

Work-in-progress papers presented at the

**15th Conference on Intelligent
Computer Mathematics
(CICM 2022)**

INFORMAL PROCEEDINGS

Kevin Buzzard and Temur Kutsia
(editors)

August 2022

Preface

The 15th Conference on Intelligent Computer Mathematics (CICM 2022) was held during September 19–23, 2022, in Tbilisi, Georgia. CICM was part of the Computational Logic Autumn Summit (CLAS 2022), which took place between September 19–30, 2022, and was hosted by Ivane Javakhishvili Tbilisi State University.

CICM was initially formed in 2008 as a joint meeting of communities involved in computer algebra systems, theorem provers, and mathematical knowledge management, as well as those involved in a variety of aspects of scientific document archives. Since then, the conference has been held annually: Birmingham (UK, 2008), Grand Bend (Canada, 2009), Paris (France, 2010), Bertinoro (Italy, 2011), Bremen (Germany, 2012), Bath (UK, 2013), Coimbra (Portugal, 2014), Washington D.C. (USA, 2015), Białystok (Poland, 2016), Edinburgh (UK, 2017), Hagenberg (Austria, 2018), Prague (Czech Republic, 2019), Bertinoro (Italy, 2020, virtual), and Timișoara (Romania, 2021, virtual). CICM 2022 was organized in hybrid mode.

This informal proceedings contains eight work-in-progress papers presented at CICM 2022. Besides them, the full conference program consisted of three invited talks and 24 formal papers, which were published in volume 13467 of Springer Lecture Notes in Artificial Intelligence series.

The Program Committee of CICM 2022 worked hard to compose a very interesting and diverse program of the conference. This work was managed using the EasyChair system, which was very helpful and convenient. We thank Besik Dundua (conference chair), Mikheil Rukhaia, Ana Idadze, and their colleagues in Tbilisi for the successful organization of the conference, which was a difficult task especially because of the challenges posed by the hybrid mode. The Kurt Gödel Society and Matthias Baaz provided invaluable support in the organization. We are grateful to Serge Autexier for his publicity work, Mădălina Erașcu for serving as the doctoral program chair, and Florian Rabe for serving as the workshop chair. We also thank the authors of the submitted papers, the PC members and external reviewers, the workshop organizers, as well as the invited speakers and the participants of the conference.

August 2022

Kevin Buzzard
Temur Kutsia

Table of Contents

A Parallel Corpus for Natural Language Machine Translation to Isabelle . <i>Anthony Bordg, Yiannos Stathopoulos, and Lawrence Paulson</i>	1
Sophize Mathematics Library <i>Abhishek Chugh</i>	7
LOL: A Library Of Lemma templates for data-driven conjecturing <i>Sólrún Halla Einarsdóttir, Moa Johansson, and Nicholas Smallbone</i>	22
PISE – Proofscape Integrated Study Environment (CICM’22 System Entry) <i>Steve Kieffer</i>	28
Formalising the Krull Topology in Lean <i>Sebastian Monnet</i>	30
Three Case Studies on Realms <i>Florian Rabe and Franziska Weber</i>	46
Setting up Set-Theoretical Foundations in Naproche <i>Marcel Schütz, Adrian De Lon, and Peter Koepke</i>	52
OEIS Semantified – A Tetrapodal Dataset <i>Michael Wagner and Michael Kohlhase</i>	65

CICM 2022 Organization

Program Committee Chairs

Kevin Buzzard	Imperial College, London, UK
Temur Kutsia	RISC, Johannes Kepler University Linz, Austria

Program Committee

Jesús Aransay	Universidad de La Rioja, Spain
David Aspinall	The University of Edinburgh, UK
Serge Autexier	DFKI, Germany
Alexander Bentkamp	Vrije Universiteit Amsterdam, the Netherlands
Alex J. Best	Vrije Universiteit Amsterdam, the Netherlands
David Cerna	Czech Academy of Sciences, Institute of Computer Science, Czechia
Shaoshi Chen	KLMM, AMSS, Chinese Academy of Sciences, China
Sander Dahmen	Vrije Universiteit Amsterdam, the Netherlands
Mădălina Erăşcu	West University of Timisoara, Romania
William Farmer	McMaster University, Canada
Cezary Kaliszyk	University of Innsbruck, Austria
Fairouz Kamareddine	Heriot-Watt University, UK
Marie Kerjean	CNRS, LIPN, Université Sorbonne Paris Nord, France
Michael Kohlhase	FAU Erlangen-Nürnberg, Germany
Olexandr Kononov	University of St Andrews, UK
Angeliki Koutsoukou-Argyraiki	University of Cambridge, UK
Robert Y. Lewis	Brown University, USA
Heather Macbeth	Fordham University, USA
Barbara Morawska	Ahmedabad University, India
Daniele Nantes-Sobrinho	Universidade de Brasília, Brazil
Pedro Quaresma	University of Coimbra, Portugal
Florian Rabe	FAU Erlangen-Nürnberg, Germany
Claudio Sacerdoti Coen	University of Bologna, Italy
Christoph Schwarzweller	Gdansk University, Poland
Sofène Tahar	Concordia University, Canada
Olaf Teschke	FIZ Karlsruhe, Germany
Wolfgang Windsteiger	RISC, Johannes Kepler University Linz, Austria

Additional Reviewers

Aksoy, Kubra	Müller, Dennis
Asakura, Takuto	Nagashima, Yutaka
Baanen, Anne	Naumowicz, Adam
Barhoumi, Oumaima	O'Connor, Liam
Buran, Michal	Olarte, Carlos
Cohen, Cyril	Rashid, Adnan
de Lima, Thaynara Arielly	Starosta, Štěpán
Deniz, Elif	Stuckey, Peter J.
Eberl, Manuel	Thiemann, René
From, Asta Halkjær	Vélez, M. Pilar
Jakubův, Jan	Wong, Thomas
Li, Haokun	Yu, Wensheng
Miquey, Étienne	Zhan, Bohua

A Parallel Corpus for Natural Language Machine Translation to Isabelle^{*}

Anthony Bordg, Yiannos Stathopoulos, and Lawrence Paulson

Department of Computer Science and Technology, University of Cambridge,
Cambridge, UK
`{apdb3,yas23,lp15}@cam.ac.uk`

Abstract. We present the Isabelle Parallel Corpus (IPC), a parallel corpus of mathematical facts and their proofs expressed in both natural language and in Isabelle/HOL. Our corpus could be useful in many tasks related to machine learning in theorem proving, such as autoformalisation. We also discuss challenges and requirements associated with constructing the IPC, identified during a pilot study, that are distinct from parallel corpora for natural language. At the time of writing, the IPC contains over 500 facts (definitions, lemmata, theorems), with parallel proof scripts included for 18 of those entries. We envisage the IPC to be a living corpus since the optimal number of parallel proof scripts for autoformalisation is a major research question.

Keywords: Parallel Corpus · Automating Formalisation · Isabelle.

1 Introduction

Parallel corpora are key resources for machine translation in NLP. A parallel corpus maps textual scripts in one language (e.g. French) to their equivalents in another language (e.g. English). The paired scripts in a parallel corpus are data points used to train language models that learn how to translate text from one language to the other.

In theorem proving a parallel corpus can be useful in tasks such as *autoformalisation* (an instance of machine translation [6,7]), formal fact retrieval and mathematical knowledge discovery [8]. Large-scale transformer models, such as those used by Codex[2], can be trained on our corpus to attack many challenging tasks in theorem proving and NLP with mathematical text.

We introduce the *Isabelle Parallel Corpus* (IPC) of natural language and Isabelle proofs. Natural language proofs in our corpus are expressed using sentences in the natural language of mathematics, with mathematical expressions transcribed using \LaTeX . These textual proofs have been extracted from textbooks, International Olympiad of Mathematics solution sheets and other real-world mathematics resources.

^{*} This work was supported by the ERC Advanced Grant ALEXANDRIA (Project GA 742178). We thank all the members of the ALEXANDRIA project for their encouragement as well as Manuel Eberl, Sean Holden and Albert Jiang.

We believe that Isabelle is a suitable ITP target for a parallel corpus of the proposed nature for two reasons. First, Isabelle’s libraries and the Archive of Formal Proofs are large and cover a wide range of mathematical topics¹. As a result, there is a plethora of artefacts that can be sourced from these collections to build a parallel corpus. Second, proofs in Isabelle can be structured like proofs in mathematical texts.

2 Annotation Requirements and Pilot Experiment

Sentence and token alignments for parallel scripts are pairings that link sentences, tokens or groups of tokens in one language to those in the other language and are considered to be beneficial to machine translation models [4,3]. However, the nature of sentence and word alignments for a parallel corpus like the IPC is unclear.

We conducted a pilot study to determine sentence and word alignment requirements of our corpus and we present one case study to illustrate the intricacies of the task. Table 1 lists the sentences in the proof of Theorem 2.2 in [1]. In the table, individual sentences are numbered 1–13, with phrases and mathematical expressions of interest labelled with Roman numerals and Greek letters, respectively.

The corresponding Isabelle proof, taken from the *HOL-Number_Theory* library, is shown in Figure 1. The left column in the figure maps the Isabelle commands (right column) to the sentences, phrases and mathematical expressions found in the natural language proof (Table 1).

One notices minor discrepancies in notations, otherwise fairly close, *card*(A d) (*resp.* *totient*, $\{1 \dots n\}$) being abbreviated $f(d)$ (*resp.* φ , S) in the textbook. Moreover, one notes that some commands in the Isabelle proof script do not have direct counterparts in the textbook source. It is certainly true for commands 3 and 6 which state trivial facts, but also for command 11 which corresponds to a mere concatenation of the facts previously proved.

In the opposite way, some sentences in the textual proof do not have direct counterparts in the Isabelle script: sentence ⑧ is more or less hidden in the lemma *card_gcd_eq_totient* which is passed to Isabelle as a rule through the keyword *by* in row 5 of the script; in the same way, the explanation, introduced by “because”, for the equivalence mentioned in sentence ⑫ is hidden in the fact *reindex_bij_witness* passed to some tactics in row 10 of the script; last, sentence ⑥ states not only that the union of the sets $A(d)$ ’s is S , but also that this union is disjoint, this last fact is not explicitly stated in Isabelle, but is visibly handled automatically by the tactic *auto* as can be seen from the use of the lemma *card_UN_disjoint* in row 8 of the script.

From this example and others studied by the authors, we observed that there are sentences in the natural language that do not correspond to any statement in

¹ As of April 2022, the Archive of Formal Proofs contains 3,396,200 lines of code, making it one of the largest collections of formal material.

Sentence	Isabelle command
α	1 <code>lemma totient divisor sum: "($\sum d \mid d \text{ dvd } n$, totient d) = n"</code> <code>proof (cases "n = 0")</code>
I	2 <code>case False</code>
	3 <code>hence "n > 0" by simp</code>
④	4 <code>define A where "A = (λd. {$k \in \{0..n\}$. gcd k n = d})"</code>
β	5 <code>have *: "card (A d) = totient (n div d)" if d: "d dvd n" for d</code> <code>using <n > 0> and d unfolding A def by (rule card_gcd_eq_totient)</code>
	6 <code>have "n = card {1..n}" by simp</code>
② ⑥	7 <code>also have "{1..n} = ($\bigcup d \in \{d. d \text{ dvd } n\}$. A d)" by safe (auto simp: A_def)</code>
(1) in ⑦	8 <code>also have "card ... = ($\sum d \mid d \text{ dvd } n$, card (A d))"</code> <code>using <n > 0> by (intro card_UN_disjoint) (auto simp: A_def)</code>
γ	9 <code>also have "... = ($\sum d \mid d \text{ dvd } n$, totient (n div d))" by (intro sum.cong_refl *) auto</code>
⑫	10 <code>also have "... = ($\sum d \mid d \text{ dvd } n$, totient d)" using <n > 0></code> <code>by (intro sum.reindex_bij_witness[of " (div) n" " (div) n"]) (auto elim: dvdE)</code>
	11 <code>finally show ?thesis ..</code>
⑬	12 <code>qed auto</code>

Fig. 1. Isabelle commands (numbered 1 to 12) mapped onto their matching sentences, phrases and mathematical expressions in the textual theorem.

Isabelle and vice-versa. Furthermore, authors of textbooks may sometimes evoke results that have been established earlier but not explicitly stated as labelled lemmata. Isabelle users may repeat proofs of facts (or variants thereof) when (a) Isabelle code is not properly factored and (b) when facts are not proven at the right level of generality.

Our pilot highlighted three challenges in constructing the IPC. The **first challenge** is to identify the annotation requirements of the corpus for aligning natural language sentences to Isabelle statements. The **second challenge** is how should dependencies in parallel textual and Isabelle proofs be incorporated in the corpus. One solution would be to integrate dependencies in the corpus and include data about the reference graph between artefacts [8]. The **third challenge** is designing a suitable annotation scheme for (a) representing Isabelle terms and mathematical expressions in textual proofs (using MathML, for example) and (b) establishing an alignment between them.

3 The Parallel Corpus and its Construction

To construct our corpus, we adopted a multi-phase approach. In the first phase, we sourced over 500 Isabelle artefacts, including definitions, lemmata, theorems and proof scripts. For each artefact we record information that includes its statement typeset in \LaTeX , its page and number as they appear in the source material as well as a \BibTeX citation. The 564 items recorded in phase 1 cover a wide range of mathematical topics: puzzles from mathematical Olympiads, analytic number theory, abstract algebra and geometry. The second phase, which is ongoing, involves pairing informal and formal proofs attached to the recorded state-

Sentence	Text
①	<p>If $\underbrace{n \geq 1}_1$ we have</p> $\sum_{d n} \overbrace{\varphi(d)}^{\alpha} = n.$
②	Let S denote the set $\{1, 2, \dots, n\}$.
③	We distribute the integers of S into disjoint sets as follows.
④	<p>For each divisor d of n, let</p> $A(d) = \{k : (k, n) = d, 1 \leq k \leq n\}.$
⑤	That is, $A(d)$ contains those elements of S which have the gcd d with n .
⑥	The sets $A(d)$ form a disjoint collection whose union is S .
⑦	<p>Therefore if $f(d)$ denotes the number of integers in $A(d)$ we have</p> $\sum_{d n} f(d) = n. \quad (1)$
⑧	But $(k, n) = d$ if and only if $(k/d, n/d) = 1$, and $0 < k \leq n$ if and only if $0 < k/d \leq n/d$.
⑨	Therefore, if we let $q = k/d$, there is a one-to-one correspondence between the elements in $A(d)$ and those integers q satisfying $0 < q \leq n/d$, $(q, n/d) = 1$.
⑩	The number of such q is $\varphi(n/d)$.
⑪	<p>Hence $\underbrace{f(d) = \varphi(n/d)}_{\beta}$ and (1) becomes</p> $\sum_{d n} \underbrace{\varphi(n/d)}_{\gamma} = n.$
⑫	But this is equivalent to the statement $\sum_{d n} \varphi(d) = n$ because when d runs through all divisors of n so does n/d .
⑬	This completes the proof.

Table 1. Natural language statement and proof of Theorem 2.2 in [1].

ments. At the time of writing, we have paired Isabelle proofs with corresponding informal proofs for 18 artefacts.

We developed an annotation tool that allows us to (a) record information about artefacts in the corpus, (b) collect parallel natural language and Isabelle proof scripts and (c) implement the annotation scheme for the IPC. Our tool is built on top of the *SErAPIS* search engine for Isabelle [5], inheriting the engine’s user interface for fact search and presentation of results. Annotators can add data to the corpus by searching for an artefact of interest and clicking the appropriate button next to each search result, which will prompt a data input form.

Figure 2 illustrates how existing annotations are presented with our tool: phase 1 information is presented as a collapsible information card. The dark grey buttons next to the artefact name allow annotators to edit or delete phase 1 data. The blue and orange buttons allow annotators to add or edit textual and Isabelle proofs, respectively (phase 2 data).

544 Factname: last_fact_of_the_theory_28_nameless_theorem Session: IMO2019 Theory: IMO2019_Q1 Locale: Kind: theorem [manual] Edit Delete Text proof Isabelle proof

Title of Natural Language Source
IMO 2019 Problems (with solutions)
Source kind
other
LaTeX
Consider a function $f: \mathbb{Z} \rightarrow \mathbb{Z}$ that fulfils the functional equation $f(2a) + 2f(b) = f(f(a+b))$ for all $a, b \in \mathbb{Z}$. Then f is either identically 0 or of the form $f(x) = 2x + c$ for some constant $c \in \mathbb{Z}$.
BibTeX
URL of Source
https://www.imo2019.uk/wp-content/uploads/2018/07/solutions-r856.pdf
Notes
Problem 1 p.4

Fig. 2. A problem from the 2019 International Mathematical Olympiads.

