tokenized (Tables 1 and 2). In addition, distinct word tokens from both L^AT_EX and Mizar must also be provided as vocabulary files.

In Mizar formulas, tokens can be and often are concatenated – as e.g. in $n < m$. We used each article’s symbol and identifier files produced by the Mizar accommodator and parser to separate such tokens. For L^AT_EX sentences, we decided

to consider dollar signs, brackets, parentheses, carets and underscores as separate tokens. We keep tags starting with backslash intact and leave all the font information (e.g. romanization or emphasis). Cross-referencing tags, styles for itemization as well as other typesetting information are removed.

4.2 Division of Data

Luong’s NMT model requires a small set of development data and test data in addition to training data. To conduct the full training-inference process the raw data needs to be divided into four parts. Our preprocessed data contains 1,056,478 pairs of Mizar- \LaTeX sentences. In order to achieve a 90:10 training-to-inference ratio we randomly divide our data into the following:

- 947,231 pairs of sentences of training data.
- 2,000 pairs of development data (for NMT model selection).
- 2,000 pairs of test data (for NMT model evaluation).
- 105,247 pairs of inference (testing) data.
- 7,820 and 16,793 unique word tokens generated for the vocabulary files of \LaTeX and Mizar sentences, respectively.

For our partition, there are 57,145 lines of common latex sentences in both the training set and the inference set, making up to 54.3% of the inference set. This is expected as mathematical proofs involve a lot of common basic proof steps. Therefore, in addition to correct translations, we are also interested in correct translations in the 48,102 non-overlapping sentences.

4.3 Choosing Hyperparameters

Luong’s NMT model provides around 70 configurable hyperparameters, many of which can affect the architecture of the neural network and in turn affect the training results. In our experiments, we decided to evaluate our model with respect to the following 7 hyperparameters that are the most relevant to the behavior of the seq2seq model (Table 4), while keeping other hyperparameters (those that are more auxiliary, experimental or non-recommended for change) at their default. Selected common hyperparameters are listed in Table 3.

Table 3. Common network hyperparameters across experiments

Name	Default value
Number of training steps	12,000
Learning rate	1.0 (0.001 when using Adam optimizer)
Forget bias for LSTM cell	1.0
Dropout rate	0.2
Batch size	128
Decoding type	Greedy

Table 4. Hyperparameters for seq2seq model

Name	Description	Value
Unit type	Type of the memory cell in RNN	LSTM (default) GRU Layer-norm LSTM
Attention	The attention mechanism	No Attention (default) (Normed) Bahdanau (Scaled) Luong
Nr. of layers	RNN layers in encoder and decoder	2 layers (default) 3/4/5/6 layers
Residual	Enables residual layers (to overcome exploding/vanishing gradients)	False (default) True
Optimizer	The gradient-based optimization method	SGD (default) Adam
Encoder type	Type of encoding methods for input sentences	Unidirectional (default) Bidirectional
Nr. of units	The dimension of parameters in a memory cell	128 (default) 256/512/1024/2048

5 Evaluation

The results are evaluated by four different metrics: (1) perplexity; (2) the BLEU rate of the final test data set; (3) the number and percentage of identical statements within all the 105,247 inference sentences and (4) the number and percentage of identical statements within the 48,102 non-overlapping inference sentences. Perplexity measures the difficulty of generating correct words in a sentence, and the BLEU rate gives a score on the quality of the overall translation. Details explaining perplexity and the BLEU rate can be found in [17] and [19], respectively. Due to the abundance of hyperparameters, we decided to do our experiments progressively, by first comparing a few basic hyperparameters, fixing the best choices and then comparing the other hyperparameters. The basic hyperparameters we chose are the type of memory cell and the attention mechanism.

5.1 Choosing the Best Memory Cell and Attention Mechanism

From Table 5 we can see that GRU and LSTM perform similarly and both perform better than Layer-normed LSTM. As LSTM performed slightly better than GRU we fixed our memory cell to be LSTM for further experiments.⁴

Published NMT evaluations show that the attention mechanism results in better performance in translation tasks. Our experiments confirm this fact and

⁴ Since training and inference involve randomness, the final results are not identical across trials, though our experience showed that the variation of the inference metrics are small.

also show that the Normed Bahdanu attention, Luong attention and Scaled Luong attention are better than Bahdanau attention (Table 6). Among them we picked the best-performing Scaled Luong attention as our new default and used this attention for our further experiments.

Table 5. Evaluation on type of memory cell (attention not enabled)

Parameter	Final test perplexity	Final test BLEU	Identical statements (%)	Identical no-overlap (%)
LSTM	3.06	41.1	40121 (38.12%)	6458 (13.43%)
GRU	3.39	34.7	37758 (35.88%)	5566 (11.57%)
Layer-norm LSTM	11.35	0.4	11200 (10.64%)	1 (0%)

Table 6. Evaluation on type of attention mechanism (LSTM cell)

Parameter	Final test perplexity	Final test BLEU	Identical statements (%)	Identical no-overlap (%)
No Attention	3.06	41.1	40121 (38.12%)	6458 (13.43%)
Bahdanau	3	40.9	44218 (42.01%)	8440 (17.55%)
Normed Bahdanau	1.92	63.5	60192 (57.19%)	18057 (37.54%)
Luong	1.89	64.8	60151 (57.15%)	18013 (37.45%)
Scaled Luong	2.13	65	60703 (57.68%)	18105 (37.64%)

5.2 The Effect of Optimizers, Residuals and Encodings with Respect to Layers

After fixing the memory cell and the attention mechanism, we tried the effects of the optimizer types and of the encoding mechanisms on our data with respect to the number of the RNN layers. We also experiment with enabling the residual layers. The results are shown in Table 7. We can observe that:

1. For RNN because of the vanishing gradient problem the result generally deteriorates when the number of layers becomes higher. Our experiments confirm this: the best-performing architecture has 3-layers.
2. Residuals can be used to alleviate the effect of vanishing gradients. We see from Table 7 that the results are generally better with residual layers enabled, though there are cases when residuals produce failures in training.
3. The NaN values are caused by the overflow of the optimization metric (bleu rate). For some hyperparameter combinations, it happens that the metric will get worse as training progresses, which ultimately leads to overflow and subsequent early stop of the training phase. Our experiments show that this overflow reappears with respect to multiple times of trainings.

4. It is interesting that the Adam optimizer, bidirectional encoding and combinations of them can also alleviate the effect of vanishing gradients.
5. The Adam optimizer performs generally better than the SGD optimizer and Bidirectional encoding performs better with less layers.⁵
6. The number of layers seems to matter less in our model than other parameters such as optimizers and encoding mechanisms, though it is notable that the more layers the longer the training time.
7. The number of identical non-overlapping statements is generally proportional to the total number of identical statements.

5.3 The Effect of the Number of Units and the Final Result

We now train our models by fixing other hyperparameters and varying the number of units. Our results in Table 8 show that performance generally gets better until 1024 units. The performance decreases when the number of units reaches 2048, which might indicate that the model starts to overfit. We have so far only used CPU versions of Tensorflow. The training real times in hours of our multi-core Xeon E5-2690 v4 2.60 GHz servers with 28 hyperthreading cores are also included to illustrate the usage of computational resources with respect to the number of units.

The best result achieved with 1024 units shows that after training for 11 h on the corpus of the 947231 aligned Mizar-L^AT_EX pairs, we can automatically translate with perfect accuracy 69179 (65.73%) of the 105247 testing pairs. Given that the translation includes quite nontrivial transformations, this is a surprisingly good performance. Also, by manually inspecting the remaining misclassifications we have found that many of those are actually semantically correct translations, typically choosing different but synonymous expressions. A simple example of such synonyms is the Mizar expression *for x st P(x) holds Q(x)*, which can be alternatively written as *for x holds P(x) implies Q(x)*. Since there are many such synonyms on various levels and they are often context-dependent, the true *semantic performance* of the translator will have to be measured by further applying the translation [22] from Mizar to MPTP/TPTP to the current results, and calling ATP systems to establish equivalence with the original Mizar formula as we do for Flyspeck in [11]. This is left as future work.