4 Conclusions and Future Work


The IPC will be made public on GitHub prior to our presentation in the hope that it will be useful to researchers in machine learning for theorem proving. We intend to continuously update the corpus (*e.g.* with sentence and token alignments in phase 3). This release strategy reflects our vision that the IPC is a *living* corpus with well-defined milestone releases to facilitate comparative analysis of machine learning models. We also intend to make our annotation tools public to continuously expand the IPC.

References

1. Apostol, T.: Introduction to Analytic Number Theory. Undergraduate Texts in Mathematics, Springer New York (2013)
2. Chen, M. et al.: Evaluating large language models trained on code. CoRR abs/2107.03374 (2021), <https://arxiv.org/abs/2107.03374>
3. Jurafsky, D., Martin, J.H.: Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition. Pearson Prentice Hall, Upper Saddle River, N.J. (2009)
4. Koehn, P.: Statistical Machine Translation. Cambridge University Press, USA, 1st edn. (2010)
5. Stathopoulos, Y., Koutsoukou-Argyraki, A., Paulson, L.C.: SErAPIS: A concept-oriented search engine for the Isabelle libraries based on natural language. In: Online proceedings of the Isabelle Workshop 2020 affiliated to IJCAR 2020 (2020), https://files.skets.net/Isabelle_Workshop_2020/Isabelle_2020_paper_4.pdf

6. Szegedy, C.: A promising path towards autoformalization and general artificial intelligence. In: Benzmüller, C., Miller, B.R. (eds.) *Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy, July 26-31, 2020, Proceedings*. Lecture Notes in Computer Science, vol. 12236, pp. 3–20. Springer (2020). https://doi.org/10.1007/978-3-030-53518-6_1
7. Wang, Q., Brown, C.E., Kaliszyk, C., Urban, J.: Exploration of neural machine translation in autoformalization of mathematics in Mizar. In: Blanchette, J., Hritcu, C. (eds.) *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*. pp. 85–98. ACM (2020). <https://doi.org/10.1145/3372885.3373827>
8. Welleck, S., Liu, J., Bras, R.L., Hajishirzi, H., Choi, Y., Cho, K.: Naturalproofs: Mathematical theorem proving in natural language. CoRR abs/2104.01112 (2021), <https://arxiv.org/abs/2104.01112>

Sophize Mathematics Library

Abhishek Chugh 

Sophize Foundation, Bengaluru - 560066, India
`abc@sophize.org`

Abstract. Sophize is a novel mathematics library and discussion platform with a mission to help our users find and organize mathematical proofs. We present Sophize’s novel yet familiar knowledge organization scheme that is used to represent a wide variety of proofs. With this scheme at its core, we have engineered a platform to logically aggregate knowledge from multiple sources. The platform curates knowledge from published research, encyclopaedias such as PlanetMath and Wikipedia, informal computer programs, and formal systems such as Metamath; and allows its users to run a federated search over them. To make this happen, two developments were necessary. First, we came up with the concept of ‘proof-generating machines’ and developed an open plugin architecture that allows proofs from computer programs to be generated as required. Second, we extended the Markdown language to represent the connections between mathematical objects found across various sources of knowledge. In addition, we also utilized this new language to create a novel communication system built specifically to aid mathematicians in solving problems collaboratively.

Keywords: Sophize · knowledge organization · proof-generating machine · belief set · metamath · markdown · argument graph · proof graph · semantic data

1 Introduction

1.1 The Sophize Platform

Sophize is a novel mathematics library and discussion platform. The author has developed the platform over the course of the last few years at <https://sophize.org>. Sophize’s primary mission is to help users find existing proofs of mathematical statements, to discover new proofs, and to utilize this knowledge in their work. We combine knowledge from multiple resources and have accumulated thousands of definitions, theorems, and proofs from a wide variety of sources, including some published research, the PlanetMath encyclopedia, Wikipedia, and the Metamath formal system [4].

1.2 Motivation

In the past couple of decades, the Internet has revolutionized our life by giving us access to seemingly unlimited information. Information from a wide variety of

topics ranging from human medicine to rocket science is now just a few keystrokes away. However, it seems that we are having even more trouble making sense of the World around us. In fact, dealing with the vast amounts of information thrown at us seems to be one of the most important problems of our generation. While we have access to a wide range of opinions on almost every topic imaginable, it has become increasingly difficult to decide which ones to trust. It seems that the Internet can find us "proofs" for anything we would like to believe.

Thus, the author's primary motivation for this project is to create a library that not just lays out all the information about the World but organizes it in a way that helps us make better sense of it. A library that helps us access justifications from a variety of viewpoints and allows us to collaboratively and objectively evaluate issues in the justifications or the viewpoints they arise from. Clearly, this is a challenging goal.

We begin our journey with a focus on pure Mathematics, where knowledge is objectively less complex as compared to other fields. In mathematics, only logically sound arguments are acceptable as justifications. These arguments develop in different foundational theories; and logical consistency is the primary criterion of correctness. Thus, as a Mathematics library, Sophize's goal is to help our users access mathematical proofs (arguments) from a variety of belief sets (roughly - foundational theories along with indicators of trust in knowledge sources) and allow our users to collaboratively and objectively evaluate issues in the arguments or the belief sets they developed in.

With our current scheme, we can incorporate nearly all mathematics. A natural extension would be to model observational data and develop criteria to quantify empirical correspondence in order to have the ability to incorporate scientific knowledge. But, for now, we are focused on developing a state-of-the-art library for the Mathematics community.

1.3 Proofs in Sophize

Mathematical proofs are based on a variety of foundations such as ZFC, intuitionistic logic, and type theory. The arguments used in any proof are considered valid or not based on criteria that can vary. Most academic mathematics is peer-reviewed and published, but some mathematical proofs can be found in community curated sources such as Wikipedia. Proofs can also be algorithmically generated, and at the highest level of verification, they are represented and verified using a formal system.

Sophize combines such an expansive range of proofs into a dense graph of propositions and logical arguments that aggregates knowledge from several documents and other data sources. We use this graph and combine it with the set of foundations and verifications chosen by each user to create proofs tailored to their needs. This introductory video gives an overview of the platform's offerings: <https://youtu.be/Wb1JbW9Otek>.

This work can also be seen as a step towards formalizing the network of information that exists in the connections of mathematical objects. The committee on planning a global library of the mathematical sciences recognized that

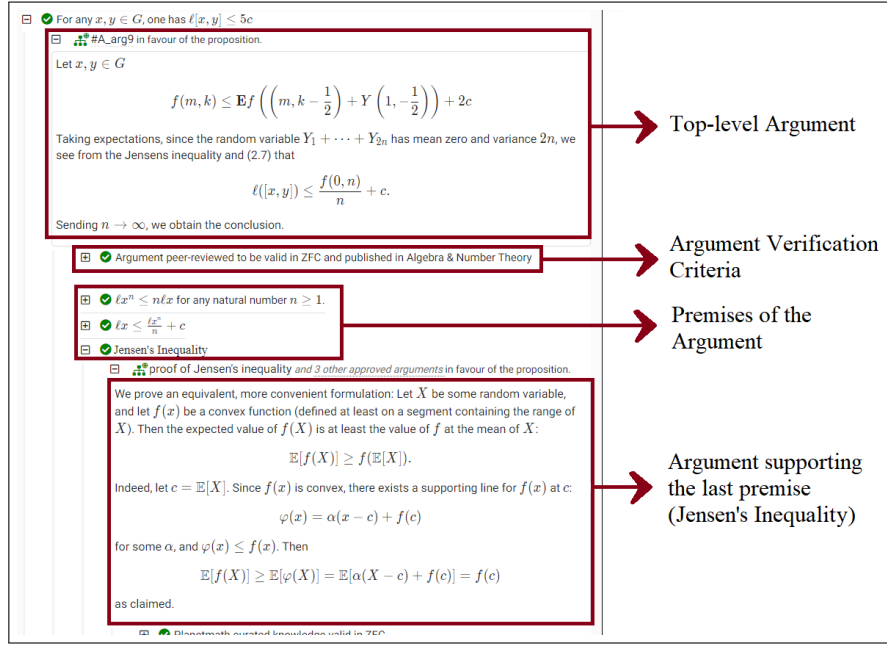


Fig. 1. A proof displayed as a graph (tree) of arguments and propositions on the Sophize platform.

this network is largely unexplored, and formalizing it has tremendous potential to accelerate math research [5]. Hence, we required novel techniques to connect mathematical entities from such a wide range of formal and informal knowledge sources.

1.4 Contribution

In this paper, we present the novel concepts and features of Sophize Mathematics Library. First, we expound Sophize's scheme to build a trust-focused logical network of mathematical knowledge. In section 3, we describe Sophize's proof-generating machines which are computer programs that generate proofs that integrate with Sophize's knowledge graphs.

We then present some features of Sophize Markdown, an extension of the Markdown language that helps present Sophize's knowledge graphs to its users. It is convenient enough to be used for casual discussions of mathematical ideas over the web. It is also powerful enough to embed mathematical entities such as definitions, theorems, and proofs from various sources, including formal systems. It thus plays an integral role in the two problems that we have mentioned above.

Section 5 discusses some other challenges and contributions made towards making Sophize a modern mathematics library. Finally, Section 6 concludes the paper.

2 Sophize’s Knowledge Organization Scheme

At its core, Sophize’s approach is to simply keep track of all arguments in favour or against any proposition. The use of some form of arguments is the common factor amongst all mathematical (and perhaps most scientific) knowledge. We use this fact to unify knowledge from multiple sources in a meaningful way as described below.

We use the set of all known arguments to track knowledge from a variety of foundational theories. For theoretical knowledge, i.e., the knowledge that doesn’t involve empirical data, the primary criterion of validity is internal (logical) consistency. Thus, for each foundational theory (‘belief set’ described below), we also keep track of any contradictions that arise from those foundations.

2.1 Core Concepts

Sophize uses the following concepts to logically organize theoretical knowledge. The datamodel for these concepts is published in JSON schema [6].

*A **resource** is an abstract concept inherited by all other top-level concepts like terms, propositions, arguments, and belief sets. Each resource has a URI and contains fields such as search tags and citations.*

A **URI** consists of two parts: a namespace-like identifier called its dataset-id, which indicates the data source, and a resource-id that specifies the resource type and its unique name in the data source. The dataset-id may be omitted if it can be inferred from the surrounding context.

For example, the Pythagorean theorem (a **Proposition**) represented in the Metamath project may have the URI *metamath/P_pythagorean* and the definition of cone (a **Term**) extracted from Wikipedia may have the URI *wiki/T_cone*. When used inside another resource in the same (‘wiki’) dataset, cone’s definition can be referred to simply as *T_cone*.

The same URI scheme will be used in this paper to refer to various terms, propositions, arguments, etc.

*A **term** is a clearly defined entity that can be used to make up a valid proposition. It can be a mathematical object, operator, symbol, data structure, algorithm, or even a person. ‘Meaningless’ primitives in formal theories are also categorized as terms.*

*A **proposition** is a grammatically valid statement that can be either true or false. Axioms, theorems, conjectures, hypotheses, lemmas, corollaries, and converses are all classified as propositions.*

*An **argument** is a set of propositions called premises along with a concluding proposition that is claimed to follow from the premises. In addition, most arguments include supporting text that explains how the conclusion follows from the premises.*

A **belief set** is roughly the set of axiomatic propositions that make up the foundations of a theory. Belief sets will be described in subsection 2.5.

Loosely defined, a **proof-generating machine** (PGM) is a computer program that generates proofs for an input proposition on-the-fly. PGMs are described in detail in section 3.

2.2 Argument Graph

An argument can be seen as a simple graph (see Fig. 2) with two types of nodes - (a) a single node representing the argument itself and (b) a set of proposition nodes representing the argument's conclusion and premises. The edges of this graph are directed and go from (a) premise nodes to argument node and (b) argument node to conclusion node. A set of arguments can thus be represented as a single graph that we call an **argument graph**.

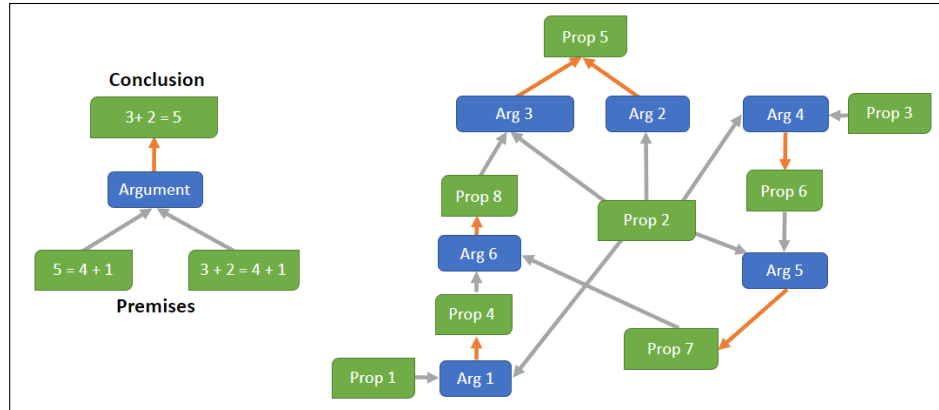


Fig. 2. An argument represented as a graph (left). A set of arguments forming an argument graph (right).

2.3 Generating Proof Graphs

A **proof graph** is a directed acyclic graph and contains complete proofs (including all alternate proofs) for all its propositions. A proof graph can be generated from an argument graph by starting with the axiom nodes and finding all nodes that can be reached from them (See Fig. 3).

Note that an argument graph combines with different sets of axioms to generate different proof graphs. The same argument may be utilized in multiple proof graphs (Fig. 5).

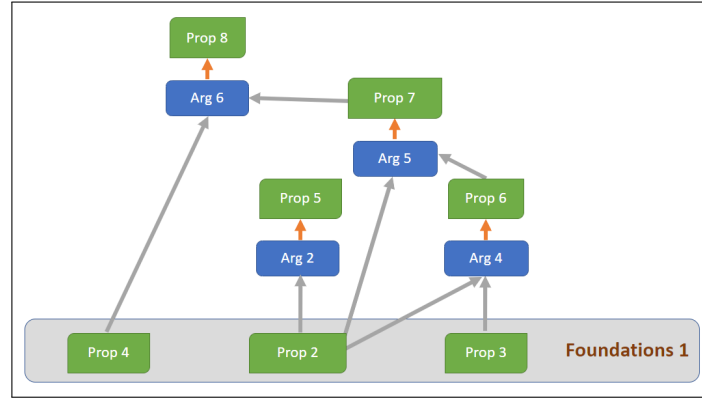


Fig. 3. Proof graph from argument graph (Fig. 2) using ‘Prop 2’, ‘Prop 3’, Prop 4’ as axioms.

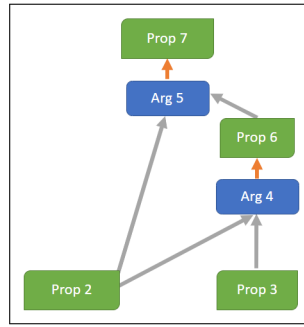


Fig. 4. Proof of ‘Prop 7’ from proof graph in Fig. 3 uses ‘Arg 4’ and ‘Arg 5’.

2.4 Validity of Arguments

The correctness of proofs obtained by the above process is contingent on the validity of the arguments in the argument graph. The validity of an argument is, in principle, independent of other arguments and objectively verifiable. Formally defined logistic systems indeed have a mechanical procedure to independently verify the validity of arguments.

However, only a small section of Mathematics is developed using formally verified arguments. The vast majority of mathematics is verified via peer review, and some of this knowledge is curated by experts in books, encyclopaedias. The claim of validity of proofs (arguments) is judged based on various parameters such as the academic standing of the claimant, the reputation of the peer reviewing journal, the history of errors in the book or encyclopaedia containing them, etc. Clearly, these criteria are subjective and each individual’s criteria for accepting the validity of an argument can vary to some extent.

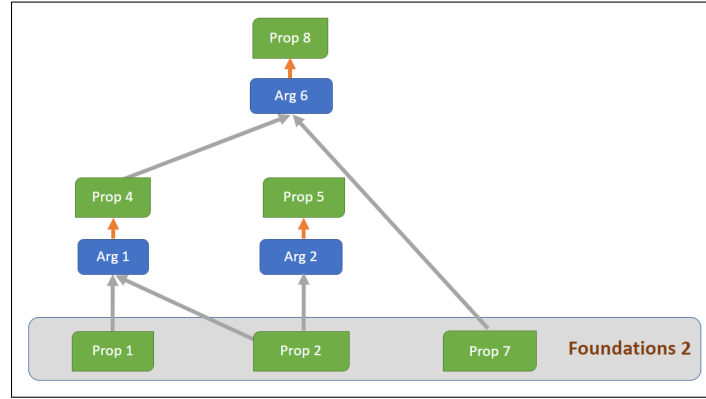


Fig. 5. Proof graph from the same argument graph (Fig. 2) using ‘Prop 1’, ‘Prop 2’, and Prop 7’ as axioms.