5.4 Greedy Covers and Edit Distances

We illustrate the combined performance of translation by comparing against selected collections of models. In Table 9 “Top- n Greedy Cover” denotes a list of n models such that each model in the list gives the maximum increase of correct translations from the previous model. In addition, we also measure the percentage of sentences (both overlap and no-overlap part) that are nearly correct. The metric of nearness we use is the word-level minimum editing distance (Levenshtein distance). We can see from Table 9 that reasonably correct translations can be generated by just using a combination of a few models.

⁵ Bidirectional encoding only works on even number of layers.

Table 7. Evaluation on various hyperparameters w.r.t. layers

Parameter	Final test perplexity	Final test BLEU	Identical statements (%)	Identical no-overlap (%)
2-Layer	3.06	41.1	40121 (38.12%)	6458 (13.43%)
3-Layer	2.10	64.2	57413 (54.55%)	16318 (33.92%)
4-Layer	2.39	45.2	49548 (47.08%)	11939 (24.82%)
5-Layer	5.92	12.8	29207 (27.75%)	2698 (5.61%)
6-Layer	4.96	20.5	29361 (27.9%)	2872 (5.97%)
2-Layer Residual	1.92	54.2	57843 (54.96%)	16511 (34.32%)
3-Layer Residual	1.94	62.6	59204 (56.25%)	17396 (36.16%)
4-Layer Residual	1.85	56.1	59773 (56.79%)	17626 (36.64%)
5-Layer Residual	2.01	63.1	59259 (56.30%)	17327 (36.02%)
6-Layer Residual	NaN	0	0 (0%)	0 (0%)
2-Layer Adam	1.78	56.6	61524 (58.46%)	18635 (38.74%)
3-Layer Adam	1.91	60.8	59005 (56.06%)	17213 (35.78%)
4-Layer Adam	1.99	51.8	57479 (54.61%)	16288 (33.86%)
5-Layer Adam	2.16	54.3	54670 (51.94%)	14769 (30.70%)
6-Layer Adam	2.82	37.4	46555 (44.23%)	10196 (21.20%)
2-Layer Adam Res.	1.75	56.1	63242 (60.09%)	19716 (40.97%)
3-Layer Adam Res.	1.70	55.4	64512 (61.30%)	20534 (42.69%)
4-Layer Adam Res.	1.68	57.8	64399 (61.19%)	20353 (42.31%)
5-Layer Adam Res.	1.65	64.3	64722 (61.50%)	20627 (42.88%)
6-Layer Adam Res.	1.66	59.7	65143 (61.90%)	20854 (43.35%)
2-Layer Bidirectional	2.39	69.5	63075 (59.93%)	19553 (40.65%)
4-Layer Bidirectional	6.03	63.4	58603 (55.68%)	17222 (35.80%)
6-Layer Bidirectional	2	56.3	57896 (55.01%)	16817 (34.96%)
2-Layer Adam Bi.	1.84	56.9	64918 (61.68%)	20830 (43.30%)
4-Layer Adam Bi.	1.94	58.4	64054 (60.86%)	20310 (42.22%)
6-Layer Adam Bi.	2.15	55.4	60616 (57.59%)	18196 (37.83%)
2-Layer Bi. Res.	2.38	24.1	47531 (45.16%)	11282 (23.45%)
4-Layer Bi. Res.	NaN	0	0 (0%)	0 (0%)
6-Layer Bi. Res.	NaN	0	0 (0%)	0 (0%)
2-Layer Adam Bi. Res.	1.67	62.2	65944 (62.66%)	21342 (44.37%)
4-Layer Adam Bi. Res.	1.62	66.5	65992 (62.70%)	21366 (44.42%)
6-Layer Adam Bi. Res.	1.63	58.3	66237 (62.93%)	21404 (44.50%)

5.5 Translating from Mizar to \LaTeX

It is interesting to see how the seq2seq model performs on our data when we treat Mizar as the source language and \LaTeX as the target language, thus emulating Bancerek’s translation toolchain. The results in Table 10 show that the model is still able to achieve meaningful translations from Mizar to \LaTeX , though the translation quality is generally not yet as good as in the other direction.

Table 8. Evaluation on number of units

Parameter	Final test perplexity	Final test BLEU	Identical statements (%)	Identical no-overlap (%)	Training time (hrs.)
128 Units	3.06	41.1	40121 (38.12%)	6458 (13.43%)	1
256 Units	1.59	64.2	63433 (60.27%)	19685 (40.92%)	3
512 Units	1.6	67.9	66361 (63.05%)	21506 (44.71%)	5
1024 Units	1.51	61.6	69179 (65.73%)	22978 (47.77%)	11
2048 Units	2.02	60	59637 (56.66%)	16284 (33.85%)	31

Table 9. Coverage w.r.t. a set of models and edit distances

	Identical Statements	0	≤ 1	≤ 2	≤ 3
Best Model	69179 (total)	65.73%	74.58%	86.07%	88.73%
- 1024 Units	22978 (no-overlap)	47.77%	59.91%	70.26%	74.33%
Top-5 Greedy Cover	78411 (total)	74.50%	82.07%	87.27%	89.06%
- 1024 Units	28708 (no-overlap)	59.68%	70.85%	78.84%	81.76%
- 4-Layer Bi. Res.					
- 512 Units					
- 6-Layer Adam Bi. Res.					
- 2048 Units					
Top-10 Greedy Cover	80922 (total)	76.89%	83.91%	88.60%	90.24%
- 1024 Units	30426 (no-overlap)	63.25%	73.74%	81.07%	83.68%
- 4-Layer Bi. Res.					
- 512 Units					
- 6-Layer Adam Bi. Res.					
- 2048 Units					
- 2-Layer Adam Bi. Res.					
- 256 Units					
- 5-Layer Adam Res.					
- 6-Layer Adam Res.					
- 2-Layer Bi. Res.					
Union of All 39 Models	83321 (total)	79.17%	85.57%	89.73%	91.25%
	32083 (no-overlap)	66.70%	76.39%	82.88%	85.30%

Table 10. Evaluation on number of units

Parameter	Final test perplexity	Final test BLEU	Identical statements	Percentage
512 Units Bidirectional Scaled Luong	2.91	57	54320	51.61%

6 A Translation Example

To illustrate the training of the neural network, we pick a specific example (again BHSP_2:3 as in Sect. 3) and watch how the translation changes as the training progresses. We can see from Table 11 that the model produces mostly gibberish in the early phases of the training. As the training progresses, the generated sentence starts to look more like the correct Mizar statement. It is interesting

Table 11. Translation with respect to training steps

Rendered \LaTeX	Suppose s_8 is convergent and s_7 is convergent . Then $\lim(s_8+s_7) = \lim s_8 + \lim s_7$
Input \LaTeX	Suppose $\{ s_{8} \}$ is convergent and $\{ s_{7} \}$ is convergent . Then $\mathop{\mathrm{lim}} (\{ s_{8} \} + \{ s_{7} \}) \mathrel{=} \mathop{\mathrm{lim}} \{ s_{8} \} + \mathop{\mathrm{lim}} \{ s_{7} \}$.
Correct	$seq1$ is convergent & $seq2$ is convergent implies $\lim (seq1 + seq2) = (\lim seq1) + (\lim seq2)$;
Snapshot-1000	x in dom f implies $(x * y) * (f (x (y (y y)))) = (x (y (y (y y))))$;
Snapshot-3000	seq is convergent & $\lim seq = 0c$ implies $seq = seq$;
Snapshot-5000	$seq1$ is convergent & $\lim seq2 = \lim seq2$ implies $\lim_inf seq1 = \lim_inf seq2$;
Snapshot-7000	seq is convergent & $seq9$ is convergent implies $\lim (seq + seq9) = (\lim seq) + (\lim seq9)$;
Snapshot-9000	$seq1$ is convergent & $\lim seq1 = \lim seq2$ implies $(seq1 + seq2) + (\lim seq1) = (\lim seq1) + (\lim seq2)$;
Snapshot-12000	$seq1$ is convergent & $seq2$ is convergent implies $\lim (seq1 + seq2) = (\lim seq1) + (\lim seq2)$;

to see that the neural network is able to learn the matching of parentheses and correct labeling of identifiers.