Sophize doesn’t promote or endorse any criteria. Instead, it allows its users to choose the criteria they believe in. The validity criterion is modelled as a proposition and is attached to arguments as a premise.

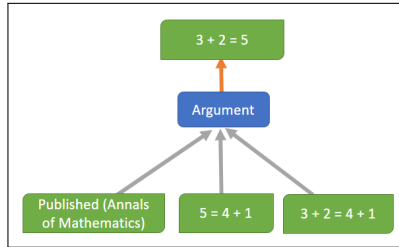


Fig. 6. Validity criteria as an argument premise.

To easily manage validity criteria, the relationships between validity criteria are also maintained in the argument graph using arguments such as "Published in journal with SJR score > 0.67 " \rightarrow "Published in Annals of Mathematics" (validation criterion should be provided for such arguments too). By maintaining such relationships, Sophize allows its users to conveniently choose multiple sources they consider to be reliable with a single manually chosen criterion.

2.5 Belief Sets

A proposition is true if it is the conclusion of a valid argument and the premises of the argument are also true. Of course, this means that we need to start with

some propositions that are assumed to be true without further justification. Traditionally, these are mathematical axioms and are considered to be self-evidently true. In mathematical inquiry, theorems are proven/disproven in one or more theories, each with its own set of axioms. Each theory may have its theoretical advantages and practical uses and there is no universal basis to choose one theory over another when trying to assert a proposition's truth/falsity.

Sophize allows any set of propositions to be considered true without evidence. Such a set of propositions is called a **belief set**. Propositions representing argument validity criteria are also part of belief sets. All propositions that are considered to be true in a belief set without proof are called its 'unsupported propositions'. In addition, a belief set can also contain a number of 'proof-generating machines' (e.g. for managing axiom-schemas) that will be discussed later. For ease of use, belief sets can also contain other belief sets.

The truth-value of a proposition is only defined within a belief set - there is no notion of absolute or universal truth on the Sophize platform. Sophize maintains proof graphs for all its belief sets.

2.6 Tracking Inconsistencies

If a proposition and its negation is proved within a belief set, it is inconsistent. As noted by the principle of explosion, any proposition can be proved from such a contradiction, thus making the claim of truth of any proposition practically worthless.

Thus, it is important to track and report contradictions in a belief set. To do that Sophize has a semantic notion of negation of a proposition. The premises or conclusion of an argument can be negations of propositions. When a proposition and its negation are proven, the system reports such contradictions for every belief set. When generating proof graphs from the argument graph, a contradicting proposition cannot be used a premise for any argument.

2.7 Potential Advantages

Apart from organizing knowledge focused on trust and aggregating disparate data sources, logical organization of mathematics can lead to many other advantages. We list some interesting ones below.

This knowledge organization scheme facilitates the re-use of propositions, and arguments across different theories. For example, to organize knowledge of different constant-curve geometries, we would create 4 belief sets - one each for absolute, Euclidean, hyperbolic and elliptic geometries. The latter three belief sets would comprise the absolute geometry belief set and the appropriate version of the parallel postulate needed for them. In such a scheme all the arguments (proofs) added to the argument graph for absolute geometry would automatically be shared by the other three geometries. The other three geometries need to only add the arguments that utilize their version of the parallel postulate.

Another use case would be to understand the consequences of important conjectures like the Riemann hypothesis by creating two belief sets - one where the

conjecture is true and one where it is false. The proof graphs would systematically store such conjectural knowledge which often gets lost especially when results from multiple conjectures are involved. Complexity theory is one such field where this need is quite apparent. Such conjectural knowledge can have practical applications too. For example, a lot of our information security systems utilize the knowledge developed with the conjectures $P \neq NP$.

Proof graphs are currently organized as trees as seen in Fig 1. Users can keep exploring the proof till they reach a point where the propositions are already familiar to them. A future application of proof graphs would be to generate narrative proofs tailored to the reader's current understanding. This could be achieved in a classroom setting where the current level of mathematical understanding of students is known or assumed. It may also be possible to infer this from the user's activity on the platform.

2.8 Limitations

The scheme described in this section has a semantic notion of negation of a proposition to track contradictions. Thus it is incapable of organizing knowledge that depends on more complex calculi such as many-valued logics.

3 Proof-Generating Machines (PGM)

A lot of mathematical knowledge is computed. Most computation machines (e.g. calculators, algebras) can be seen as performing the following tasks

- Parsing the input into a language that can represent math formulae.
- Finding and outputting an equivalent form of the formula.

Thus, their focus is not on finding the proof but only finding the equivalent simplified representation of the input formula (Although many computer algebras do show the steps of the computation). With Sophize, the focus instead is to generate proofs of the input propositions. A **proof-generating machine** (PGM) is a computer program that generates a partial proof graph against or in favour of an argument. PGMs can also have the capability to perform tasks 1 and 2, mentioned above.

PGMs take in some text as input which is parsed by the each PGM as they wish. Typically the text is treated either as a proposition (e.g. ' $3 + 5 = 8$ ') or a formula (eg. ' $5 + 6$ ') and in the latter case the PGM is responsible for finding an appropriate proposition (eg. ' $5 + 6 = 11$ ') for the given input. The PGM then generates a proof graph for this proposition and returns back to the user.

The Sophize platform itself merely forwards the request to the associated PGM and performs some checks on the output returned. Any http server that can process a proof request can be easily plugged in to Sophize platform using its HTTP address. One server can implement multiple PGMs. An overview of PGMs generating Metamath proofs is here: <https://youtu.be/hJtEI03ioLM>

3.1 ‘Active’ PGMs

As with other parts of Sophize, the validity of the output of a PGM is not taken for granted. Each PGM must specify all the propositions (premises) that it will use in the root nodes of the proof graph it returns. In addition, a PGM may also delegate parts of proofs to another PGM. These PGMs must also be specified. Similar to argument validation criterion, each PGM should also have premise indicating PGM’s correctness criteria (e.g. tested by X, verified by some committee, another program etc.)

A PGM is considered ‘active’ in a belief set iff:

1. all of its premises are true in the belief set.
2. all the PGMs it delegates-to are active in the belief set.

The output of a PGM is not used in belief sets in which it is not active.

3.2 PGM Response

The response of a request to any PGM has the following components:

- Truth value, i.e., whether the PGM considers the input proposition (provided or computed) true or false. PGM can also indicate that it doesn’t understand the input or can’t prove/disprove the proposition.
- Proof graph (or a subset of proof graph) generated by the PGM in favour or against the input proposition. This component is skipped if only computation results are requested.

Note that unlike other propositions and arguments on Sophize, these are not stored on disk. They have a temporary URI specified in a slightly different format. The leaves of the DAG can have two kinds of arguments:

1. Arguments whose the premises are limited to the premises of the PGM.
2. Special ‘Arguments’ that indicate that the generation of the proof of this argument’s conclusion must be delegated to another PGM. Seeing such arguments, the Sophize platform requests the remainder of the proof from the indicated PGMs as needed to allow users to browse the proof uninterrupted. To paginate large proofs, a PGM may also delegate to itself after generating some (say 100) arguments.

3.3 Materializing results

Sophize allows saving the fact that a machine has returned a valid proof for a proposition (not the actual proof itself which can be quite large). Then this proposition can be used in further proofs without the need to invoke the proof generation every time. The proof, of course, can be requested by the users whenever required.

4 Sophize Markdown

In the author's view, it is crucial to maintain the proofs graphs for building trustable knowledge. However, the presentation of proofs as graphs to users is perhaps not very convenient. Traditional text-based narration remains indispensable when presenting knowledge to the reader.

We have extended the Markdown language - a lightweight markup language for creating formatted text using a plain-text editor - for building a rich narration interface. It allows users to quickly encode and present semantic data, to show the truth status of propositions available in prose, and to effortlessly access proof graphs.

With Sophize Markdown, we can easily add $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ by enclosing it between two \$ signs (or \$\$ for display math). A detailed specification of Sophize Markdown is available [1] and many of its features are demonstrated at <https://youtu.be/5UYOpQwcjCk>. We highlight relevant features here:

4.1 Embedding Semantic Data

With Sophize Markdown we can easily link resources such as terms, propositions, and arguments. Links can be added using the pound sign (#) and the resource's URI. There are multiple resource link options that can be used for changing the behaviour and text of the resource link. The following example utilizes some of the options.

Sample Markdown Code

```
# Conic section

A conic section is a curve obtained as the intersection of the
#(wiki/T_conical_surface, 'surface') of a #wiki/T_cone with a
#wiki/T_plane. There are three types of conic sections.

## Ellipse
#oxford/T_ellipse|EXPAND #wiki/P_ellipse_area|EXPAND

## Parabola
...
```

The above code is rendered as:

Conic Section

A conic section is a curve obtained as the intersection of the surface of a cone with a plane. There are three types of conic sections.

Ellipse

An ellipse is a regular oval shape, traced by a point moving in a plane so that the sum of its distances from two other points is constant. The area $A_{ellipse}$ enclosed by an ellipse is

$$A_{ellipse} = \pi ab$$

where a and b are the lengths of the semi-major and semi-minor axes, respectively.

Parabola

...

In the above example, the ‘EXPAND’ option expanded the definition of the term, and statement of the proposition in place. Thus we were able to easily reuse concepts and theorems already extracted from other sources (say, Wikipedia and the Oxford dictionary in this case). Clicking on the underlined concepts pops-up a modal dialog with the definition of the term clicked.

4.2 Truth Status

When a belief set is chosen, Sophize markdown adds an icon next to each proposition link. This icon indicates whether the proposition has been proven or disproven (or both) in the currently selected belief set using an appropriate symbol such as a check-mark or a cross. Clicking on the icon brings up the proof graph as shown in Fig. 1. If the users switches from one belief set to another, the icon is updated based on the truth values in the new belief set but other parts of the article are unchanged. This functionality is demonstrated in the demo in subsection 1.3. Similarly, an icon is added next to a proof generating machine which indicates whether the PGM is active or not.

4.3 Formal Language Support

In the case of formal languages, when propositions can be fully parsed, Sophize can automatically attach links to appropriate resources. This provides a convenient interface, where the user types in the native language, and the final output automatically allows users to explore all concepts that make up the input statement in depth. Currently, this is demonstrated in the use of Sophize Markdown with the Metamath language.

5 Building a Modern Mathematics Library

This sections discusses a some practical issues need to be overcome to build a modern scalable library with reliable information.

5.1 Semantic Data Extraction

Utilizing the full range of features offered by the Sophize platform requires the availability of semantic information. However, extracting semantic information from unstructured or semi-structured literature is a difficult problem. This problem can be seen as composed of several sub-problems:

- Finding definitions and propositions in prose.
- Associating the appropriate entity with a term used in prose for quick lookup.
- Identifying arguments along with premises and conclusion.
- Identifying duplicate definitions and propositions used in different sources.

Solving these problems would require manual effort (e.g. crowd-sourcing) combined with automated extraction tools. Recent progress in machine learning and natural language processing techniques [2] does make this problem seem tractable.

5.2 Data Management

Sophize divides its data into groups called datasets for copyright and data access management. For example, the ‘wiki’ dataset that extracts knowledge from Wikipedia mathematics glossaries releases its data under ‘CC-BY-SA’ license, whereas the ‘metamath’ dataset is public (‘CC0’). Our signed-in users are free to create their own datasets and we allow them to choose who can read, write or comment in these datasets. Another interesting access control is the restriction of use of resources (especially argument-validity criteria) from an otherwise open-to-read dataset.

Information can be added to Sophize platform using intuitive web forms. However, it is infeasible to extract information from large repositories in this manner. Sophize publishes its data schema (explained in Section 2.1) and helper libraries in multiple languages. The extracted information is can thus be stored as text (JSON) files in a repository. Sophize provides dataset owners the ability to sync knowledge such repositories to the central Sophize database. The datasets ‘wiki’ and ‘planetmath’ use this scheme for keeping the script-generated data synced with Sophize’s central database.

5.3 Search Interface

Sophize allows its users to run a federated search over its resources using an Elasticsearch server. We even allow users to \LaTeX to their search queries. The server runs search a space-insensitive and case-insensitive search over \LaTeX formulae by encoding \LaTeX formulae as lexemes and an Elasticsearch analyzer processes them appropriately. (See section 6 for sources)

5.4 Collaboration

The Sophize platform includes an innovative mathematics communication interface designed to help our users discuss mathematics and collaboratively discover new proofs. The design focuses on making the collaboration more productive and enjoyable by building the right set of technical tools that aid in effective organization and summarization of existing progress. With Sophize Markdown at its core, Sophize’s collaboration interface[1] (demo: <https://youtu.be/d3gaalJ7UQM>) is able to provide key features such as easy linking of existing math content, \LaTeX support, and live preview of comment drafts.

6 Conclusion And Future Work

We have presented the Sophize platform, a modern open mathematics library focused on providing reliable and detailed knowledge to its users. We extract and aggregate logical arguments from a wide variety of sources including published research, encyclopedias, formal theorem provers, and formal/informal programs. While there exist other systems that work with disparate data sources, we are aware of no other systems that attempt to logically aggregate knowledge across these data sources. Using this organization scheme we can maintain proofs of results from different theoretical foundations and tailor them to a user’s knowledge reliability criterion.

We show how proof-generating machines can be used to instantaneously compute proofs that embed into previously computed graphs. We showcase how Sophize Markdown allow users to effortlessly browse proof-graphs and lookup entities in math literature. While there are other ways to represent semantic knowledge, such as content MathML and OpenMath, their scope is somewhat limited to specifying the meaning of the mathematical formula. The sTeX system [3] is perhaps most similar to Sophize Markdown but it needs creation of \LaTeX documents and use of \LaTeX XML systems. This makes its use in real-time web workflows impractical.

In its current state, Sophize has limited information to share with its users. Thus, we would first like to develop automated or semi-automated techniques to extract semantic knowledge from literature and expand the knowledge base that we share with our community. Using the proof-graphs and the semantic information collected we would like to make it easier for users to find relevant content with better search results.

Sources

The source code for several aspects of the Sophize platform (include Sophize Markdown) is at available at <https://github.com/Sophize>. The math encoder and Elasticsearch analyzer mentioned in section 5.3 are available at <https://gist.github.com/abc-sophize>.

Acknowledgements. A part of the research presented here was supported by the International Mathematical Union.

References

1. Chugh, A.: Sophize markdown and collaboration interface. EasyChair Preprint no. 6207 (2021), <https://easychair.org/publications/preprint/G3sC>
2. Ginev, D., Miller, B.R.: Scientific statement classification over arXiv.org. CoRR abs/1908.10993 (2019), <http://arxiv.org/abs/1908.10993>
3. Kohlhase, M.: Using LaTeX as a semantic markup format. Mathematics in Computer Science **2**(2), 279–304 (Dec 2008). <https://doi.org/10.1007/s11786-008-0055-5>
4. Megill, N.D., Wheeler, D.A.: Metamath: A Computer Language for Mathematical Proofs. Lulu Press, Morrisville, North Carolina (2019), <http://us.metamath.org/downloads/metamath.pdf>
5. National Research Council: Developing a 21st Century Global Library for Mathematics Research. The National Academies Press, Washington, DC (2014). <https://doi.org/10.17226/18619>
6. Sophize datamodel, <https://github.com/Sophize/datamodel-json>

LOL: A Library Of Lemma templates for data-driven conjecturing

Sólrun Halla Einarisdóttir, Moa Johansson, and Nicholas Smallbone

Chalmers University of Technology, Gothenburg, Sweden.
{slrn, moa.johansson, nicsma}@chalmers.se

Abstract. This paper describes a dataset of lemmas and their structural *templates* extracted from Isabelle’s Archive of Formal Proofs¹ (AFP). A template is an expression describing the shape of a given lemma. We hypothesise that many lemmas and theorems have similar shapes, and that knowledge of common shapes and their frequencies can be useful for downstream tasks such as automated conjecturing. Our dataset currently contains 22767 lemmas of 6567 distinct shapes.

Keywords: Theory exploration · Automated conjecturing · Theorem proving.

1 Introduction

Theory exploration is a method for automatically inventing interesting conjectures in a given formalised theory. While our previous system QuickSpec [5], can be very efficient on moderately sized theories, its runtime eventually hits exponential growth. We therefore introduced RoughSpec [2], to handle these cases. RoughSpec takes *templates* as an additional input, specifying the shapes of conjectures the user is interested in, and thus limiting the search. For example, by giving the template $F\ X\ (G\ Y\ Z) = G\ (F\ X\ Y)\ (F\ X\ Z)$, the user can instruct RoughSpec to search for distributivity properties.

In order to gain a more robust empirical understanding of what kinds of templates are useful and to provide a dataset for data-driven experiments we have mined such templates from the Archive of Formal Proofs (AFP). In this initial version, we have focused on equational lemmas, but this will be extended in the future. The AFP currently contains 676 entries from 425 authors, containing almost 200,000 lemmas and more than 3 million lines of code. The entries consist of proof formalizations from a variety of areas of Computer Science, Logic and Mathematics and we believe they are a good source of interesting and useful lemmas, as the lemmas we find there are human-invented as part of proofs that Isabelle users have seen a reason to formalize.

¹ <https://www.isa-afp.org/index.html>

2 Dataset

Our dataset contains 22767 equational lemmas captured by 6567 different templates, and an additional 10454 lemmas whose shapes cannot be fully expressed using our template language, due to the presence of lambdas and quantifiers. Covering also these in the template language is an upcoming extension. They were extracted from 2169 different theory files from 611 AFP entries. Of the remaining 65 AFP entries and 4279 theory files, 2516 files did not contain equational lemmas and 1763 files threw errors in the processing. To automate the lemma extraction, we processed the theories in batch mode, but some theory files seemed to be incompatible with this style of batch mode processing and threw errors resulting in no lemmas being extracted from those theory files. We plan to investigate this further and find a remedy so that lemmas from these theories can also be added to our dataset.

We chose to start by processing only the equational lemmas from the AFP because our current theory exploration tools are primarily designed to generate equational conjectures. However, our intention is to extend both the dataset and the capabilities of our tools in the future to also cover other shapes of lemmas and conjectures. The dataset along with the code used to generate it is available at: <https://github.com/solrun/LibraryOfLemmas>.

2.1 Data Format

For each extracted lemma we output the lemma name (including the name of the file it is defined in), a string representation of the lemma statement, and a template representation of the lemma. The template representation shows the lemma statement's term structure with function and variable names abstracted away but using integer labels to keep track of function symbols and variables that occur more than once in the lemma statement.

Example For example, for the lemma `psi_sqrt` defined in the theory *Bertrand.thy* in the AFP session *Bertrand's Postulate* [1], we output the following data:

```
("Bertrand.psi_sqrt", "psi (Discrete.sqrt ?n) = psi_even ?n",
  template_equation
    (template_app (template_hole 2
      , template_app (template_hole 1
        , template_var 0)),
      template_app (template_hole 0, template_var 0)))
```

We see that the variable `?n` is represented by `template_var 0` and the function symbols `psi` and `Discrete.sqrt` by `template_hole 0` and `template_hole 1` respectively. We replace function symbols by `template_holes` and variables by `template_vars`, which are treated differently by RoughSpec as it builds conjectures. Holes are instantiated by concrete functions given in the input signature, while vars are treated as variables.

The extracted templates may also contain the keywords `template_dunno` or `t_empty`, which mean that the lemma in question contains term structure not yet covered by our template language, such as a lambda expression or an existential quantification.

2.2 Template Frequencies in Dataset

The assumption behind RoughSpec was that certain lemma shapes are common and can be used to efficiently conjecture many (but not all) lemmas by analogy to known shapes. Does this dataset support such an assumption about the space of lemmas in the AFP? In Figure 1 we can see that indeed, a small number of templates occur very frequently in our dataset while the majority occur very seldom with 4099 templates occurring only once. The 10 most frequent templates together describe 3057 lemmas or 13.5% of the lemmas in our set, while more than 50% of the lemmas can be described using only 266 of the 6567 templates, as displayed in Figure 2.2. This supports our hypothesis that only a smaller number of templates is needed to discover many lemmas using template-based conjecturing.

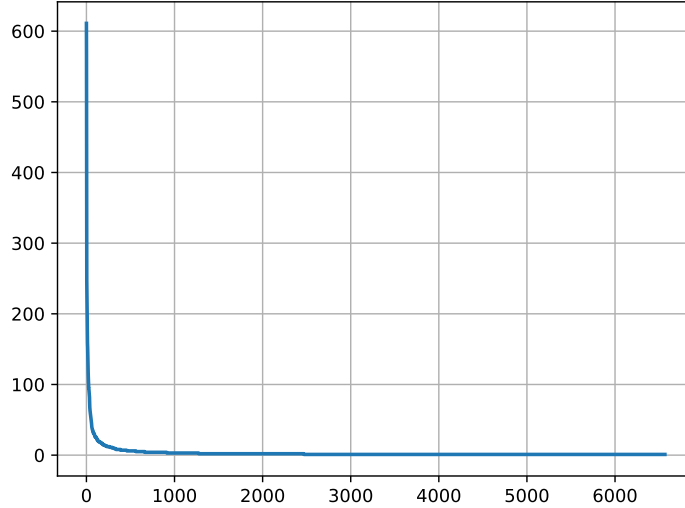


Fig. 1. Number of lemmas per template, sorted by frequency.

Table 1 shows the top 10 most frequently occurring templates in our dataset, where `# lemmas` represents the number of lemmas matching the template, `# thys` is the number of different theory files it occurs in and `# sessions` is the number of different AFP sessions it occurs in. Template holes are represented by a question mark followed by a capitalized name, while variables are represented by a capitalized name. Among these common templates, we see a lot of similarity:

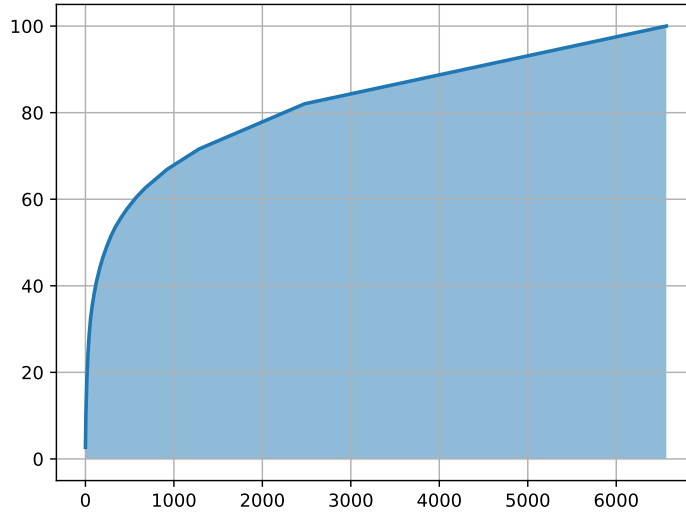


Fig. 2. Cumulative percentage of lemmas in the dataset covered by most frequent templates.

many of the templates in Table 1 resemble each other, differing only in the number of variables or the order of application. For example (1) (6) and (8), (3) (5) and (7), and (9) and (10) should be grouped together and described by common “supertemplates”. This could further reduce the space of templates to be searched over by RoughSpec, as discussed in Section 3.

Template	# lemmas	# thys	# sessions
1 $?F (?G X Y) = ?H (?F X) (?F Y)$	611	261	172
2 $?F X = ?G (?H X)$	566	265	169
3 $X = ?F (?G X)$	340	191	139
4 $?F X = ?F (?G X)$	280	149	118
5 $X = ?F ?G X$	247	136	98
6 $?F (?G X Y) Z = ?H (?F X Z) (?F Y Z)$	233	90	70
7 $X = ?F X ?G$	210	132	103
8 $?F X (?G Y Z) = ?H (?F X Y) (?F X Z)$	194	90	74
9 $?F = ?G (?H X)$	192	65	56
10 $?F = ?G ?H X$	184	110	85

Table 1.

In our previous work [2], we defined a set of 10 default templates capturing very common properties which we found useful in our case studies. Comparing these to the most frequent templates as shown above, we see that 4 out of our 10 default templates are also in the 10 most frequently occurring templates in our dataset as shown in Table 1: *homomorphism* (1), *cancel* (4), *left-id-elem* (5),

right-id-elim (7). Of the remaining six default templates one did not show up at all, and the other occur in places 20–388. The second most commonly occurring template in the dataset, $?F X = ?G (?H X)$, is in a style we had previously disregarded as being too general to be suited to template-based theory exploration, but seeing how common this exact form of equivalence template seems to be we will definitely try out using it in future experiments. The differences between our collection of default templates and the most common templates in the dataset show the value of collecting a dataset for empirical evaluation.

3 Discussion: Applications of the Dataset

Learning templates. Our aim is to build a neuro-symbolic system for conjecturing, where given a theory, a machine learning system selects the most promising templates, and a symbolic system fills in the templates to produce conjectures, avoiding any conjecture which is trivial, trivially false, or already known. While neural conjecturing has been attempted [6,4], a current problem is that such systems often also produce many non-theorems or variants of known conjectures.

Our hypothesis is that a neuro-symbolic approach combines the best of both worlds: machine learning to learn which parts of the search space to focus on, and symbolic methods to reason about and evaluate specific conjectures. A similar idea was explored by Heras et al. [3], in the context of failed proofs, where templates were extracted from lemmas in proofs of similar statements.

Extending the dataset. We are currently working on expanding this dataset to also contain non-equational lemmas, such as conditionals, inequalities, and predicates. We also plan to extend the template language to cover e.g. lambda abstractions and quantifiers. With these extensions we should be able to cover all the lemmas in the AFP. Adding more data concerning for example the topic of the theory where the lemma in question is defined and used or the function definitions involved may also prove necessary in order to use this dataset to learn what templates are useful in various theorem proving contexts.

This dataset was extracted from Isabelle theory files. However, the data format is not specific to Isabelle and it would be interesting to compare this dataset to one extracted from a different source language.

Template families and generalization. We will divide the templates in the dataset into families of similar templates and develop techniques where one template from such a family can be generalized to describe the other family members. Our hypothesis is that we can then define a small set of “supertemplates” where using those templates and their generalizations we can generate a large proportion of the lemmas we need.

Acknowledgements This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation.

References

1. Biendarra, J., Eberl, M.: Bertrand’s postulate. Archive of Formal Proofs (Jan 2017), https://isa-afp.org/entries/Bertrands_Postulate.html, formal proof development.
2. Einarsdóttir, S.H., Smallbone, N., Johansson, M.: Template-based theory exploration: Discovering properties of functional programs by testing. In: Chitil, O. (ed.) IFL 2020: 32nd Symposium on Implementation and Application of Functional Languages, Virtual Event / Canterbury, UK, September 2-4, 2020. pp. 67–78. ACM (2020). <https://doi.org/10.1145/3462172.3462192>
3. Heras, J., Komendantskaya, E., Johansson, M., Maclean, E.: Proof-pattern recognition and lemma discovery in ACL2. In: McMillan, K.L., Middeldorp, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8312, pp. 389–406. Springer (2013). https://doi.org/10.1007/978-3-642-45221-5_27
4. Rabe, M.N., Lee, D., Bansal, K., Szegedy, C.: Mathematical reasoning via self-supervised skip-tree training. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net (2021), <https://openreview.net/forum?id=YmqAnY0CMEy>
5. Smallbone, N., Johansson, M., Claessen, K., Alghed, M.: Quick specifications for the busy programmer. Journal of Functional Programming **27** (2017). <https://doi.org/10.1017/S0956796817000090>
6. Urban, J., Jakubuv, J.: First neural conjecturing datasets and experiments. In: Benzmlüller, C., Miller, B.R. (eds.) Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy, July 26-31, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12236, pp. 315–323. Springer (2020). https://doi.org/10.1007/978-3-030-53518-6_24

PISE – Proofscape Integrated Study Environment

(CICM’22 System Entry)

Steve Kieffer

Alpine Mathematics

Abstract. This system entry gives a brief overview of PISE: an application for representing, studying, and annotating informal proofs from the mathematical literature.

PISE

Steve Kieffer, Alpine Mathematics

Tool:	PISE (v22.8)
Impl. in:	JavaScript & Python
License:	Apache 2.0
Download:	https://proofscape.org/download

Description. PISE (Proofscape Integrated Study Environment) is an application for representing, studying, and annotating informal proofs from the mathematical literature. PISE aims to provide the equivalent for mathematics of an annotated chess game at chess.com, or the detailed and clickable driving directions available in Google maps.

In systems of this kind, the display before the user has two sides: a graphical side, where all the steps to be understood are laid out on a type of board or map; and a textual side, where a discussion guides the user through the steps. The discussion usually has embedded links or other clickable “widgets” that cause the graphical side to “advance,” “navigate,” or “light up” so as to show the part the text is currently talking about. PISE does all this with mathematical proofs. The two sides are called *proof charts* (graphical), and *annotations* (text), and are authored in the Proofscape argument mapping language [2].

Proofscape argument maps are a semi-formal representation of mathematical proofs. The different node types help encode standard structures like proof by contradiction or by cases, while the set of legal inferences is open. This means it is the translator’s job to interpret the intention of the proof’s author, that a given step C follow from previous steps P_1, \dots, P_n , and encode this by drawing arrows from the P_i to C . This system is chosen over any formal logic, because the purpose is to faithfully represent the original form of expression of proofs from the literature. Conceivably, translating from Proofscape to a formal system could later allow automatic filling in of missing steps, or checking existing steps.

Proofscape enables users to provide *expansions* and *example explorers*. An expansion is a set of extra steps that can be added to a proof to help fill in

difficult inferential leaps. Expansions can come in multiple layers, allowing the community of users to progressively explain more and more, until everything is clear. Users are free to open the expansions they need, and ignore others. Meanwhile, an *example explorer* is a set of annotation widgets allowing users to rapidly explore numerical examples of the types of mathematical objects in play at any given step in a proof. The examples are computed using the computer algebra system SymPy [3].

Applications. The purpose of PISE is to make mathematical literature more accessible. On the one hand, it can be used to communicate contemporary work. On the other, improved access to the proofs that make up historic mathematical literature could facilitate the *genetic method* of studying and understanding mathematics through its origins, as first described by Otto Toeplitz [4] and later echoed by authors like Harold M. Edwards [1].

References

1. Edwards, H.M.: Fermat’s last theorem: a genetic introduction to algebraic number theory, Graduate Texts in Mathematics, vol. 50. Springer (1977)
2. Kieffer, S.: Argument mapping for mathematics in Proofscape. In: Dwyer, T., Purchase, H.C., Delaney, A.J. (eds.) Diagrammatic Representation and Inference - 8th International Conference, Diagrams 2014, Melbourne, VIC, Australia, July 28 - August 1, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8578, pp. 57–63. Springer (2014), https://doi.org/10.1007/978-3-662-44043-8_10
3. Meurer, A. et al.: SymPy: symbolic computing in Python. PeerJ Comput. Sci. **3**, e103 (2017), <https://doi.org/10.7717/peerj-cs.103>
4. Toeplitz, O.: Das Problem der Universitätsvorlesungen über Infinitesimalrechnung und ihrer Abgrenzung gegenüber der Infinitesimalrechnung an den höheren Schulen. Jahresbericht der Deutschen Mathematiker-Vereinigung **36**, 88–99 (1927)

Formalising the Krull Topology in Lean

Sebastian Monnet

London School of Geometry and Number Theory

Abstract. The Galois group of an infinite Galois extension has a natural topology, called the *Krull topology*, which has the important property of being profinite. It is difficult to talk about Galois representations, and hence the Langlands Program, without first defining the Krull topology. We explain our formalisation of this topology, and our proof that it is profinite, in the Lean 3 theorem prover.

Keywords: Formalized mathematics · algebraic number theory · Lean · mathlib · Galois groups

1 Introduction

The *Langlands Program* is one of the largest and most ambitious projects in modern mathematics. The program essentially says that there is a correspondence between Galois representations and automorphic forms. Galois representations are required to be *continuous*, which means that it is difficult to define them, let alone state the Langlands conjectures, without first defining the appropriate *Krull topology* on Galois groups. Recent work in [5] has formalised the adèles of a global field, paving the way towards the automorphic side of the Langlands philosophy. Meanwhile, we have formalised the Krull topology, laying groundwork for the Galois-theoretic side.

One interesting feature of our formalisation is that we define the Krull topology for *all* field extensions, without requiring them to be Galois. This is unusually general, and it inspired us to think about how far the abstraction could go. We found that category theory provides a natural language to express the idea in vast generality.

The structure of the paper is as follows. In Section 2, we recap the relevant mathematics by defining field extensions, Galois groups, and the Krull topology, as well as explaining what it means for this topology to be profinite. We conclude Section 2 by explaining informally a proof of profiniteness. Subsequently, in Section 3, we explain our implementation of this definition and proof, building on Lean’s mathematics library, `mathlib`.

Concretely, our contributions are as follows: In `mathlib`, we created a new file, `field_theory/krull_topology`, which currently contains the definition of the Krull topology and proofs that it is Hausdorff and totally disconnected. An overview of this file’s contents can be found at: https://leanprover-community.github.io/mathlib_docs/field_theory/krull_topology.html. At the time of writing, our proof that the Krull topology is profinite has not yet been merged

into `mathlib`, and the most recent version can be found in the Pull Request at <https://github.com/leanprover-community/mathlib/pull/13307>.