7 Conclusion and Future Work

We for the first time harnessed neural networks in the formalization of mathematics. Due to the lack of aligned informal-formal corpora, we generated informalized \LaTeX from Mizar by using and modifying the current translation done for the journal Formalized Mathematics. Our results show that for a significant proportion of the inference data, neural network is able to generate correct Mizar statements from \LaTeX . In particular, when trained on the 947,231 aligned Mizar- \LaTeX pairs, the best method achieves perfect translation on 65.73% of the 105,247 test pairs, and the union of all methods produces perfect translations on 79.17% of the test pairs.

Even though these are results on a synthetic dataset, such a good performance is surprising to us and also very encouraging. It means that state-of-the-art neural methods are capable of learning quite nontrivial informal-to-formal transformations, and have a great potential to help with automating computer understanding of mathematical and scientific writings.

It is also clear that many of the translations that are currently classified by us as imperfect (i.e., syntactically different from the aligned formal statement) are semantically correct. This is due to a number of synonymous formulations allowed by the Mizar language. Obvious future work thus includes a full semantic evaluation, i.e., using translation to MPTP/TPTP and ATP systems to check if the resulting formal statements are equivalent to their aligned counterparts. As in [11, 12], this will likely also show that the translator can produce semantically different, but still provable statements and conjectures.

Another line of research opened by these results is an extension of the translation to full informalized Mizar proofs, then to the ProofWiki corpus aligned by Bancerek recently to Mizar, and (using these as bridges) eventually to arbitrary L^AT_EX texts. The power and the limits of the current neural architectures in automated formalization and reasoning is worth of further understanding, and we are also open to the possibility of adapting existing formalized libraries to tolerate the great variety of natural language proofs.

References

1. Bancerek, G.: Automatic translation in formalized mathematics. *Mech. Math. Appl.* **5**(2), 19–31 (2006)
2. Bancerek, G., et al.: Mizar: state-of-the-art and beyond. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) *CICM 2015. LNCS (LNAI)*, vol. 9150, pp. 261–279. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20615-8_17
3. Bancerek, G., Carlson, P.: Mizar and the machine translation of mathematics documents (1994). http://www.mizar.org/project/banc_carl93.ps
4. Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards QED. *J. Formaliz. Reason.* **9**(1), 101–148 (2016)
5. Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734. Association for Computational Linguistics, October 2014
6. Cybenko, G.: Approximation by superpositions of a sigmoidal function. *Math. Control Sig. Syst.* **2**(4), 303–314 (1989)
7. Goodfellow, I., Bengio, Y., Courville, A.: *Deep Learning*. MIT Press, Cambridge (2016). <http://www.deeplearningbook.org>
8. Hales, T.: *Dense Sphere Packings: A Blueprint for Formal Proofs*. London Mathematical Society Lecture Note Series, vol. 400. Cambridge University Press, Cambridge (2012)
9. Hales, T.C., Adams, M., Bauer, G., Dang, D.T., Harrison, J., Hoang, T.L., Kaliszyk, C., Magron, V., McLaughlin, S., Nguyen, T.T., Nguyen, T.Q., Nipkow, T., Obua, S., Pleso, J., Rute, J., Solovyev, A., Ta, A.H.T., Tran, T.N., Trieu, D.T., Urban, J., Vu, K.K., Zumkeller, R.: A formal proof of the Kepler conjecture. *Forum Math. Pi* **5**, e2 (2017)
10. Hornik, K.: Approximation capabilities of multilayer feedforward networks. *Neural Netw.* **4**(2), 251–257 (1991)

11. Kaliszzyk, C., Urban, J., Vyskočil, J.: Automating formalization by statistical and semantic parsing of mathematics. In: Ayala-Rincón, M., Muñoz, C.A. (eds.) ITP 2017. LNCS, vol. 10499, pp. 12–27. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66107-0_2
12. Kaliszzyk, C., Urban, J., Vyskočil, J.: Learning to parse on aligned corpora (rough diamond). In: Urban, C., Zhang, X. (eds.) ITP 2015. LNCS, vol. 9236, pp. 227–233. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22102-1_15
13. Kaliszzyk, C., Urban, J., Vyskočil, J., Geuvers, H.: Developing corpus-based translation methods between informal and formal mathematics: project description. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) CICM 2014. LNCS (LNAI), vol. 8543, pp. 435–439. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08434-3_34
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates Inc., New York (2012)
15. Luong, M., Brevdo, E., Zhao, R.: Neural machine translation (seq2seq) tutorial (2017). <https://github.com/tensorflow/nmt>
16. Maddison, C.J., Huang, A., Sutskever, I., Silver, D.: Move evaluation in Go using deep convolutional neural networks. CoRR, abs/1412.6564 (2014)
17. Neubig, G.: Neural machine translation and sequence-to-sequence models: a tutorial. CoRR, abs/1703.01619 (2017)
18. Olah, C.: Understanding LSTM networks. <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
19. Papineni, K., Roukos, S., Ward, T., Zhu, W.: BLEU: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, pp. 311–318. ACL (2002)
20. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Proceedings of the 27th International Conference on Neural Information Processing Systems, NIPS 2014, vol. 2, pp. 3104–3112. MIT Press, Cambridge (2014)
21. Trybulec, A., Rudnicki, P.: A collection of TeXed Mizar abstracts (1989). <http://www.mizar.org/project/TR-89-18.pdf>
22. Urban, J.: MPTP 0.2: design, implementation, and initial experiments. *J. Autom. Reason.* **37**(1–2), 21–43 (2006)
23. Urban, J., Vyskočil, J.: Theorem proving in large formal mathematics as an emerging AI field. In: Bonacina, M.P., Stickel, M.E. (eds.) Automated Reasoning and Mathematics: Essays in Memory of William W. McCune. LNCS (LNAI), vol. 7788, pp. 240–257. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36675-8_13



Deep Learning for Math Knowledge Processing

Abdou Youssef^{1,2}(✉) and Bruce R. Miller²

¹ The George Washington University, Washington, D.C., USA
ayoussef@gwu.edu

² Applied and Computational Mathematics Division, NIST, Gaithersburg, MD, USA
{youssef,miller}@nist.gov

Abstract. The vast and fast-growing STEM literature makes it imperative to develop systems for automated math-semantics extraction from technical content, and for semantically-enabled processing of such content. Grammar-based techniques alone are inadequate for the task. We present a new project for using deep learning (DL) for that purpose. It will explore a number of DL and representation-learning models, which have shown superior performance in applications that involve sequences of data. As math and science involve sequences of text, symbols and equations, such as deep learning models are expected to deliver good performance in math-semantics extraction and processing.

The project has several goals: (1) to apply different DL models to math-semantics extraction and processing, designing more suitable models as needed, for such foundational tasks as accurate tagging and automated translation from \LaTeX to semantically-resolved machine understandable forms such as cMathML; (2) to create and make available to the public labeled math-content datasets for model training and testing, and Word2Vec/Math2Vec representations derived from large math datasets; and (3) to conduct extensive comparative performance evaluations gaining insights into which DL models, data representations, and traditional machine learning models, are best for the above-stated goals.

1 Introduction

The amount of STEM knowledge is so vast and growing so rapidly that it is impossible to keep current and take full advantage of that knowledge, causing many lost opportunities for advances, and wasted time reinventing wheels. Therefore, automated math-semantic extraction from technical content, and semantically-enabled processing of such content, which are here our overarching goals, are essential for future progress and for the synthesis of large and exponentially growing volumes of technical text.

Much of the mathematical literature, old and new, is now online^{1,2,3,4,5,6,7} mostly in \LaTeX [25] and PDF formats, and to a lesser extent in presentation-MathML⁸. The pressing needs and wide availability online have prompted ongoing research in math content processing and math knowledge management [6]. Continuing progress in those areas and in applications thereof relies on being able to determine the semantics of the technical/math literature in an automated way. Great strides have been made in doing this for natural language text [22, 35], such as word-sense disambiguation and semantics determination. The technical literature, however, is not just natural language but is full of math expressions. The latter makes it very challenging to determine the semantics of technical content by machine for many reasons, such as the higher levels of abstraction and the implicit knowledge that technical authors assume.