2 Mathematical Preliminaries

We summarise the key mathematical theory underlying our work, starting with field extensions, Galois groups, and the Krull topology, which are familiar to most number theorists. Once we have defined these objects, we move on to explaining the language of *filters*, which give a convenient framework for dealing with topological notions. Finally, we apply these filters to prove that the Krull topology is profinite.

2.1 Field Theory and Galois Theory

A *field extension* L/K is a pair of fields K and L , such that K is a subset of L . For example, the field \mathbb{Q} of rational numbers is a subset of the real numbers \mathbb{R} , so \mathbb{R}/\mathbb{Q} is a field extension. Similarly, \mathbb{C}/\mathbb{R} is a field extension, where \mathbb{C} denotes the complex numbers. For a field extension L/K , an element $\alpha \in L$ is said to be *algebraic over K* if it is a root of some polynomial with coefficients in K . If $\alpha \in L$ is algebraic over K , then there is a unique monic polynomial $f_\alpha(X) \in K[X]$ of least degree such that $f_\alpha(\alpha) = 0$. This polynomial is called the *minimal polynomial* of α over K . A field extension L/K is said to be *algebraic* if every element of L is algebraic over K . The extension \mathbb{C}/\mathbb{R} is algebraic, but \mathbb{R}/\mathbb{Q} is not.

Two important properties of some field extensions are *normality* and *separability*. The extension L/K is said to be *normal* if it is algebraic and the minimal polynomial of each $\alpha \in L$ can be factorised into linear factors over L . The extension \mathbb{C}/\mathbb{R} is normal. For an example of an extension that is algebraic but not normal, consider the field

$$\mathbb{Q}(\sqrt[3]{2}) = \{a + b2^{1/3} + c2^{2/3} : a, b, c \in \mathbb{Q}\}$$

as an extension of \mathbb{Q} . The minimal polynomial of $\sqrt[3]{2}$ is $X^3 - 2$, which cannot be factorised into linear factors over $\mathbb{Q}(\sqrt[3]{2})$, since it has roots $e^{\pm 2\pi i/3} \sqrt[3]{2}$, and these are not in $\mathbb{Q}(\sqrt[3]{2})$. Meanwhile, an algebraic extension L/K is said to be *separable* if the minimal polynomial of each $\alpha \in L$ over K splits into distinct linear factors over an algebraic closure of L . All algebraic extensions of \mathbb{Q} are separable, since each minimal polynomial is coprime to its derivative, meaning that it cannot have repeated roots in any extension.

The classic example of a non-separable algebraic extension comes from the field

$$L = \frac{\mathbb{F}_p(T)[X]}{(X^p - T)},$$

where p is prime, \mathbb{F}_p is the field with p elements, and $\mathbb{F}_p(T)$ is the field of rational functions over \mathbb{F}_p . Write $\sqrt[p]{T}$ for the element of L represented by $X \in \mathbb{F}_p(T)[X]$.

As the notation suggests, intuitively L is obtained from $\mathbb{F}_p(T)$ by adjoining a p^{th} root of T . Then $L/\mathbb{F}_p(T)$ is a non-separable algebraic extension, since the minimal polynomial $f_{\sqrt[p]{T}}(y)$ of $\sqrt[p]{T}$ factorises as $(y - \sqrt[p]{T})^p$ over L . The extension L/K is said to be *Galois* if it is normal and separable.

If L/K is a field extension, then L naturally has the structure of a vector space over K . We define the *degree* of this extension to be the dimension of L as a K -vector space, and we denote it by $[L : K]$. The most obvious example is \mathbb{C}/\mathbb{R} . Since the complex plane is a 2-dimensional real vector space with basis $\{1, i\}$, the degree is $[\mathbb{C} : \mathbb{R}] = 2$. A slightly more involved example is $\mathbb{Q}(\sqrt[3]{2})/\mathbb{Q}$ from before, which has degree 3, since it has basis $\{1, 2^{1/3}, 2^{2/3}\}$ over \mathbb{Q} .

If L/K is a field extension, then an *intermediate field* of L/K is another field E such that $K \subseteq E \subseteq L$. We will also refer to intermediate fields as *subextensions* of L/K . In the case where a subextension F of L/K is of finite degree over K , we will call it a *finite subextension*.

Definition 1. Let L/K be a field extension. A K -algebra homomorphism $L \rightarrow L$ is a function $\sigma : L \rightarrow L$ satisfying the following three axioms:

1. $\sigma(x + y) = \sigma(x) + \sigma(y)$ for all $x, y \in L$,
2. $\sigma(xy) = \sigma(x)\sigma(y)$ for all $x, y \in L$,
3. $\sigma(x) = x$ for all $x \in K$.

If moreover $\sigma : L \rightarrow L$ is a bijection, then it is called a K -algebra isomorphism.

Definition 2. Let L/K be any field extension. We define the Galois group $\text{Gal}(L/K)$ of L/K to be the set of K -algebra isomorphisms $\sigma : L \rightarrow L$, which is a group under composition.

Remark 1. It is slightly unconventional to define the Galois group of a field extension that is not a Galois extension. Usually, this object would be denoted $\text{Aut}_K(L)$. However, in `mathlib`, both objects are represented by the same notation, and all of our Lean definitions and results apply to non-Galois extensions. Since it will not matter to us whether an extension is Galois, we will use the notation $\text{Gal}(L/K)$ for all extensions L/K .

In general, one may define a pair of maps

$$\{\text{subgroups of } \text{Gal}(L/K)\} \xrightleftharpoons[\text{Gal}(L/E) \leftarrow E]{H \mapsto L^H} \{\text{intermediate fields of } L/K\},$$

where

$$L^H := \{x \in L : \sigma(x) = x \text{ for all } \sigma \in H\}.$$

Note that for an intermediate field E of L/K , the group $\text{Gal}(L/E)$ is indeed a subgroup of $\text{Gal}(L/K)$, since an isomorphism of L fixing E certainly also fixed K .

We call L^H the *fixed subfield* of H , since it consists of the elements of L that are fixed by H . Similarly, when viewed as a subgroup of $\text{Gal}(L/K)$, the group

$\text{Gal}(L/E)$ is called the *fixing subgroup* of E , since it consists of the elements of $\text{Gal}(L/K)$ fixing E .

One reason to care about Galois groups is the following theorem, which is a special case of [6], Page 120, Theorem 7.34.

Theorem 1 (Fundamental Theorem of Galois Theory). *If L/K is a Galois extension of finite degree, then the maps $H \mapsto L^H$ and $E \mapsto \text{Gal}(L/E)$ are mutually inverse bijections.*

2.2 The Krull Topology

The Fundamental Theorem of Galois Theory breaks down for infinite Galois extensions. See [3], Examples 3.10 and 3.11 for counterexamples. To salvage the theorem, we define a topology on $\text{Gal}(L/K)$.

Let L/K be a Galois extension, possibly of infinite degree. Recall that a *finite subextension* of L/K is an intermediate field F such that F/K is of finite degree.

Definition 3. *We define the Krull topology on $\text{Gal}(L/K)$ to be the topology generated by sets of the form*

$$\sigma \text{Gal}(L/F) := \{\sigma f : f \in \text{Gal}(L/F)\},$$

where $\sigma \in \text{Gal}(L/K)$ and F/K is a finite subextension of L/K .

If cosets $\sigma \text{Gal}(L/F_1)$ and $\tau \text{Gal}(L/F_2)$ have nonempty intersection, then for every $\varphi \in \sigma \text{Gal}(L/F_1) \cap \tau \text{Gal}(L/F_2)$, we have

$$\varphi \in \varphi \text{Gal}(L/F_1 F_2) \subseteq \sigma \text{Gal}(L/F_1) \cap \tau \text{Gal}(L/F_2),$$

where $F_1 F_2$ is the smallest subfield of L containing F_1 and F_2 , which is also of finite degree over K . Therefore, the open sets of the Krull topology are precisely the unions of cosets of the form $\sigma \text{Gal}(L/F)$ for finite subextensions F of L/K .

Definition 4. *A topological group is a group G , equipped with a topology, such that the maps*

$$\begin{aligned} G \times G &\rightarrow G, & G &\rightarrow G \\ (x, y) &\mapsto xy, & x &\mapsto x^{-1} \end{aligned}$$

are both continuous.

In the remainder of Section 2, we will state several results whose proofs are elementary. Since the focus of the paper is on computation, we will omit these elementary proofs in order to spend more time discussing our implementation.

Lemma 1. *Let L/K be a (possibly infinite) Galois extension. The Krull topology makes $\text{Gal}(L/K)$ into a topological group.*

The following theorem salvages the Fundamental Theorem of Galois Theory for infinite extensions.

Theorem 2 (Krull). *The mappings $E \mapsto \text{Gal}(L/E)$ and $H \mapsto L^H$ are mutually inverse bijections between intermediate fields of L/K and closed subgroups of $\text{Gal}(L/K)$.*

Proof. This is Part (1) of [1], Theorem I.2.8.

Definition 5. *A profinite group is a topological group that is isomorphic to the limit of an inverse system of finite groups, with their discrete topologies.*

As F ranges over finite normal subextensions F/K , the groups $\text{Gal}(F/K)$ form an inverse system of finite topological groups, equipped with their discrete topologies. The restriction maps $\text{Gal}(L/K) \rightarrow \text{Gal}(F/K)$ make the group $\text{Gal}(L/K)$ into the limit of this inverse system, in the category of topological groups.

Therefore, $\text{Gal}(L/K)$ is a profinite group. The following theorem gives a convenient explicit condition for profiniteness.

Theorem 3. *A topological group is profinite if and only if its topology is compact, Hausdorff, and totally disconnected.*

Proof. This is Theorem 2 of [7].

In `mathlib`, a space is *defined* to be profinite if it is compact, Hausdorff, and totally disconnected, so those are the conditions we proved.

2.3 Filters and Filter Bases

Our proof of profiniteness uses the language of filters, which we now explain.

Definition 6. *A filter on a set X is a collection \mathcal{F} of subsets of X , satisfying the following axioms:*

1. (Universality) $X \in \mathcal{F}$,
2. (Upward closure) If $S \in \mathcal{F}$ and $S \subseteq T \subseteq X$, then $T \in \mathcal{F}$,
3. (Closure under intersection) If $S, T \in \mathcal{F}$, then $S \cap T \in \mathcal{F}$.

Definition 7. *A filter bundle on a set X is any function*

$$\mathcal{N} : X \rightarrow \{\text{filters on } X\}.$$

Lemma 2. *Let \mathcal{N} be a filter bundle on a set X . Define*

$$\mathcal{T} = \{U \subseteq X : U \in \mathcal{N}(x) \text{ for all } x \in U\}.$$

Then \mathcal{T} is a topology on X .

The topology from Lemma 2 is called the topology *induced by* \mathcal{N} . We also introduce the concept of a filter basis, which is a reduced set of data that generates a filter.

Definition 8. A filter basis on a set X is a collection \mathcal{B} of subsets of X , satisfying the following two axioms:

1. $\mathcal{B} \neq \emptyset$
2. For all $U, V \in \mathcal{B}$, there is some $W \in \mathcal{B}$ with $W \subseteq U \cap V$.

Lemma 3. Let \mathcal{B} be a filter basis on a set X . The collection

$$\mathcal{F} = \{U \subseteq X : D \subseteq U \text{ for some } D \in \mathcal{B}\}$$

is a filter on X .

The filter \mathcal{F} from Lemma 3 is called the *filter induced by* \mathcal{B} .

2.4 Group Filter Bases

At this point, we can come up with a strategy for obtaining a topology on a group from a filter basis. The naïve strategy is as follows:

1. Start with a filter basis \mathcal{B} on a group G .
2. For each $g \in G$, write $g \cdot \mathcal{B}$ for the collection of cosets $\{g \cdot D : D \in \mathcal{B}\}$.
3. It is easy to see that each $g \cdot \mathcal{B}$ is a filter basis on G , so we may define $\mathcal{N}(g)$ to be the filter induced by $g \cdot \mathcal{B}$.
4. Then \mathcal{N} is a well-defined filter bundle on G , so it induces a topology.

Let \mathcal{B} be a filter basis on a group G , and define a topology on G as outlined above. This topology is called the *topology induced by* \mathcal{B} .

Each filter basis on G does induce a topology, but that topology will not in general make G into a topological group, since multiplication and inversion may not be continuous. We can remedy this by imposing four additional conditions on our filter basis, which we do in the following definition.

Definition 9. Let G be a group. A group filter basis on G is a filter basis \mathcal{B} on G such that:

1. $1 \in U$ for all $U \in \mathcal{B}$.
2. For all $U \in \mathcal{B}$, there is some $V \in \mathcal{B}$ with $V \cdot V \subseteq U$.
3. For all $U \in \mathcal{B}$, there is some $V \in \mathcal{B}$ with $V \subseteq U^{-1}$.
4. For all $x \in G$ and $U \in \mathcal{B}$, there is some $V \in \mathcal{B}$ with $xVx^{-1} \subseteq U$.

Group filter bases were first defined in [2], Pages 222-223, and the discussion preceding that definition explains how to define the induced group topology on G , as we have done here. In `mathlib`, group filter bases were formalised by Patrick Massot.

It turns out that these are the conditions we need for the induced topology on G to make G into a topological group, as we see from the following theorem. The proof is trickier than most of the surrounding lemmas, but still elementary, so we omit it.

Theorem 4. *Let \mathcal{B} be a group filter basis on a group G . Then the topology induced by \mathcal{B} makes G into a topological group.*

Lemma 4. *Let L/K be any extension of fields, and define the set*

$$\mathcal{B} = \{\text{Gal}(L/F) : F/K \text{ is a finite subextension of } L/K\}.$$

Then \mathcal{B} is a group filter basis for $\text{Gal}(L/K)$.

For an extension L/K , the group filter basis from Lemma 4 is called the *standard group filter basis* on $\text{Gal}(L/K)$.

Lemma 5. *Let L/K be a Galois extension of fields. The topology induced by the standard group filter basis on $\text{Gal}(L/K)$ is equal to the Krull topology.*

2.5 Proof that the Krull Topology is Profinite

We have written Lean proofs that the Krull topology is Hausdorff, totally disconnected, and compact. The former two properties are elementary, so we will not explain their proofs here. Proving compactness is more difficult, and this subsection is devoted to explaining its proof informally.

Compactness has an equivalent characterisation involving filters. Before we can state this criterion (which we do in Theorem 5), we need two more definitions.

Definition 10. *Let X be a topological space. An ultrafilter on X is a filter \mathcal{F} on X satisfying the following two axioms:*

1. $\emptyset \notin \mathcal{F}$
2. *For any filter \mathcal{G} on X with $\emptyset \notin \mathcal{G}$ and $\mathcal{F} \subseteq \mathcal{G}$, we have $\mathcal{G} = \mathcal{F}$.*

Definition 11. *Given a point x of a topological space X , the neighbourhood filter of x is the filter*

$$\mathcal{N}(x) = \{N \subseteq X : \text{there is some open } U \subseteq X \text{ with } x \in U \subseteq N\}.$$

The following theorem gives a convenient equivalent condition for a topological space to be compact.

Theorem 5. *Let X be a topological space. The following are equivalent:*

1. *X is compact.*
2. *For every ultrafilter \mathcal{F} on X , there is some $x \in X$ such that $\mathcal{N}(x) \subseteq \mathcal{F}$.*

Proof. This is Theorem 2.2.5 of [4].

Following Theorem 5, the key idea in our proof of compactness is as follows: take an arbitrary ultrafilter \mathcal{F} on $\text{Gal}(L/K)$ and construct an element $\sigma \in \text{Gal}(L/K)$ such that $\mathcal{N}(\sigma) \subseteq \mathcal{F}$. For each point $x \in L$ and each finite normal subextension F/K containing x , we will construct a K -algebra homomorphism

$\varphi_{F,x} : F \rightarrow L$, satisfying certain properties. We will then glue these “local” K -algebra homomorphisms together to define a “global” K -algebra isomorphism $\sigma : L \rightarrow L$.

So, let \mathcal{F} be an ultrafilter on $\text{Gal}(L/K)$. Let $x \in L$ and let F/K be a finite subextension containing x . There is a restriction map $p : \text{Gal}(L/K) \rightarrow \text{Hom}_K(F, L)$, and we obtain a pushforward $p_*\mathcal{F}$ of \mathcal{F} along p , which consists of sets whose preimages under p are in \mathcal{F} . It is easy to see that the pushforward of an ultrafilter is an ultrafilter, and also that ultrafilters on finite sets are always principal¹. It follows that $p_*\mathcal{F}$ is generated by some element $\varphi_{F,x} \in \text{Hom}_K(F, L)$. It turns out that the element $\varphi_{F,x}(x) \in L$ is independent of the choice of F , so there is a well-defined element $\sigma(x) \in L$ such that $\sigma(x) = \varphi_{F,x}(x)$ for all finite subextensions F containing x . This allows us to define a function $\sigma : L \rightarrow L$.