In [48], Youssef presented a parts-of-math (POM) tagging project which, in its early stages, uses grammar-based and knowledge-based approaches for parsing and for tagging math terms with a combination of definite tags and *tentative* tags. In later stages, machine learning (ML) techniques [19, 33] will be used to disambiguate between the tags. Meanwhile, Miller, who has developed the widely used \LaTeX XML conversion tool [31], has been creating and refining techniques for preserving and sometimes inferring math semantics from \LaTeX sources. The tool tries to balance faithful emulation of the \TeX engine, while preserving whatever semantics may be implied by the markup used. The DLMF project [39] leverages that approach by developing and using semantic markup which declares the intent of the symbols and structures being used. This is a labor-intensive, and as yet, unfinished, task.

Other efforts [11, 15, 24, 36, 38, 43, 44, 46, 47] are developing approaches for math language processing. Much like \LaTeX XML and the POM tagger, most rely on grammar-based and other traditional NLP techniques [22]. Such techniques alone are inadequate for semantics extraction from informal/ \LaTeX -formatted technical content, and for translating from human readable forms to machine understandable, computable form. Fortunately, recent ML techniques, especially deep neural networks (DNNs) [4], have shown great success in NLP [23, 28, 29, 32, 41] and object recognition [27], suggesting potential for math-semantics extraction.

Therefore, this new project will use deep learning for automated math-semantics inferencing from \LaTeX -formatted math documents. It will draw on \LaTeX XML and the POM tagger for tentative semantic tagging, and then use deep learning (DL) models and other machine learning (ML) techniques for

¹ arXiv.org, <https://arxiv.org/>.

² NIST Digital Library of Mathematical Functions (DLMF) <https://dlmf.nist.gov/>.

³ (World) Digital Mathematics Library, <https://www.math.unibielefeld.de/~rehmann/DML/dml.links.html>.

⁴ The European Digital Mathematics Library, <https://eudml.org/>.

⁵ Göttinger Digitalisierungszentrum, <http://gdz.sub.uni-goettingen.de/gdz/>.

⁶ The database MathSciNet, <http://www.ams.org/mathscinet/>.

⁷ The database zbMATH, <http://www.zentralblatt-math.org/zbmath/>.

⁸ <https://www.w3.org/Math/>.

disambiguation. Another NIST project, namely the DLMF [39], will be a source of reliably annotated training and testing data.

To achieve our overarching goals, we will explore certain DL models, including unsupervised representation-learning models [2, 3, 9, 28–30, 32, 40, 41], and supervised learning models such as feedforward DNNs [4], Recurrent Neural Networks (RNNs) [7, 9], Long Short-Term Memory (LSTM) networks [21], and Convolutional Neural Networks (CNNs) [23, 27]. These learning models have shown superior performance in NLP representations and classification, and the last 3 models in applications involving ordered sequences (e.g., sentences) [17, 45], such as automatic language translation [8, 9, 12]. Therefore, we expect that those models will be effective in similar math applications such as semantic extraction and automated translation from \LaTeX to machine-computable forms.

2 Objectives of the Project

The project has the following major objectives:

1. To apply different DNNs, especially RNNs [7, 9], LSTMs [21], and CNNs [23], and different representation-learning word-embedding models (Word2Vec, Doc2Vec) [29, 30, 32, 41], for math-entity representation learning as well as semantics extraction, and if needed, to design new and better suitable DL models for these purposes. The resulting math-entity embedding networks will be termed Math2Vec (embedding) models.

Math-semantics disambiguation methods, illustrated in Sect. 6, will be developed. Also, sequence-to-sequence DNNs will be trained and tested, for such foundational tasks as automated translation from informal/ \LaTeX form to machine understandable forms such as computer programs and semantic math markup (e.g., cMathML).

Towards this objective, extensive comparative performance evaluations will be conducted to gain insight into which DNN models, designs, data representations, and such, are best for math-semantics extraction and processing.

2. To train traditional machine learning models, such as Support Vector Machines [10] and Random Forests [5], using the Math2Vec-generated feature vectors of mathematical entities, and to test these traditional ML models for performing math-semantic extraction and disambiguation tasks.

Again, extensive performance evaluations of those traditional ML models, and comparison between them and the DNN models identified in the first objective, will be conducted to determine whether the more data-demanding and more time-consuming DNNs deliver more performance than those less-demanding, traditional ML techniques, in math-semantic extraction.

3. To create and make available to the public (a) sizable, labeled math-content datasets for DL (and other ML) training and testing, (b) Math2Vec models and representations (derived from large math-content datasets) along with software APIs for accessing and interacting with those models and representations, and (c) trained, tested, and optimized DL and ML models (along with API software) for ready use by the math processing community.

3 Significance and Applications of the Project

The semantic extraction capabilities that will result from this project, and from the related projects on which it builds, can have many applications, such as:

- Automated semantic enrichment and annotations of digital math libraries. This leads to better math search and math UIs, as explained next.
- Search [18]: Semantically-enabled, ‘deeper’ math search finds hits based not only on literal keyword matches but on semantic matches [16].
- Better math user interfaces: Semantic annotations of math symbols and expressions provide on-demand explanations/reminders as well as hyperlink connections between various math entities.
- Robust, reliable and accurate automated presentation-to-computation (P2C) conversion of math expressions from natural forms like \LaTeX or presentation MathML into machine-computable forms such as Content MathML, computer algebra systems [11], automated reasoning systems and proof assistants. Such ambitious applications become possible when the correct semantics of symbols and expressions can be inferred.
- Math Question/Answer capabilities at both manuscript and collection levels.

In addition, the resulting R&D resources and capabilities (Goal 3 in Sect. 2) can greatly aid the math-processing research community in developing considerably more advanced math-processing tools and applications, for the benefit of mathematicians, scientists, and end-users of mathematics.

4 Foundations and Related Work

Deep learning has emerged in recent years as the most powerful machine learning paradigm for classification, object recognition, computer vision, machine translation, speech recognition, representation learning, and the like [13]. This was attributed to availability of much larger datasets and more powerful computers (e.g., GPUs) for training, and to advances in design and optimization methods of deep (i.e., many-layer) neural networks [13].

Especially relevant to text processing are the recent advances in unsupervised representation learning (RL), especially *word embedding* [2, 3, 9, 28–30, 32, 40, 41] and more recent structured [42] and hierarchical [37] embeddings, and sequence-to-sequence supervised learning models [17, 45].

The RL models take as input a text collection, and generate a numerical feature vector (typically 100D or 300D), called *embedding*, for each word (or sentence/paragraph) in the collection. This vector captures latent semantics of a word from the contexts of its occurrences in the collection; in particular, words that co-occur often in close proximity tend to have similar feature vectors (where similarity is measured by the cosine, the Euclidean distance, etc.). Consider an example of semantic similarity captured by Word2Vec [32, 40]. Let $V(w)$ denote the Word2Vec-derived feature vector of word w . It was found that $V(\text{man}) - V(\text{woman}) \approx V(\text{king}) - V(\text{queen})$. Thus to find the female form of **king**

(knowing **man** and **woman**), one searches for the word whose vector is closest to $U = V(\mathbf{man}) - V(\mathbf{king}) + V(\mathbf{queen})$, and discovers the desired answer, **queen**.

This project will use Word2Vec and similar embedders (e.g., GloVe and Doc2Vec [20]) to create a Math2Vec embedding model for generating feature vectors for various math symbols, and then explore vector space operations on such feature vectors to determine/disambiguate the meanings and roles of math entities in their contexts. Examples of our novel use of Math2Vec embeddings are provided in Sect. 6. Note that Math2Vec will subsume Word2Vec and Doc2Vec, working on text and math content in an integrated way, thus helping to determine the semantics of math entities not just from the equations containing them, but also for the text surrounding them.

Applications of embedding to math language processing have started to emerge, such as equation embedding [26]. Equation embedding is useful for identifying similar equations and contextual descriptive keywords; the latter can be used as tentative tags (much like the ones identified by our POM tagger), but will need to be further disambiguated, which is one of the foci of this project, as illustrated in Sect. 6.

With regard to DNNs, certain models such RNNs [7, 9] and LSTMs [21] have shown a great ability to learn sequence-to-sequence mappings [17, 45]. For example, they can be trained to map an arbitrary input sentence from one language to another. When properly trained on two parallel corpora in two languages, the trained model becomes a language translator. In this project, we will exploit this potential to train similar DNNs to become \LaTeX -to-cMathML translators.

Note that for certain supervised learning tasks, such as disambiguating the meaning of a math token, and classifying a math document (as to its math area), traditional ML models such as Support Vector Machines and Random Forests could be used. SVMs and RFs have the advantage of needing much less training data and training time than DNNs because of the former’s much smaller number of parameters to optimize, but DNNs have shown considerably higher performance (e.g., accuracy) in complicated tasks [4]. However, one can combine the benefits of both. Specifically, embedding models can be used to learn good feature vectors of math entities, and then use those vectors as input to train SVMs and RFs. This hybrid approach will be explored in this project.

Aside from deep learning and other ML models, this project will build upon two other projects/tools by the authors, as explained in some detail in the next two subsections.

4.1 The Parts-of-Math Tagger

In [48], a part-of-math (POM) tagger was described. In that project, which is still ongoing, a tagset for math terms and expressions, as well as various relevant features, were identified (see Table 1 for a brief summary), and algorithms and software for the early stages of the tagger were developed. In those early stages, the tagger takes as input a math document, and tags each math term and some sub-expressions with two kinds of tags. The first kind consists of definite tags (such as operation, relation, numerator, etc.) that the tagger is certain of. The

Table 1. Features tentatively computed by the POM tagger, and to be disambiguated in this deep learning project.

Feature	Explanation
Category	The grammatical role or part-of-math: <i>operation, operator, relation, function, variable, parameter, constant, quantifier, separator, punctuation, abbreviation/acronym, delimiter, left-delimiter, right-delimiter, constructor, accent, etc.</i>
Subcategory	Further specializes the category, e.g., for ‘<’, the category is <i>relation</i> and the subcategory is <i>order</i> . Values can be <i>order, numerator, denominator, lower-limit (of an integral), constraint/condition, definition, etc.</i> For an accent, indicates the accent position (e.g. pre-superscript)
Signature	The data type, and, for functions and operators, the arity and types of arguments. which arguments are variables and which are parameters, May also include notational aspects such as whether it has subscripts/superscripts/prescripts
Font	The font characteristics: <i>Typeface, Font-style, and Font-weight</i>
Status	Whether the notation is <i>Generic, Standard</i> (i.e. commonly understood), or <i>Defined</i> (in the manuscript), and whether it is implicitly or explicitly defined

second kind consists of alternative, tentative features (including alternative roles and meanings) drawn from a knowledge base that was developed for that project. Those tags, and especially the tentative features, will be heavily utilized in this deep learning project in ways that are illustrated in Sect. 6. Also, in [48], a number of math ambiguities were identified, a summary of which is provided here in Table 2. It is those ambiguities that we will disambiguate using deep learning (and new algorithms), again as illustrated in Sect. 6.

4.2 \LaTeX ml

\LaTeX XML is a \LaTeX to XML converter, whose internal XML format can be converted to standard formats such as HTML, MathML (both content and presentation), among others. \LaTeX XML will play two critical roles in this project. Firstly, to provide access to large corpora of scientific documents, such as those in arXiv; Secondly, in conjunction with the DLMF, to deliver a more moderate sized, but curated, collection of semantically tagged mathematical formula.

Since \TeX documents are essentially programming markup embedded within the document text, simple parsing techniques to extract the plain text for input to DNN’s, let alone the mathematics, are destined for failure. One must execute, rather than parse, a \TeX document. \LaTeX XML therefore mimics the \TeX engine converting the documents to an internal XML format from which the plain text will easily be extracted, including as much or as little of the styling and metadata as is desired. It is also highly extensible, providing for declarative markup and semantic macros which allow embedding the semantic tagging within the documents. Moreover, the mathematical portions can be converted to math lexemes

Table 2. Math ambiguities identified in the POM project [48].

Ambiguity	Explanation
Superscript	Can indicate a power, an index, the order of differentiation, a postfix unary operator, etc.
Juxtaposition	Can imply multiplication, concatenation, or function application
Accent	Can represent an applied operator, or be a morphological part of the name. E.g. ‘ y' ’ can be the derivative of y or simply a distinct variable
Part of math	Different roles give rise to distinct parse trees, as well as different semantics. E.g. ‘ ’ and ‘ ’ can be punctuation, operators, relations, or delimiters. Specific fencing delimiters can imply very specific semantics. E.g. the bra/ket notations ‘ $\langle \cdot $ ’, ‘ $ \cdot \rangle$ ’, compared to the inner product ‘ $\langle \cdot, \cdot \rangle$ ’
Scope	Typically occurring when delimiters are omitted. E.g. ‘ $\sin 2\pi x + 5$ ’ is likely intended to mean ‘ $\sin(2\pi x) + 5$ ’
Data type	Necessary to completely resolve semantics; conversely, can help disambiguate other ambiguities, e.g. Superscript

which uniquely capture each symbol and the markup-imposed structure (e.g., for fractions). Thus, a large subset of potentially complexly formatted scientific documents in arXiv become available for feeding into a DNN.

In \LaTeX currently, we start from a fundamentally declarative and grammatical point of view: Namely the task is to represent mathematical expressions as a correctly structured trees, with each node in the tree labeled with its semantic intent. Correctly assigned grammatical roles, or part-of-math, for each symbol, is sufficient to determine the correct tree, given a rich enough grammar for mathematical expressions. Of course, many but far from all symbols used in \LaTeX markup have universally agreed roles. The wrong choice, say between a simple variable or a function, leads to different trees (again, see Table 2). Thus, the first task in the DLMP was to specify declarative markup to specify grammatical roles. This being largely accomplished, the attention turns to disambiguating the meanings of the symbols as well. Here, semantic macros for the mathematical objects comes to aid. In its current form, the semantics of DLMP’s mathematics are partly resolved, with much yet to be done. We will continue this process, searching for the sweet-spot between impractical fully explicit markup and the AI which the current project aims to achieve.

5 The Datasets to Be Collected, Created and Used

The data collected, created, and used in this project divide into several sets.

A— The term-labeled dataset: This dataset will consist of many math documents where (nearly) all the math terms are correctly labeled. It will be initially derived from the DLMP, where thoroughly vetted annotations of the math terms are available. More labeled documents will be added later.

The labeled dataset will have a dual-purpose use: (1) for training and testing supervised learning models, and (2) as a resource for disambiguating math semantics of new math documents, as will be illustrated in Sect. 6.

- B– The Doc-class-labeled Dataset:** Math-term disambiguation will utilize feature-based classifiers, and one feature to be used is the math field(s) of the input document. This dataset will be used to train document-classifiers. It is being collected from several sources, *e.g.*, the ArXiv and the DLMF.
- C– The Dataset for generating Math2Vec Embeddings:** This dataset will be used for (1) testing the effectiveness of a variety of Math2Vec embedding algorithms, and (2) generating Math2Vec embeddings for a large vocabulary of math text and math symbols. It will include the above datasets A and B, and more, and will be quite large and cover many math areas so that the embeddings can be used by the community as readily available feature vectors of math terms.
- D– Application-specific labeled datasets:** Labeled datasets will be created for specific sequence-to-sequence translations, *e.g.*, \LaTeX -to-MathML translation. Such a dataset will consist of semantically tagged math expressions in \LaTeX , and their corresponding MathML formats. They will be used for training and testing supervised models specific to the selected applications.

6 Samples of Tagging and Disambiguation Algorithms

This section gives a few new algorithms and techniques which illustrate the deep learning methods that will be developed in this project for semantic extraction/tagging, and for semantic disambiguation. Such algorithms will rely on a fully term-labeled dataset (*e.g.*, dataset A), which will be also referred to as the *reference document(s)* or simply, *Resources*, and denoted R . We will assume that Math2Vec is applied to R once offline to vectorize all its terms.

We will use the following notations and definitions in this section.

D a document in the dataset.

$t \in D$ a term occurring in document D , where a term is a word or an artificial lexeme representing a mathematical token.

$\langle xy \dots \rangle \in D$ a sequence of terms occurring in document D .