Now, for any elements $x, y \in L$, setting $F = K(x, y)$ shows that $\sigma(x + y) = \sigma(x) + \sigma(y)$, and similarly for products. Therefore, σ is a ring homomorphism. Clearly σ fixes K , so it is a K -algebra homomorphism.

We need to show that σ is actually a K -algebra isomorphism $L \rightarrow L$. Field homomorphisms are always injective, so we only need to show that σ is surjective. Let $y \in L$ and write f_y for its minimal polynomial over K . Let y_1, y_2, \dots, y_n be the roots of f_y in L , and let $F = K(y_1, \dots, y_n)$. Then F/K is a finite extension and $\varphi_{F,y}$ is a K -algebra homomorphism $F \rightarrow F$, which means that $\varphi_{F,y}$ gives an isomorphism $F \rightarrow F$. In particular, there is some $x \in F \subseteq L$ such that $\varphi_{F,y}(x) = y$, so $\sigma(x) = y$. Therefore, σ is surjective.

Now, if $U \in \mathcal{N}(\sigma)$, then by definition of the Krull topology, there is some finite subextension F/K with $\sigma \cdot \text{Gal}(L/F) \subseteq U$. Now, if $p : \text{Gal}(L/K) \rightarrow \text{Gal}(L/F)$ is the restriction map, then by definition, the ultrafilter $p_*\mathcal{F}$ is generated by $\sigma|_F$, which means that $\{\sigma|_F\} \in p_*\mathcal{F}$. By definition of pushforwards, the set

$$p^{-1}(\{\sigma|_F\}) = \sigma \text{Gal}(L/F)$$

is in \mathcal{F} , which means that U is too, by upward closure. Therefore we have $\mathcal{N}(\sigma) \subseteq \mathcal{F}$, so Theorem 5 tells us that $\text{Gal}(L/K)$ is a compact topological space.

3 Implementation in Lean

In this section, we give an overview of our implementation in Lean. We start by explaining how we defined the Krull topology and then discuss our proof that it is profinite.

3.1 Type Theory Basics

Instead of set theory, Lean is based on the formalism of *dependent type theory*. Roughly speaking, one can think of a *type* as a collection of things, like a set. Instead of being called elements, the things inside a type are called *terms*. Given

¹ We have not officially defined this. It just means that the ultrafilter consists of all sets containing a designated element, called the “generator” of the ultrafilter.

a type X , we write $x : X$ to say that x is a term of type X . In keeping with the analogy to sets, we basically write $:$ to mean \in .

For any pair X, Y of types, there is another type $X \rightarrow Y$ of *functions* from X to Y . These do what we expect them to; they assign a term of Y to each term of X .

There is a special sort of type called a *proposition*. A proposition can have at most one term. If the proposition has a term, we say that it is *true*, and *false* otherwise. If a proposition P is true, then we call the unique term $p : P$ the *proof* of P . The fact that propositions have at most one term is called *proof irrelevance*, and it is a foundational design choice of Lean.

Moreover, for any propositions P and Q , the type $P \rightarrow Q$ of functions from P to Q is also a proposition. If the proposition $P \rightarrow Q$ is true (i.e. if there exists a function from P to Q), then we say that P *implies* Q . Note that this is just a formalism - it is not obvious that these notions of propositions, truth, and implication actually align with our conventional understanding of the words. It turns out though that traditional logic is naturally emergent from the formalism.

For example, suppose that we have propositions P, Q , and R , such that P implies Q and Q implies R . By definition, the propositions $P \rightarrow Q$ and $Q \rightarrow R$ are true. Let $f : P \rightarrow Q$ and $g : Q \rightarrow R$ be the proofs of these propositions. Then, composing functions, we obtain a term $g \circ f$ of type $P \rightarrow R$. That is, we have constructed a proof of $P \rightarrow R$, which means that $P \rightarrow R$ is true, so P implies R . Therefore, we have recovered the intuitive notion of transitivity of implication from our formal definitions.

3.2 Definition of the Krull Topology

Let L/K be a field extension, not necessarily Galois. In Lean, we define the Krull topology on $\text{Gal}(L/K)$ to be the topology generated by the standard group filter basis on $\text{Gal}(L/K)$. In the `mathlib` API, the Galois group of L/K is denoted $L \simeq_a [K] L$.

Remark 2. Understanding the notation $L \simeq_a [K] L$ sheds light on the way that `mathlib` is organised, so we will take a moment to explain it. In `mathlib`, we write $L \rightarrow L$ for the type of functions from L to L . These functions do not see the additional structure of L as a K -algebra. On the other hand, we write $L \rightarrow_a [K] L$ for the type of K -algebra homomorphisms from L to L . Mathematicians typically think of an algebra homomorphism as being “the same thing” as its underlying function; a homomorphism is just a function satisfying some additional conditions. In `mathlib`, however, an algebra homomorphism is a different object from its underlying function. The algebra homomorphism *contains* the underlying function, along with proofs that the function satisfies the required properties.

Finally then, $L \simeq_a [K] L$ denotes the type of K -algebra *equivalences* from L to L . Again, in `mathlib`, a K -algebra equivalence is different from a bijective homomorphism. A term of $L \simeq_a [K] L$ consists of the following data:

1. A term `to_fun` of type $L \rightarrow L$,

2. A term `inv_fun` of type $L \rightarrow L$,
3. Proofs that `to_fun` and `inv_fun` are mutual inverses, and also that they satisfy the properties of K -algebra homomorphisms.

These different data structures give insight into why formalising mathematics is difficult. When using Lean, we always have to keep track of distinctions between objects that we intuitively consider to be “the same”, but whose implementations are fundamentally different objects.

For each intermediate field E of L/K , the subgroup of terms σ of $L \simeq_a [K] L$ fixing E is called `E.fixing_subgroup`. Upon encountering this notation for the first time, one might ask how Lean knows that `E.fixing_subgroup` is in fact a subgroup of $L \simeq_a [K] L$. That is, the subgroup depends on L and K , whereas its definition mentions only E . The explanation is that E is a term of type `intermediate_field K L`, so we can recover K and L by looking at the type of E .

Remark 3. Note that $L \simeq_a [K] L$ and `K.fixing_subgroup` are different objects in `mathlib`; although they both represent the Galois group of L/K , the former has type `Type u` for some universe u , while the latter has type `set (L \simeq_a [K] L)`.

In order to define the standard group filter basis, we define `finite_exts K L` to be the set of intermediate fields F of L/K such that F/K is finite dimensional:

```
def finite_exts (K : Type*) [field K] (L : Type*) [field L]
  [algebra K L] :
  set (intermediate_field K L) :=
  {E | finite_dimensional K E}
```

Subsequently, we define the set `fixed_by_finite K L` consist of subsets of the form `F.fixing_subgroup`, as F ranges over `finite_exts K L`:

```
def fixed_by_finite (K L : Type*) [field K] [field L]
  [algebra K L]: set (subgroup (L  $\simeq_a$  [K] L)) :=
  intermediate_field.fixing_subgroup '' (finite_exts K L)
```

Remark 4. The `''` in the definition of `fixed_by_finite` denotes the image of a set under a function. In general, if X and Y are types, $f : X \rightarrow Y$ is a function, and $S : \text{set } X$ is a set of terms of X , then $f '' S$ denotes the image of S under the function f .

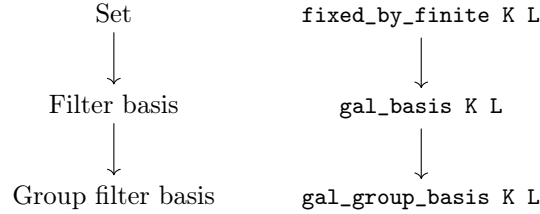
The elements of `fixed_by_finite K L` are then precisely the elements of the standard group filter basis. However, as far as Lean is concerned, `fixed_by_finite K L` is *not* a group filter basis, but merely a set of subgroups of $L \simeq_a [K] L$. In `mathlib`, a term of type `group_filter_basis K L` consists of the following data:

1. A term of type `filter_basis K L`,
2. Four proofs, showing that the filter basis in question satisfies the additional axioms of a group filter basis.

In turn, a term of type `filter_basis K L` consists of:

1. A term of type `set (L \simeq_a [K] L)`,
2. Two proofs, showing that the set in question satisfies the axioms of a filter basis.

The underlying set of our filter basis is `fixed_by_finite K L`, which we package into the filter basis `gal_basis K L`. Subsequently, we use `gal_basis K L` as the underlying filter basis of the group filter basis `gal_group_basis K L`. This hierarchy is somewhat awkward to write out in prose, so we summarise it in the following diagram:



Remark 5. It is important to keep track of the different types of these terms, since Lean considers them to be totally different objects. This can be unintuitive for mathematicians, since we would generally consider the (group) filter basis to be *the same thing* as its underlying set.

We can now define the Krull topology, `krull_topology K L`, on the group $L \simeq_a [K] L$ by:

```

instance krull_topology (K L : Type*) [field K] [field L]
  [algebra K L] :
  topological_space (L  $\simeq_a$  [K] L) :=
  group_filter_basis.topology (gal_group_basis K L)

```

Remark 6. We defined `krull_topology K L` as an instance, which means that the type class inference system understands it as “the” topology on a Galois group. This means that we can make topological statements about subsets of $L \simeq_a [K] L$, and the elaborator will automatically infer that we are talking about the Krull topology.

Remark 7. Our definition of the Krull topology is valid for any field extension, not necessarily normal or separable. This is more general than definitions in the literature, which made us think about how far the generalisation could go. In fact, it can be taken much further, as we now explain. Fix a category \mathcal{C} , and let L be an object of \mathcal{C} . There is a category $\text{subobj}(L)$ of *subobjects* of L . The objects of $\text{subobj}(L)$ are pairs (E, i) , where $E \in \mathcal{C}$ and $i : E \hookrightarrow L$ is a monomorphism. The morphisms of $\text{subobj}(L)$ are the obvious commutative triangles. Suppose further that there is a set S of objects of $\text{subobj}(\mathcal{C})$, whose members we will call *small* objects of \mathcal{C} , with the following two axioms:

1. (The intersection of subobjects contains a subobject). For all $(E_1, i_1), (E_2, i_2) \in S$, there is some $(E, i) \in S$ and maps $f_j : E \rightarrow E_j$ such that the diagram

$$\begin{array}{ccc}
 & E_1 & \\
 f_1 \nearrow & \downarrow i_1 & \\
 E & \xrightarrow{i} & L \\
 f_2 \searrow & \uparrow i_2 & \\
 & E_2 &
 \end{array}$$

commutes.

2. (Automorphisms preserve subobjects). For all $(E, i) \in S$ and all $\sigma \in \text{Aut}_{\mathcal{C}}(L)$, we also have $(E, \sigma \circ i) \in S$.

For each $(E, i) \in \text{subobj}(\mathcal{C})$, define the *fixing subgroup* of (E, i) to be $F(E, i) \subseteq \text{Aut}_{\mathcal{C}}(L)$ to be

$$F(E, i) = \{\sigma \in \text{Aut}_{\mathcal{C}}(L) : \sigma \circ i = i\}.$$

Then the collection

$$\mathcal{B} = \{F(E, i) : (E, i) \in S\}$$

is a group filter basis for $\text{Aut}_{\mathcal{C}}(L)$, so it gives it the structure of a topological group. The Krull topology is a special case of this construction, where \mathcal{C} is the category of K -algebras and L is viewed as an object of \mathcal{C} . In that case, Axiom (1) comes from the fact that the intersection of two K -subalgebras of L is a K -subalgebra of L , and Axiom (2) comes from the fact that $\sigma(E)$ is a K -subalgebra of L , for all K -subalgebras E and all K -isomorphisms $\sigma : L \rightarrow L$. We did not treat this level of generality in Lean, since it seemed quite far-removed from our objective.

One of the most surprising difficulties we encountered was in proving that `fixed_by_finite` K L satisfies Axiom (2) of Definition 9, which is equivalent to proving that the join of two finite-dimensional field extensions is finite-dimensional. Mathematically, this is very simple; if F/K and E/K are finite extensions with bases $\{x_i\}$ and $\{y_j\}$ respectively, then the products $\{x_i y_j\}$ form a finite spanning set for the join FE , so FE is finite-dimensional over K . Since this result is so elementary, we assumed that it must already be in `mathlib`, so we asked in the Zulip chat² about where it might be. It turned out that the result was not in `mathlib`, and that proving it in Lean was actually quite difficult. Thomas Browning generously wrote the proof, and it is now in `mathlib` under the name `intermediate_field.finite_dimensional_sup`.

The difficulty stemmed from the fact that finiteness is hard to formalise, and there are numerous ways to approach it in Lean. For example, given a type \mathbf{x} , there are types `list` \mathbf{x} , `multiset` \mathbf{x} , and `finset` \mathbf{x} , which all capture slightly different notions of “a finite collection of terms of \mathbf{x} ”, depending on whether we

² Much of Lean’s community uses a dedicated server on the chat and collaborative software, Zulip.

care about ordering and multiplicity. There is also a type called `fintype` X , which has a term `if` and only if X contains finitely many elements. On the other hand, given a term s of type `set` X , there is a proposition `finite` s , which says that the set is finite. Each approach has pros and cons, and choosing the right tool for the job can be difficult. Moreover, we often have to manage interactions between multiple notions of finiteness, which requires a clear understanding of the relationships between them.

3.3 Proof of Profiniteness

The proofs that $\text{Gal}(L/K)$ is Hausdorff and totally disconnected are straightforward, so we will not say much about them. They are formalised in

```
lemma krull_topology_t2 {K L : Type*} [field K] [field L]
  [algebra K L] (h_int : algebra.is_integral K L) :
  t2_space (L  $\simeq_a$  [K] L) :=
```

and

```
lemma krull_topology_totally_disconnected {K L : Type*} [field K]
  [field L] [algebra K L] (h_int : algebra.is_integral K L) :
  is_totally_disconnected (set.univ : set (L  $\simeq_a$  [K] L)) :=
```

Remark 8. In the lemmas above, `h_int` is a term of type `is_integral` K L . Therefore, the Krull topology is Hausdorff and totally disconnected whenever the extension L/K is algebraic. Note that normality and separability do not enter the picture.

Our proof of compactness, which we explained informally in Section 2.5, is more involved. Given an ultrafilter \mathcal{F} on $\text{Gal}(L/K)$, a finite normal subextension F/K , and an element $x \in F$, we defined a K -algebra homomorphism $\varphi_{F,x} : F \rightarrow L$. We then glued the various $\varphi_{F,x}$ together to obtain a map $\sigma \in \text{Gal}(L/K)$ with $\mathcal{N}(\sigma) \subseteq \mathcal{F}$. In Lean, the homomorphism $\varphi_{F,x}$ is defined by

```
protected noncomputable def ultrafilter.generator_of_pushforward
  (h_findim : finite_dimensional K E) (f : ultrafilter (L  $\rightarrow_a$  [K] L)) :
  E  $\rightarrow_a$  [K] L :=
  classical.some $ ultrafilter.eq_principal_of_fintype _ $
    f.map $  $\lambda$   $\sigma$ ,  $\sigma$ .comp $ intermediate_field.val _
```

Remark 9. The definition above is labelled as `noncomputable` because it uses Lean's axiom of choice. In particular,

```
$ ultrafilter.eq_principal_of_fintype _ $
  f.map $  $\lambda$   $\sigma$ ,  $\sigma$ .comp $ intermediate_field.val _
```

is a term of type

```
 $\exists$  (x : F  $\rightarrow_a$  [K] L),  $\uparrow$ (ultrafilter.map ( $\lambda$  ( $\sigma$  : L  $\simeq_a$  [K] L),
   $\sigma$ .to_alg_hom.comp F.val) f) = pure x.
```


This means that there exists *some* K -algebra homomorphism $F \rightarrow L$ that generates the ultrafilter $p_*\mathcal{F}$ on $\text{Hom}_K(F, L)$. However, the statement is nonconstructive³, so we have to invoke the axiom of choice to take a specific such algebra homomorphism, which is our $\varphi_{F,x}$.

Subsequently, we glue the local K -algebra homomorphisms $\varphi_{F,x}$ together to obtain the function $\sigma : L \rightarrow L$, defined by:

```
protected noncomputable def
  ultrafilter.glued_generators_of_pushforwards_function
    (h_int : algebra.is_integral K L) (f : ultrafilter (L →a [K] L))
    (x : L) :
L :=
```

Now that we have defined σ as a function, we need to define it as a K -algebra homomorphism by

```
noncomputable def
  ultrafilter.glued_generators_of_pushforwards_alg_hom
    (f : ultrafilter (L →a [K] L)) (h_int : algebra.is_integral K L) :
L →a [K] L :=
```

Next, we prove a lemma, saying that the algebra homomorphism is bijective:

```
lemma ultrafilter.glued_generators_of_pushforwards_alg_hom_bijection
  (h_int : algebra.is_integral K L) (f : ultrafilter (L →a [K] L)) :
function.bijjective
  (ultrafilter.glued_generators_of_pushforwards_alg_hom f h_int) :=
```

As we saw in Remark 2, `mathlib` considers a K -algebra equivalence $E \simeq_a [K] L$ to be different from a bijective algebra homomorphism. It consists of two *different* functions $E \rightarrow L$ and $L \rightarrow E$, together with proofs that they are mutual inverses and that they satisfy the axioms of K -algebra homomorphisms. Thankfully, `mathlib` contains a definition, `alg_equiv.of_bijjective`, which takes a bijective algebra homomorphism and constructs an algebra equivalence whose underlying function equals the underlying function of the algebra homomorphism. We include the statement of `alg_equiv.of_bijjective` for completeness:

```
noncomputable def alg_equiv.of_bijjective (f : A1 →a [R] A2)
  (hf : function.bijjective f) : A1 ≃a [R] A2 :=
```

Now we can finally define σ as a term of the Galois group $L \simeq_a [K] L$, as follows:

```
noncomputable def ultrafilter.glued_generators_of_pushforwards_alg_equiv
  (h_int : algebra.is_integral K L) (f : ultrafilter (L →a [K] L)) :
(L ≃a [K] L) :=
alg_equiv.of_bijjective
  (ultrafilter.glued_generators_of_pushforwards_alg_hom f h_int)
  (ultrafilter.glued_generators_of_pushforwards_alg_hom_bijection
    h_int f)
```

³ Meaning that there is no explicit formula for this homomorphism.

All that remains is to show that this equivalence is actually a limit of the ultrafilter⁴, which we do with the following lemma:

```
lemma ultrafilter_converges_to_glued_equiv
  (h_int : algebra.is_integral K L) (f : ultrafilter (L  $\simeq_a$  [K] L)) :
  (f : filter (L  $\simeq_a$  [K] L))  $\leq$ 
  nhds (ultrafilter.glued_generators_of_pushforwards_alg_equiv h_int
    (f.map ( $\lambda$  ( $\sigma$  : L  $\simeq_a$  [K] L),  $\sigma$ .to_alg_hom))) :=
```

At this point, we are pretty much done; our actual proof of compactness is the lemma:

```
lemma krull_topology_compact {K L : Type*} [field K] [field L]
  [algebra K L] (h_int : algebra.is_integral K L) :
  is_compact (set.univ : set (L  $\simeq_a$  [K] L)) :=
```

This is fairly immediate from `is_compact_iff_ultrafilter_le_nhds`, which is `mathlib`'s statement of 5. Finally, we prove profiniteness by

```
def krull_topology_profinite {K L : Type*} [field K] [field L]
  [algebra K L] (h_int : algebra.is_integral K L)
  (minpoly K x) :
  Profinite :=
  { to_CompHaus := krull_topology_comphaus h_int,
    is_totally_disconnected :=
    krull_topology_totally_disconnected_space h_int }
```

4 Conclusion and Acknowledgements

For any field extension L/K , not necessarily algebraic, normal, or separable, we defined a canonical topology on the group $L \simeq_a [K] L$, making it into a topological group. This topology generalises the Krull topology, which is typically only defined for Galois extensions. Moreover, we proved that this topology is profinite whenever the extension is algebraic (but not necessarily normal or separable).

Immense thanks are due to Kevin Buzzard for his support throughout the project. He has been very generous with his time and has written many articles' worth of exposition to me via Zulip messages. More generally, the `mathlib` community has answered any and all questions posed in the Leanprover Zulip server. For anybody starting out in Lean, my top piece of advice would be to make use of this community to the fullest. As long as you are demonstrating effort, no question is too basic!



I would also like to thank Patrick Massot for helping me understand some of the technical details of filter bases and Thomas Browning for proving the `finite_dimensional_sup` lemma, as well as everyone who has commented on my Pull Requests or replied to my Zulip questions.

⁴ Which is just the esoteric way of saying that $\mathcal{N}(\sigma) \subseteq \mathcal{F}$.

References

1. Berhuy, G.: Infinite Galois theory, p. 13–25. London Mathematical Society Lecture Note Series, Cambridge University Press (2010). <https://doi.org/10.1017/CBO9781139107051.003>
2. Bourbaki, N.: General topology. Chapters 1–4. Elements of Mathematics (Berlin), Springer-Verlag, Berlin (1998), translated from the French, Reprint of the 1989 English translation
3. Conrad, K.: Infinite Galois Theory (2020), <https://ctnt-summer.math.uconn.edu/wp-content/uploads/sites/1632/2020/06/CTNT-InfGaloisTheory.pdf>
4. Dudley, R.M.: Real analysis and probability, Cambridge Studies in Advanced Mathematics, vol. 74. Cambridge University Press, Cambridge (2002). <https://doi.org/10.1017/CBO9780511755347>, revised reprint of the 1989 original
5. de Frutos-Fernández, M.I.: Formalizing the ring of adèles of a global field (2022). <https://doi.org/10.48550/ARXIV.2203.16344>
6. Howie, J.M.: Fields and Galois theory. Springer Undergraduate Mathematics Series, Springer-Verlag London, Ltd., London (2006)
7. Shatz, S.S.: Profinite groups, arithmetic, and geometry. Annals of Mathematics Studies, No. 67, Princeton University Press, Princeton, N.J.; University of Tokyo Press, Tokyo (1972)

Three Case Studies on Realms

Florian Rabe  and Franziska Weber 

University Erlangen-Nuremberg, Germany

Abstract. Realms are a proposed high-level structuring concept for theory graphs that allows abstracting over isomorphic formalizations of the same concept. But it is a relatively complex language feature, and it is difficult to assess if its practical value outweighs the cost of implementing it. We present a dataset of three case studies in the MMT system in order to evaluate to what extent realms can be achieved in a more lightweight way as a design pattern (as opposed to a heavyweight primitive feature) in existing formal systems. Our formalizations show that this approach is promising, and we identify a small set of generally useful language features that enables realm-like formalizations.

1 Introduction and Related Work

Motivation. In 2014, Carette, Farmer, and Kohlhase introduced the concept of *realms* [1] as a high-level structuring feature for a formal mathematical language (FML). The basic idea is to provide an abstraction layer at which multiple equivalent formalizations of the same mathematical concept can be identified.

For example (taken from [1]), the first-order theory of a group can be defined in terms of a multiplication operation \circ or a division operation $/$, and the resulting two theories can be connected by theory isomorphisms, e.g., mapping $/ \mapsto \lambda x, y. x \circ y^{-1}$. From the user perspective it should not matter, which of these formalizations is chosen: when using a group, both operations should be available; when creating a group, giving either operation should suffice.

Such non-trivial alternative choices are typical for formalizing mathematical concepts, and the problem identified in [1] is as pressing today as it was then. However, while the idea of realms was well-received (best paper award), neither [1] nor any follow-up work conducted a detailed investigation of how realms should be implemented in a practical FML. In particular, the relative merit between the following two options remains unclear:

- P add realms as a primitive feature in a new FML,
- E realize realms as an emergent feature in an existing FML akin to a design pattern in software engineering.

[1] appears to favour P. It defines a realm using a tuple of a set B of isomorphic base theories, one tree of conservative extensions for each base theory, and a face theory F that provides the outside interface of the realm and is realized by each base theory. But that is a very complex FML feature to specify, implement, and use in practice, especially as [1] also advocates for heavyweight realm-level

operations like translation and union. On the other hand, E would be a much simpler alternative and might allow reusing existing theory-level operations.

Contribution. We present three case studies that we have conducted to explore the feasibility of E: the realms of a lattice, a topological space, and the natural numbers.¹ Our formalizations are of moderate size (about 70 MMT modules with about 1500 lines of formalization) but carefully chosen to reveal the promise and limitation of E in practice. Our conclusion is that E becomes feasible if existing FMLs are extended with certain lightweight and independently useful features.

We use the MMT system [5] for our formalizations. But our results apply correspondingly to any system that supports theories and theory morphisms (such as Hets [2], PVS [3], or Isabelle [4]) or allows encoding them (such as any dependent type theory with record types).

2 Case Studies

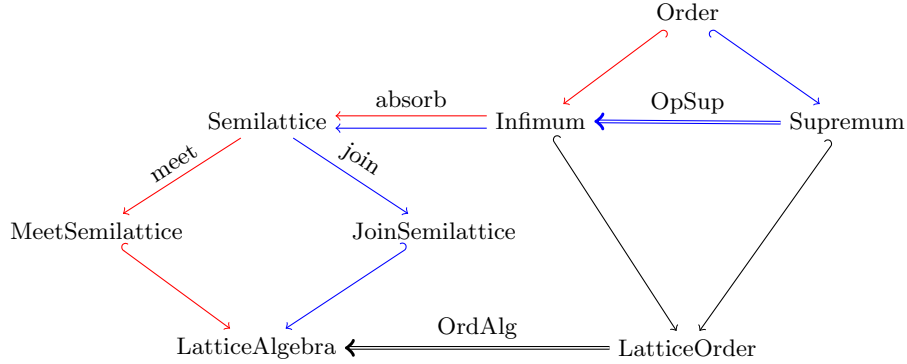
2.1 A Seemingly Simple Realm: Lattices

Formalization. We start with the theory of lattices, which can be built in two isomorphic ways: (i) based on *Semilattice* (with a binary operation \circ) imported twice using *meet* : $\circ \mapsto \sqcap$ resp. *join* : $\circ \mapsto \sqcup$, or (ii) based on an order \leq with infimum and supremum. Fig. 1 gives a high-level summary of our formalization as a diagram of theories, where we use hooked arrows for inclusion morphisms and double arrows for isomorphisms. Seen as a realm, we have two isomorphic base theories *LatticeAlgebra* and *LatticeOrder*. The face arises as their union and is thus canonically isomorphic to both.

Both theories involved are defined modularly as indicated in Fig. 1, and that significantly increases the complexity. We immediately notice a hierarchy of realms: bounded lattices (omitted from Fig. 1) form a larger, and *Semilattice* and *Infimum* a smaller realm. We also notice overlapping realms: *Semilattice* and *Supremum* form a realm as well, but *Infimum* and *Supremum* should not be in the same realm because they represent different concepts.

The isomorphism *absorb* maps $\leq \mapsto \lambda x, y. x \circ y = x$. By composing it with *meet*, we obtain the infimum operation in algebraic lattices. Correspondingly, we obtain the supremum by composing it with *join* and *OpSup* (which maps $\leq \mapsto \lambda x, y. y \leq x$). Here we encountered a subtle problem: *LatticeAlgebra* now inherits two separate implementations of *Order*: via the red morphism through *Infimum* and *meet*, and via the blue morphism through *Supremum*, *OpSup*, and *join*. It is easy to prove that these two morphisms are equal; but the equality must be *proved*. MMT, on the contrary, expects morphism to be equal *definitionally* and therefore flags an error; other FMLs are similarly limited.

¹ See <https://gl.mathhub.info/MMT/LATIN2/-/blob/devel/source/casestudies/2022-realm-case-studies.md>.

**Fig. 1.** Realm of lattices

Conclusions. The lightweight approach E scaled very well because it did not require realms as primitive objects. We believe that the rigid and complex realm structure of approach P would have gotten in the way of an elegant, modular formalization.

We identified one necessary feature: FMLs like MMT tend to support only theories and theory morphisms but not non-trivial proofs of equality between morphisms. This is challenging to implement (because it makes the structure of the diagram undecidable) but critical for even basic realm support.

2.2 A Complex Hierarchy of Realms: Topological Spaces

Formalization. Further stress-testing the approach, we moved on to topologies, see Fig. 2. In the middle, we have the realm of *Closure* consisting of 3 isomorphic theories. It is extended to the realm of *Topology* consisting of 6 isomorphic theories, 3 of which include the ones from *Closure*. For example, *OpenTop* uses the set of open sets as a primitive, whereas *ClosureTop* uses a closure operator. Fig. 2 omits the extension to the realm of continuous functions consisting of 6 corresponding isomorphic theories.

To aggregate the isomorphic theories into a single face theory, we use MMT’s feature of implicit morphisms [6]. For example, the theory *Topology* (not shown in Fig. 2) arises by including all 6 base theories and declaring the 6 isomorphisms between them as implicit. Consequently, users of *Topology* have seamless access to the operations of all base theories: the implicitness of the isomorphisms means that MMT inserts them automatically wherever needed. As for lattices above, we are limited by the missing feature of proving morphisms equal: concretely, the $//$ -marked subdiagrams commute definitionally, but the $//^?$ -marked ones require proof.

Conclusions. [1] uses a tree of extensions for each base theory. But an extension with definitions or theorems anyway yields an isomorphic theory. Therefore, we found no practical need to conceptually distinguish base theories and extensions. Instead, we found it much more convenient to formalize realms simply as groups of isomorphic theories.

Like for lattices, proving isomorphisms critically requires proving equalities of morphisms. Assuming it supports that feature, an FML can realize realms via approach E if it supports aggregating such an isomorphic group into a single face theory. We found MMT’s implicit morphisms are one elegant way to do that.

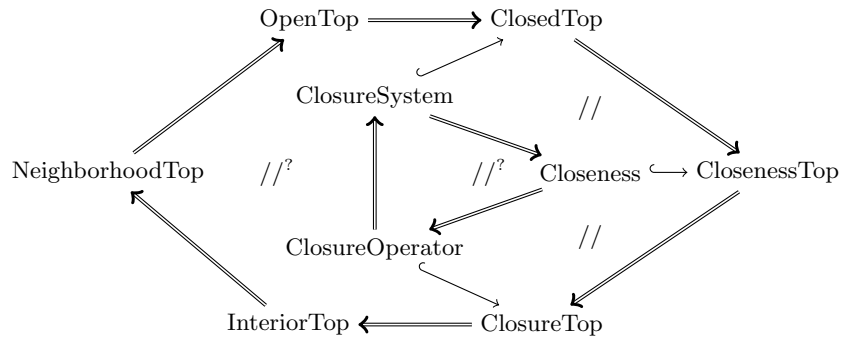


Fig. 2. Realm of topological spaces

2.3 Realms with Non-Isomorphic Theories: Natural Numbers

Formalization. [1] also proposes realms where the base theories are not isomorphic to each other. It gives natural numbers as an example where the base theories are Peano axioms, a formalization

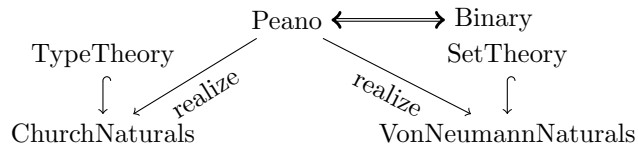


Fig. 3. Realm of natural numbers

of binary arithmetic, and realizations of natural numbers within some formalism such as Church numerals within type theory or von Neumann numerals within set theory. We formalized natural numbers with basic arithmetic operations 4 times as shown in Fig. 3.

Naturally, we only have morphisms from the axiomatic formulations to those on top of some formalism but not the other way around. Thus, these formulations are not isomorphic. For *Binary* arithmetic we chose an axiomatic formalization,

for which we could give an isomorphism to Peano’s unary arithmetic. Alternatively, we could have realized binary arithmetic on top of some formalism that provides lists of bits.

Conclusions. We encountered a surprising problem: [1] claims multiple examples of realms that are not realms according to its own definition, which requires the base theories to be isomorphic. Examples are the above realm of natural numbers or the realm of the equivalent definitions of real numbers in set theory. Here an entirely different FML concept of realm must be introduced.

3 Conclusion and Future Work

Based on our case studies, we suggest splitting the concept of realm into two concepts. The first one is a special case of the definition in [1], the second one is not covered by it.

The first kind of realm consists of a group of isomorphic theories. If an FML system supports aggregating such groups into single units, these realms can be realized elegantly. Existing FMLs like MMT are close to that already but must provide stronger support for equality of morphisms.

The second kind of realm consists of an axiomatic theory A (or multiple isomorphic ones) with outgoing morphisms $m_i : A \rightarrow F_i$ into different formalisms. Critically, these morphisms must be conservative (i.e., A -sentences s must hold in A iff $m_i(s)$ holds in F_i). Conservativity is very difficult to prove even informally and generally not well-supported by FML systems. But if conservativity can be ensured, it is easy for FML systems to use the partial inverse of a conservative morphisms to transport knowledge from F_1 to F_2 via $m_1^{-1}; m_2$.