$y \in \langle xy \dots \rangle \in D$ a term that is part of a sequence of terms occurring in document D .

$V_D(t)$ The feature vector (*i.e.*, embedding) of a term $t \in D$, derived by running Math2Vec on D .

R The above-defined Resources, assumed to be fully term-labeled, and vectorized offline by a Math2Vec so that $V_R(t)$ are available for all terms $t \in R$.

Tags($t \in R$) The set of morphological/lexical/syntactic/semantic tags of a term $t \in R$; they are the provided for all terms t in Resources R .

TTags($t \in D$) The *tentative* tags of term t in D . Within each tag category, there are several tags that t can have, tentatively, where the precise tag in that category is yet to be determined. TTags($t \in D$) can be derived by a basic math tagger, like the first-stage of the POM tagger [48].

Algorithm 1. math-Tag(D, R)

Require: Document D as input;
Require: Resources R , tagged, and pre-vectorized with Math2Vec;
 $\{V_D(t) \mid \forall \text{ terms } t \in D\} \leftarrow$ Call Math2Vec on D ;
 $\{\text{Tags}(t \in D) \mid \forall \text{ terms } t \in D\} \leftarrow$ Call the basic math tagger on D ;
for all $t \in D$ **do** *//find closest tentative match in R to t*
 $S \leftarrow \{w \in R \mid \text{Tags}(w \in R) \subseteq \text{Tags}(t \in D)\}$;
 $\tau \leftarrow \arg \min_{w \in S} \|V_R(w) - V_D(t)\|^2$; *// The $V_R(w)$'s are available in R*
 $\text{Tags}(t \in D) \leftarrow \text{Tags}(\tau \in R)$; *//Tags($\tau \in R$) are available in R*
end for
return $\text{Tags}(t \in D)$ for all terms $t \in D$;

$\|v - w\|^2$ A measure of the difference between two feature vectors; it can be one of several different norms, e.g., $\|\cdot\|_2$ or $\|\cdot\|_1$.

$\Delta_K(t, s) = V_K(s) - V_K(t)$ The difference between two feature vectors within a given document; it captures some relationship between t and s .

$\text{match}_{R,D}(t, y)$ Given a term $y \in D$, a term $t \in R$ is called a *close tentative match* of y if

$$\text{Tags}(t \in R) \subseteq \text{Tags}(y \in D) \text{ and } \|V_R(t) - V_D(y)\|^2 < \delta$$

for some preset small threshold δ . That is, the (definite) tags of t in R are among the tentative (i.e., possible/likely) tags of y in D .

6.1 Tagging and Semantic Extraction

Algorithm 1 is an algorithm for tagging/semantic extraction. Note that the norm used could be replaced by a similarity metric, such as the cosine similarity between two vectors, in which case, $\arg \min$ is replaced by $\arg \max$.

6.2 Disambiguation

Disambiguation of Primes. Primes (\prime) fall under accent ambiguities (see Table 2). A prime after a math symbol, e.g., y' , in a document D , can denote the derivative of y (as in the case where y is a differentiable function), or the logical complement of y (as in the case where y is a Boolean variable/function), or a morphological glyph where y' is simply another identifier having at most an abstract relation to y . In this subsection, we give an algorithm for disambiguating (\prime). It is assumed that the tokenizer initially treats every single prime as a separate token. Algorithm 2 presents a prime-disambiguating algorithm:

Disambiguation of Superscripts. Superscripts present another major ambiguity in math. Given a math term/expression like ' y^z ', representing ' y^z ', in document D , where z is a single term or an expression, the role of the superscript z is ambiguous. It can be the power exponent of y , or simply an index

Algorithm 2. disambiguate-prime('y'; D, R)

Require: Document D as input;
Require: Term pair 'y' as input, in document D ;
Require: Resources R , tagged, and pre-vectorized with Math2Vec;
 $\{\text{Tags}(t \in D) \mid \forall \text{ terms } t \in D\} \leftarrow$ Call the basic math tagger on D ;
 $\{V_D(t) \mid \forall \text{ terms } t \in D\} \leftarrow$ Call Math2Vec on D ;

// Find next in R all terms t that are closest tentative matches to y and occur as 't'
 $E \leftarrow \{t \in R \mid ('t' \in R) \ \& \ \text{match}_{R,D}(t, y)\}$

$\tau \leftarrow \arg \min_{t \in E} \|\Delta_D(y, t) - \Delta_R(t, t)\|^2$; *// 'τt' most resembles 'y' of all 't' ∈ R*

// Find next the most frequent tag of t ∈ 'τt' ∈ R

$tag \leftarrow \arg \max_{\xi \in \bigcup \text{Tags}(t \in 'τt' \in R)} \text{count}(\xi \in \text{Tags}(t \in 'τt' \in R))$;

if $tag = \text{derivative}$ **then**
 'y' is tagged as the derivative of y ;
else if $tag = \text{booleancomplement}$ **then**
 'y' is tagged as the Boolean complement of y ;
else if tag designates a glyph **then** *// as a morphological identifier name*
 'y' is tagged as an identifier.
end if

return tag ;

of y , or the order of the derivative of y (as for example, $y^{(3)}$ denotes the third derivative of function y), or yet other alternatives. We give in Algorithm 3 a method for disambiguating the superscript.

Note that for more accuracy, where possible, one can increase the constraints on t in the first step of the disambiguate-superscript algorithm. For example, if z is enclosed between parentheses in 'y^z', we can require the term after '^' in ('t^' ∈ R) to be also enclosed between parentheses.

Disambiguation of Juxtaposition. Another major math ambiguity is the juxtaposition ambiguity (See Table 2). We focus in this subsection on a small but important manifestation of this ambiguity. Given $y(x \dots)$ in document D , it is unclear what the 'suppressed operation' between y and $(x \dots)$ is. It can be the multiplication of y with $(x \dots)$, or the application of the function y at argument $(x \dots)$. If y is definitely known to be a non-function numerical quantity, then there is no ambiguity—it is multiplication. But if it is not certain to the algorithm what y is, which oftentimes is the case, then the suppressed operation is ambiguous. Even if y is known to be a function, there is still a (small) chance that the suppressed operation can be either, and must therefore be resolved. Algorithm 4 shows how to disambiguate the juxtaposition operator.

In this project, similar algorithms will be developed for as many ambiguities as possible, and will be tested, and modified as necessary, to maximize their disambiguation accuracy.

Algorithm 3. disambiguate-superscript($y^{\wedge z}$; D, R)

Require: Document D as input;
Require: Term triple $y^{\wedge z}$ as input, in document D ;
Require: Resources R , tagged, and pre-vectorized with Math2Vec;
 $\{\text{Tags}(t \in D) \mid \forall \text{ terms } t \in D\} \leftarrow$ Call the basic math tagger on D ;
 $\{V_D(t) \mid \forall \text{ terms } t \in D\} \leftarrow$ Call Math2Vec on D ;

// Find in R all terms t that are closest tentative matches to y and occur as t^{\wedge}
 $E \leftarrow \{t \in R \mid (t^{\wedge} \in R) \ \& \ \text{match}_{R,D}(t, y)\};$

$\tau \leftarrow \arg \min_{t \in E} \|\Delta_D(y, \wedge) - \Delta_R(t, \wedge)\|^2$; *// t^{\wedge} most resembles y^{\wedge} of all $t^{\wedge} \in R$.*

// Find next the most frequent tag of $\wedge \in t^{\wedge} \in R$
 $\text{tag} \leftarrow \arg \max_{\xi \in \cup \text{Tags}(\wedge \in t^{\wedge} \in R)} \text{count}(\xi \in \text{Tags}(\wedge \in t^{\wedge} \in R));$

if $\text{tag} = \text{power}$ **then**
 $y^{\wedge z}$ is tagged as the expression ‘ y to the power z ’;
else if tag is an index **then**
 $y^{\wedge z}$ is tagged as ‘ y indexed by z ’;
else if tag designates an order of a derivative **then**
 $y^{\wedge z}$ is tagged as the ‘ z^{th} derivative of y ’.
else
...
end if
return tag ;

6.3 Classification-Based Disambiguation of Math-Tagging

The taggers/disambiguators of the previous subsections relied mainly on math-term embeddings, 1-nearest-neighbor (generalizable to K-nearest-neighbor [1, 14]) voting, and maximum likelihood. But math embeddings can be used as feature vectors for supervised training of classifiers (e.g., feedforward DNNs, SVM, RF, etc.) to disambiguate math tags. In this subsection, we illustrate briefly this supervised approach, focusing on the disambiguation of ‘ \prime ’.

Using the training subset of dataset A, train a classifier to classify the meaning/role of ‘ \prime ’ in an input ‘ y' ’ that is represented as the average (or concatenation) of both the embedding vector of y and the embedding of ‘ \prime ’, using the term-labels in A as training targets. Once the classifier is trained, to disambiguate the tag of an input ‘ y' ’ in an input math document D , call Algorithm 5.

This classification approach will be applied in this project to all the math ambiguities summarized in Table 2, testing a number of classifiers and a number of embedding approaches.

7 DNN’s for Math Sequence-to-Sequence Translation

RNN models, such as LSTM networks, can be trained to map sequences to sequences. We will train and test such models to map Latex-formatted math

Algorithm 4. disambiguate-Mult-vs.-FunctionApply($'y(x \dots)'$; D, R)

Require: Document D as input;
Require: Term sequence $y(x \dots)$ as input, in document D ;
Require: Resources R , tagged, and pre-vectorized with Math2Vec;
 $\{\text{Tags}(t \in D) \mid \forall \text{ terms } t \in D\} \leftarrow$ Call the basic math tagger on D ;
 $\{V_D(t) \mid \forall \text{ terms } t \in D\} \leftarrow$ Call Math2Vec on D ;

*// Find in R all terms t that are closest tentative matches to y and occur as $t \cdot ('$
 $E \leftarrow \{t \in R \mid (t \cdot (' \in R) \ \& \ \text{match}_{R,D}(t, y)\}$;*

$\tau \leftarrow \arg \min_{t \in E} \|\Delta_D(y, '(') - \Delta_R(t, '(')\|^2$ *// '(' most resembles 'y(' of all $t \cdot (' \in R$.*

// Find next the most frequent tag of invisible operators $\cdot \in \tau \cdot (' \in R$
 $\text{tag} \leftarrow \arg \max_{\xi \in \cup \text{Tags}(\cdot \in \tau \cdot (' \in R)} \text{count}(\xi \in \text{Tags}(\cdot \in \tau \cdot (' \in R));$

// Now resolve the invisible operator according to the value of tag.

...

return tag;

Algorithm 5. supervised-disambiguate-prime($'y'$; D, C)

Require: Document D as input;
Require: Term pair $'y'$ as input, in document D ;
Require: Classifier C , a pre-trained classifier of $'$;
 $\{V_D(t) \mid \forall \text{ terms } t \in D\} \leftarrow$ Call Math2Vec on D ;
 $V_D(y') \leftarrow$ Average or concatenate $V_D(y)$ and $V_D(')$;
 $\text{tag} \leftarrow$ Call Classifier C on $V_D(y')$;
return tag;

expressions to MathML counterpart, using Dataset D for the training and testing. Each input in the training is a Latex-formatted math expression, and the target is the corresponding MathML translation.

8 Testing and Performance Evaluation

The testing and evaluation methods will depend on the specific model, algorithm or application being designed or considered. For example, for semantic tagging/disambiguation, the term-labeled dataset (A) will be divided into two subsets: (1) the 1st subset is for supervised-training of the tagger/disambiguator if the latter is a feature-based classifier, or, if it is a K-nearest-neighbor based tagger/disambiguator, the first subset will be used as the reference “memory” for majority-voting-based selection of the most likely math tag for an input math term being labeled; and (2) the 2nd subset is for testing. The performance metric to be used is *accuracy*, i.e., percentage of correctly tagged math terms in the test data subset of the labeled dataset A. For document classification, as another example, the dataset B will be divided into training set and testing set, and used for feature-based classifier training and testing, respectively, and accuracy will

be the performance metric used. Likewise, for sequence-to-sequence translation, dataset D will be used for training and testing as explained in Sect. 7, and accuracy (i.e., percentage of correctly translated expressions) will be used as the performance metric.

For Math2Vec embedding models, which are unsupervised, the datasets used need not be labeled. As for testing and evaluation of those models, we will use subjective and objective methods. The subjective evaluation will be in term-similarities and term-relationships, *e.g.*, is \cos to \arccos the same as \exp is to \log ? The objective evaluation will be somewhat indirect. For example, to compare two competing embedders, we train a semantic tagger (as a classifier) using as feature vectors the embeddings produced by one of the two embedders applied on the dataset (A), resulting in one tagger, and, in parallel, train the same original semantic tagger using the embeddings produced by the other embedder applied on the dataset (A). Then compare the performance of the two taggers on the test data subset of set (A), using term-label accuracy as metric. The embedder that leads to higher tagger accuracy is taken to be the better embedder.

9 Stages of the Project

The project will be completed in stages, using `deeplearning4j`⁹ as the baseline software for much of the development and testing. The stages can be summarized as follows:

Stage 1: Dataset Collection/labeling. A large corpus of documents from various areas of mathematics will be collected and used to compute embeddings (i.e. feature vectors). A subset of this corpus will be labeled by assigning, semi-manually, the meanings/roles of their math entities.

Stage 2: Create Math2Vec embeddings, models, software. The embeddings of a large set of math entities will be computed, using this dataset. A Math2Vec embedder will be developed and fine-tuned for this purpose.

Stage 3: Develop semantic extraction/disambiguation systems. DNNs as well as Support Vector Machines and Random Forests will be trained using the labeled dataset and Math2Vec embeddings to extract and/or disambiguate math semantics from documents. Considerable testing, optimization, modifications, and fine-tuning of the various learning models will be carried out, and conclusions will be drawn about best models and practices.

Stage 4: Develop math sequence-to-sequence models. Supervised learning models for math sequence-to-sequence translations of \LaTeX -to-cMathML will be designed, trained, tested and optimized.

Stage 5: Put resources in the public domain. These will include the datasets A, B, C, and D described in Sect. 5.

⁹ <https://deeplearning4j.org/>.

10 Current State of the Project

- A dataset of 6000 papers in \LaTeX format has been collected, spanning many areas of mathematics. The DLMF pages, not yet part of this dataset, will be used as well.
- Software for reading those papers, properly tokenizing the math terms as well as the English text, and converting the math and text tokens into feature vectors using Word2Vec, is nearly complete. Accordingly, the project is about to convert the vocabulary in the 6000-paper dataset into numerical vectors, and to begin the implementation and testing of the disambiguation algorithms illustrated in Sect. 6.
- \LaTeX ML and the POM tagger are in a ready state to be used to read the datasets, and provide a good amount of labeling of the math terms. Those labels, though often tentative, will serve two purposes. First, for a fraction of the dataset, we will manually look at those labels and correct them where needed to create a validly labeled reference dataset. Second, the documents with the tentative labels (specifically, alternative features) will serve as a testing set for disambiguation algorithms, as illustrated in Sect. 6.

References

1. Altman, N.S.: An introduction to kernel and nearest-neighbor nonparametric regression. *Am. Stat.* **46**(3), 175–185 (1992)
2. Bengio, Y.: Deep learning of representations: looking forward. In: Dediu, A.-H., Martín-Vide, C., Mitkov, R., Truthe, B. (eds.) *SLSP 2013. LNCS (LNAI)*, vol. 7978, pp. 1–37. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39593-2_1
3. Bordes, A., Glorot, X., Weston, J., Bengio, Y.: Joint learning of words and meaning representations for open-text semantic parsing. In: *AISTATS* (2012)
4. Bengio, Y., LeCun, Y., Hinton, G.: Deep learning. *Nature* **521**, 436–444 (2015)
5. Breiman, L.: Random forests. *Mach. Learn.* **45**(1), 5–32 (2001)
6. Carette, J., Farmer, W.M.: A review of mathematical knowledge management. In: Carette, J., Dixon, L., Coen, C.S., Watt, S.M. (eds.) *CICM 2009. LNCS (LNAI)*, vol. 5625, pp. 233–246. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02614-0_21
7. Chung, J., Gülçehre, Ç., Cho, K., Bengio, Y.: Gated feedback recurrent neural networks. In: *ICML* (2015)
8. Cho, K., Van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches. *ArXiv e-prints*, abs/1409.1259 (2014)
9. Cho, K., van Merriënboer, B., Gulcehre, C., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: *The Empirical Methods in Natural Language Processing (EMNLP 2014)* (2014)
10. Cortes, C., Vapnik, V.: Support-vector networks. *Mach. Learn.* **20**(3), 273–297 (1995)

11. Cohl, H.S., et al.: Semantic preserving bijective mappings of mathematical formulae between document preparation systems and computer algebra systems. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) C1CM 2017. LNCS (LNAI), vol. 10383, pp. 115–131. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_9
12. Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R., Makhoul, J.: Fast and robust neural network joint models for statistical machine translation. In: Proceedings of the ACL 2014 (2014)
13. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
14. Garcia, S., Derrac, J., Cano, J., Herrera, F.: Prototype selection for nearest neighbor classification: taxonomy and empirical study. IEEE Trans. Pattern Anal. Mach. Intell. **34**(3), 417–435 (2012)
15. Ginev, D., Jucovschi, C., Anca, S., Grigore, M., David, C., Kohlhase, M.: An architecture for linguistic and semantic analysis on the arXMLiv corpus. In: Applications of Semantic Technologies (AST) Workshop at Informatik (2009)
16. Gao, L., et al.: Preliminary exploration of formula embedding for mathematical information retrieval: can mathematical formulae be embedded like a natural language? [arXiv:1707.05154](https://arxiv.org/abs/1707.05154) (2017)
17. Graves, A.: Supervised Sequence Labelling with Recurrent Neural Networks. SCI. Springer, Heidelberg (2012). <https://doi.org/10.1007/978-3-642-24797-2>
18. Guidi, F., Sacerdoti Coen, C.: A survey on retrieval of mathematical knowledge. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) C1CM 2015. LNCS (LNAI), vol. 9150, pp. 296–315. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20615-8_20
19. Hastie, T.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd edn. Springer, New York (2013)
20. Lau, J.H., Baldwin, T.: An empirical evaluation of doc2vec with practical insights into document embedding generation. In: 1st Workshop on Representation Learning for NLP (2016)
21. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
22. Jurafsky, D., Martin, J.H.: Speech and Language Processing. Pearson Education, London (2009)
23. Kim, Y.: Convolutional neural networks for sentence classification. In: Conference on Empirical Methods in NLP, October 2014, Doha, Qatar, pp. 1746–1751 (2014)
24. Kohlhase, M.: Semantic markup for mathematical statements. v1.2 (2016)
25. Kottwitz, S.: *LATEX* Beginner’s Guide. PACKT Publishing, Birmingham (2001)
26. Kstovski, K., Blei, D.M.: Equation Embeddings, March 2018. <https://arxiv.org/abs/1803.09123>
27. LeCun, Y., Kavukcuoglu, K., Farabet, C.: Convolutional networks and applications in vision. In: 2010 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 253–256 (2010)
28. Lai, S., Liu, K., He, S., Zhao, J.: How to generate a good word embedding. IEEE Intell. Syst. **31**(6), 5–14 (2016)
29. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International Conference on Machine Learning, pp. 1188–1196, January 2014
30. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: International Conference on Learning Representations: Workshops Track (2013)

31. Miller, B.: $\mathcal{L}^{\text{TEXXML}}$: A \mathcal{L}^{TEX} to XML/HTML/MathML Converter. <http://dlmf.nist.gov/LaTeXXML/>
32. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS, pp. 3111–3119 (2013)
33. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. MIT Press, Cambridge (2012)
34. Malon, C.D., Uchida, S., Suzuki, M.: Mathematical symbol recognition with support vector machines. Pattern Recogn. Lett. **29**, 1326–1332 (2008)
35. Navigli, R.: Word sense disambiguation: a survey. ACM Comput. Surv. **41**(2), 1–69 (2009)
36. Neumaier, A., Schodl, P.: A framework for representing and processing arbitrary mathematics. In: International Conference on Knowledge Engineering and Ontology Development, pp. 476–479 (2010)
37. Nickel, M., Kiela, D.: Poincare embeddings for learning hierarchical representations. In: Advances in Neural Information Processing Systems (2017)
38. Nghiem, M.-Q., Yokoi, K., Matsubayashi, Y., Aizawa, A.: Mining coreference relations between formulas and text using Wikipedia. In: 2nd Workshop on NLP Challenges in the Information Explosion Era, Beijing, pp. 69–74 (2010)
39. Olver, F.W.J., Olde Daalhuis, A.B., Lozier, D.W., Schneider, B.I., Boisvert, R.F., Clark, C.W., Miller, B.R., Saunders, B.V., (eds.): NIST Digital Library of Mathematical Functions. <https://dlmf.nist.gov/>, Release 1.0.18 of 27 Mar 2018
40. Piotr, B., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Trans. Assoc. Comput. Linguist. **5**, 135–146 (2017)
41. Pennington, J., Socher, R., Manning, C.D.: GloVe: global vectors for word representation. In: The 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 25–29 October 2014, pp. 1532–1543 (2014)
42. Rudolph, M., Ruiz, F., Athey, S., Blei, D.M.: Structured embedding models for grouped data. In: NIPS, pp. 250–260 (2017)
43. Schoneberg, U., Sperber, W.: POS tagging and its applications for mathematics. In: CICM 2014, Coimbra, Portugal, pp. 213–223 (2014)
44. Schubotz, M., Grigorev, A., Leich, M., Cohl, H.S., Meuschke, N., Gipp, B., Youssef, A., Markl, V.: Semantification of identifiers in mathematics for better math information retrieval. In: The 39th Annual ACM SIGIR Conference (SIGIR 2016), Pisa, Italy, pp. 135–144, July 2016
45. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: NIPS (2014)
46. Watt, S.M.: Exploiting implicit mathematical semantics in conversion between TEX and MathML. TUGBoat **23**(1), 108 (2002)
47. Wolska, M., Grigore, M., Kohlhase, M.: Using discourse context to interpret object-denoting mathematical expressions. In: Towards Digital Mathematics Library, DML workshop, pp. 85–101. Masaryk University, Brno (2011)
48. Youssef, A.: Part-of-math tagging and applications. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) CICM 2017. LNCS (LNAI), vol. 10383, pp. 356–374. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_25

Author Index

- Ait-Sadoune, Idir 23
- Bancerek, Grzegorz 1
- Betzendahl, Jonas 7
- Boldo, Sylvie 87
- Brucker, Achim D. 23
- Carette, Jacques 76
- Cohl, Howard S. 39, 104
- Crisafulli, Paolo 23
- Czajka, Łukasz 53
- Dana-Picard, Thierry 60
- Ekici, Burak 53
- Farmer, William M. 76
- Gallois-Wong, Diane 87
- Gipp, Bela 104
- Greiner-Petter, André 39, 104
- Hilaire, Thibault 87
- Hölle, Stefan 125
- Humenberger, Andreas 111
- Immler, Fabian 178
- Jakubův, Jan 118
- Jaroschek, Maximilian 111
- Junk, Michael 125
- Kaliszyk, Cezary 53, 131, 255
- Kohlhase, Andrea 147
- Kohlhase, Michael 7, 147, 209, 232
- Köstler, Harald 232
- Kovács, Laura 111
- Kovács, Zoltán 60, 164
- Maletzky, Alexander 178
- Miller, Bruce R. 271
- Minimair, Manfred 194
- Müller, Dennis 209
- Nagashima, Yutaka 225
- Naumowicz, Adam 1
- Ouypornkochagorn, Taweechai 147
- Pąk, Karol 131
- Parsert, Julian 225
- Pollinger, Theresa 232
- Rabe, Florian 209
- Sahli, Sebastian 125
- Schreiner, Wolfgang 248
- Schubotz, Moritz 39, 104
- Sharoda, Yasmine 76
- Urban, Josef 1, 118, 255
- Wang, Qingxiang 255
- Wolff, Burkhard 23
- Youssef, Abdou 271