References

1. 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.) *Intelligent Computer Mathematics - International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings. Lecture Notes in Computer Science*, vol. 8543, pp. 252–266. Springer (2014). https://doi.org/10.1007/978-3-319-08434-3_19
2. Mossakowski, T., Maeder, C., Lüttich, K.: The heterogeneous tool set, HETS. In: Grumberg, O., Huth, M. (eds.) *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings. Lecture Notes in Computer Science*, vol. 4424, pp. 519–522. Springer (2007). https://doi.org/10.1007/978-3-540-71209-1_40
3. Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system. In: Kapur, D. (ed.) *Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, June 15-18, 1992, Proceedings. Lecture Notes in Computer Science*, vol. 607, pp. 748–752. Springer (1992). https://doi.org/10.1007/3-540-55602-8_217

4. Paulson, L.C.: Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow), Lecture Notes in Computer Science, vol. 828. Springer (1994). <https://doi.org/10.1007/BFb0030541>, <https://doi.org/10.1007/BFb0030541>
5. Rabe, F., Kohlhase, M.: A scalable module system. *Inf. Comput.* **230**, 1–54 (2013). <https://doi.org/10.1016/j.ic.2013.06.001>
6. Rabe, F., Müller, D.: Structuring theories with implicit morphisms. In: Fiadeiro, J.L., Tutu, I. (eds.) *Recent Trends in Algebraic Development Techniques - 24th IFIP WG 1.3 International Workshop, WADT 2018, Egham, UK, July 2-5, 2018, Revised Selected Papers*. Lecture Notes in Computer Science, vol. 11563, pp. 154–173. Springer (2018). https://doi.org/10.1007/978-3-030-23220-7_9

Setting up Set-Theoretical Foundations in Naproche

Marcel Schütz² , Adrian De Lon¹ , and Peter Koepke¹ 

¹ University of Bonn, Germany

<https://www.math.uni-bonn.de/ag/logik/>

² TU Darmstadt, Germany

Abstract. Bringing out the potential of interactive theorem provers requires efficient mathematical foundations. The current release of the natural language proof assistant Naproche has become sufficiently stable to allow a broader and principled approach towards libraries of basic mathematical material. We present Naproche’s new ontology of objects, classes, maps, etc. and two foundational libraries about basic set theory and number theory. These foundations are then used, e.g., in a formalization of the Cantor–Schröder–Bernstein theorem.

1 Introduction

The Naproche (Natural Proof Checking) proof assistant [12] is being developed for a high degree of naturalness of accepted proof texts and of its user interaction [3, 8, 10]. It is available as a component of the Isabelle prover platform [6]. Naproche supports prototypical formalizations of university-level material in a natural mathematical language and style that is immediately readable by mathematicians. Naproche comes with a number of chapter-sized example formalizations which also comprise the foundational files discussed in this paper.

Most Naproche formalizations thus far are self-contained texts that introduce their own axiomatic environments and lead up to individual well-known theorems. Ad-hoc axiomatic set-ups allow elegant mathematical “miniatures” with a minimal number of axioms, but the consistency and compatibility of axiomatic assumptions pose problems. Moreover, duplicate definitions cause ambiguities and reproving basic theorems is inefficient.

In this paper we describe steps towards a next level of formalizations with fixed ontological foundations and libraries of basic material. The current release of Naproche has a built-in language for classes, objects, and maps within a Kelley–Morse-like set theory [7, 13]. Some Kelley–Morse axioms are fixed in foundational files whereas further axioms, like the axiom of infinity can be postulated by a user if needed. The current Naproche release includes two example libraries on sets and on integers that are then used to formalize more specialized results.

2 Intuitive Mathematical Ontologies

The point of departure for *natural* formal mathematics and *natural* proof assistants is the language that is actually employed in contemporary mathematical publications, combining argumentative natural language and symbolic terms. So far, the language of mathematics has been subject to only a small number of linguistic investigations [5,1]. Like ordinary language, this language is ambiguous and incomplete, but mathematicians use it as an efficient means of specialized communication.

The language of mathematics indicates possible or intended ontologies by its choice of natural language words and phrases and by their grammatical categories: e.g.,

- definite nouns denote specific objects or individua of some universe, whether such a universe is taken for real or a convenient mental construct;
- common nouns denote collections, classes, sets, sorts, or types of (related) objects;
- these collections can be modified and specified by adjectives or relative sentences.

“Aristotle” is a distinct individuum which belongs to the collection of “humans”; the common noun “human” can be modified by adjectives like “mortal”. Similarly “zero” denotes a constant which is an element of the “natural numbers”; natural numbers can, e.g., be “even” or “odd”. Symbolic notation is used for brevity and exactness: “0” is used for “zero” and “ \mathbb{N} ” for the collection of natural numbers.

Collections are intuitively expressed as the “collection, class, type, set, ... of all x such that ϕ ” or by *comprehension terms* like

$$\{x \mid \phi\} \text{ or } \{x : \phi\}$$

where ϕ is a mathematical statement. Often such collections are considered to be objects themselves. One can, e.g., say that \mathbb{N} is (an object that is) infinite.

This brief impression suggests ontologies of mathematical objects and collections of objects, where the distinctions between various categories and their properties are somewhat fluid, similar to other natural language ontologies.

From a type-theoretic perspective (mathematical) common nouns like “number” or “rectangle” can be interpreted as types. Since common nouns do not exactly satisfy the laws of mathematical type theories, one speaks of “soft” types [18]. The properties of soft types lie between sets and types: there are set-theoretical properties like inclusions of types (“natural numbers” form a subtype or subset of all “numbers”). On the other hand natural language quantifications are usually bounded by types: we say “every integer ...” instead of “for every n , if n is an integer then ...”. A computer-processable natural language for mathematics should use soft types to denote collections of mathematical objects.

3 The Naproche System

The Naproche (Natural Proof Checking) proof assistant [12] is available as a component in the latest release of the Isabelle prover platform [6] for Linux, macOS, and Windows. We have also built a simple web interface for a quick start without installation or high system requirements [17]. Naproche supports prototypical formalizations of university-level material in a natural mathematical language and a style that is immediately readable by mathematicians.

The Naproche system uses the controlled natural language ForTheL (Formula Theory Language) as its input language which is designed to closely approximate common constructs of the language of mathematics [16]. At the same time ForTheL is a completely formal language which allows its translation into formal logics and further processing by automated theorem provers or checkers. ForTheL has been developed over several decades and is an outgrowth of the Evidence Algorithm project [11].

ForTheL handles soft types as “notions”, and it provides various methods for the introduction and processing of notions. Notions can be viewed midway between sets, classes, and types. Naproche translates notions into first-order predicates which can be used to form type guards. The language of a ForTheL text is fixed by signature commands and definitions. A language for the additive structure of the natural numbers can be introduced by the commands:

```
[synonym number/numbers]
```

Signature. A natural number is a notion.

Signature. 0 is a natural number.

Signature. Assume that m, n are natural numbers. $m + n$ is a natural number.

ForTheL may be embedded into L^AT_EX documents; for the example above one could enter the following into a `.ftl.tex` document in Isabelle/jEdit:

```
\begin{forthel}
  [synonym number/numbers]
  \begin{signature}
    A natural number is a notion.
  \end{signature}
  \begin{signature}
    $0$ is a natural number.
  \end{signature}
  \begin{signature}
    Assume that $m,n$ are natural numbers.
    $m+n$ is a natural number.
  \end{signature}
\end{forthel}
```

After fixing a bit of vocabulary (“number” and its plural “numbers” can be used synonymously in the sequel) there are three L^AT_EX signature environments using the usual combination of natural language and symbols. The first-order meaning of the commands can be seen by mouse-hovering over the jEdit buffer. The noun phrase “natural number” corresponds to a newly introduced unary predicate symbol `aNaturalNumber()`. The second command introduces the internal constant symbol `0` and translates to the (tagged) first-order formula

$$\forall v_0. ((\text{HeadTerm} :: v_0 = 0) \rightarrow \text{aNaturalNumber}(v_0))$$

which is logically equivalent to `aNaturalNumber(0)`. Finally the introduction of the addition symbol `+` is translated to the following first-order formulas:

$$\frac{\text{aNaturalNumber}(m) \wedge \text{aNaturalNumber}(n)}{\forall v_0. ((\text{HeadTerm} :: v_0 = m + n) \rightarrow \text{aNaturalNumber}(v_0))}$$

These formulas also express that addition has the type $+: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$.

Note that the first-order approach to types yields a very flexible *dependent* type system where number systems can be cumulative ($\mathbb{N} \subseteq \mathbb{R} \subseteq \mathbb{C}$), and notions can depend on parameters (“subsets of \mathbb{N} ”, “divisors of n ”).

4 Sets and Classes in Naproche

As Naproche is directed at common mathematical usage, comprehension terms in verbal and symbolic form are provided by ForTheL. One can, e.g., define

Definition. $\mathbb{N} = \{ x \mid x \text{ is a natural number} \}$.

or, verbally,

Definition. \mathbb{N} is the collection of all natural numbers.

Naproche translates these definitions into an internal first-order format which will eventually be passed to external ATPs. Therefore a treatment of comprehension terms in the sense of standard set or class theories is hard-coded into Naproche. In line with Kelley–Morse class theory with urelement (KMU) [13] comprehension terms are registered as *classes* which consist of “small” *objects*.

Thus Naproche translates our definition to the first-order formula

$$\forall v_0. (v_0 = \mathbb{N} \leftrightarrow \text{aClass}(v_0) \wedge \forall v_1. (v_1 \in v_0 \leftrightarrow \text{aObject}(v_1) \wedge \text{aNaturalNumber}(v_1)))$$

In the Naproche implementation of KMU we have objects, elements, sets and classes – note that Naproche allows to use the terms ‘class’ and ‘collection’ synonymously – satisfying:

- every element of a class is an object;
- a set is a class that is an object.

These notions with some elementary properties are hard-coded into Naproche. Moreover we provide notions of functions and maps, which in close analogy with sets and classes behave as follows:

- the application of a map to an object in its domain is an object;
- a function is a map which is an object;
- these notions are kept abstract, but behave like the usual set-theoretic encoding of functions and maps as graphs.

4.1 Kelley–Morse-like Foundations in Naproche

Naproche’s built-in vocabulary and mechanisms for classes and maps do not form a sufficient basis for formalizations. Formalizations in earlier Naproche versions each defined their own additional axioms as preliminaries: for instance, one would find multiple and even slightly different definitions of “subset” across different formalizations.

We have replaced such ad-hoc preliminaries with a common shared file `preliminaries.ftl.tex`, which describes a weak fragment of KMU. It expands on Naproche’s built-in notions with some common axioms (e.g. extensionality) and defines general notions such as “subclass”. Besides some minor technical definitions and axioms we require:

Axiom (Class Extensionality). If S is a subclass of T and T is a subclass of S then $S = T$.

Axiom (Separation Axiom). Assume that X is a set. Let T be a subclass of X . Then T is a set.

Axiom (Functional Extensionality). Assume $\text{dom}(f) = \text{dom}(g)$ and for every $x \in \text{dom}(f)$ $f(x) = g(x)$. Then $f = g$.

Axiom (Replacement Axiom). Let X be a set. Assume that X is a subset of the domain of f . Then $f[X]$ is a set.

Here $f[X]$ denotes the image of X under f . Although the axioms of separation and replacement are potentially powerful axioms, the theory described in `preliminaries.ftl.tex` constitutes a relatively weak theory since it includes neither the axiom of infinity nor the axiom of powersets.

4.2 Applications

The preliminaries file is used in some of the example formalizations that come with Naproche. The mutilated checkerboard problem is only concerned with finite sets of checkerboard positions and dominoes [2]. Thus we do not need the axiom of infinity and the weaker class theory of `preliminaries.ftl.tex` suffices.

Our example file `numbers.ftl.tex` on the number systems $\mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}$ in contrast demands that the various infinite collections of numbers are sets. This is expressed axiomatically by

Axiom. \mathbb{R} is a set.

In standard set theory this could be proved by the axiom of infinity which says that \mathbb{N} is a set, and then one could construct the set \mathbb{R} from \mathbb{N} by forming the powerset of \mathbb{N} together with other, weaker set construction principles.

5 Foundational Libraries in Naproche

In addition to the `preliminaries.ftl.tex` file, Naproche also provides two larger libraries which cover foundational formalizations about basic set theory and arithmetic. In contrast to previous formalizations as stand-alone documents, this is a first systematic attempt to design a collection of formalizations which are primarily intended to be read in other ForTheL texts. The two libraries are organized as L^AT_EX documents with a book-like chapter structure which are written in a literary style with comments around the formalized core³. Importing their contents can be done chapter by chapter so that one is not forced to import the whole library when one is just interested in reusing some specific definitions or theorems they provide. The libraries are designed towards a high degree of human readability which aims to be a step towards mathematical texts being accessible both to computers and to mathematicians unfamiliar with texts written in synthetic formal languages. Besides being used as a foundation for more ambitious formalizations in Naproche, the libraries can be regarded as a presentation of formally verified undergraduate mathematics in the style of, e.g., the Stacks Project [15].

5.1 A Foundational Library for Sets

The set theory library⁴ is a self-contained introduction to a Zermelo–Fraenkel-like set theory, without the axiom of infinity, embedded into Naproche’s built-in fragment of KMU. It states the usual ZF axioms, defines common operations on sets, like unions, intersections and complements, and provides detailed proofs of their algebraic properties. Moreover, the notion of functions between sets is introduced together with basic properties like being one-to-one or invertible. As with sets, common operations on functions are defined, e.g., composition, restriction, images and preimages. Their algebraic properties are proved in detail as well. Finally, the notion of equinumerosity is introduced.

The material in the library is presented in a naturally readable format, as demonstrated for instance by the formulation of the Knaster–Tarski fixed point theorem:

³ See e.g., the PDF-printout `$ISABELLE_HOME/contrib/naproche-20212111/examples/set-theory/set-theory.ftl.pdf`

⁴ Located in `$ISABELLE_HOME/contrib/naproche-20212111/examples/set-theory`

Theorem 11.5 (Knaster-Tarski). Let h be a function from $\mathcal{P}(x)$ to $\mathcal{P}(x)$ that preserves subsets. Then h has a fixed point.

In this example, $\mathcal{P}(x)$ denotes the powerset of x , where x has been pretyped as a set earlier in the document.

In contrast to the foundational material in the file `preliminaries.ftl.tex` presented above, which puts a rather strong emphasis on the notion of classes and maps, the primary notions of this library are sets and functions. Classes and maps only play a rather technical role in the library, for instance by using them to formulate statements which in ZF could only be formulated as axiom *schemas*. The decision to follow a more *set*-theoretical approach in the sense of Zermelo–Fraenkel in contrast to a more *class*-theoretical one according to Kelley–Morse is motivated by the lower degree of complexity of the ontology in a setting where just sets and functions play a major role compared to a setting where in addition to them also classes and maps are involved on an equal footing and not just as technical helpers.

The first chapter of the library introduces the notion of subsets and states the axioms of *set extensionality*, *separation*, *set existence*, *pairing* and *union*. Moreover, unions, intersections and complements are defined and some of their algebraic properties are proved. To state the axiom of separation, which in ZF would actually be an axiom *schema*, we make use of Naproche’s built-in notion of classes together with the built-in axiom schema of *class comprehension*. Just as we postulated *separation* and *replacement* in the above file `preliminaries.ftl.tex`, we again formulate:

Axiom 1.9 (Separation). Let C be a collection and x be a set. Assume that every element of C is contained in x . Then C is a set.

(Note that the term *collection* is just a built-in synonym for *class*.) Using this axiom to define a set y which contains all elements of a given set x satisfying a formula φ , we can proceed as follows. First, by class comprehension, we can define y as the *class* $\{ u \in x \mid \varphi(u) \}$ and then, by the separation axiom, we can show that y is actually a set. As an example of this procedure, consider the proof of the existence of relative complements:

Lemma 1.46. Let x, y be sets. There exists a set z such that $z = \{ w \mid w \in x \text{ and } w \notin y \}$.

Proof. Define $z = \{ w \mid w \in x \text{ and } w \notin y \}$. Then every element of z is contained in x . Hence z is a set (by Separation). \square

Here we use Naproche’s built-in mechanism of defining classes via comprehension terms which serves as an implementation of the mentioned axiom schema of class comprehension. This way we define a class z which is then shown to be actually a set by referencing to the separation axiom (for more on this referencing mechanism see below).

Whereas Naproche allows to define classes of the form $\{u \mid \psi(u)\}$ or $\{u \in x \mid \psi(u)\}$ (where the latter is just an abbreviation for $\{u \mid u \in x \wedge \psi(u)\}$) for arbitrary formulas ψ , it is currently not possible to directly refer to formulas in a ForTheL statement as if they were objects because of the first-order nature of ForTheL. Thus the common ZF-like formulation of the separation axiom as “Let ψ be a formula. Then for any set x there exists a set y such that $y = \{u \in x \mid \psi(u)\}$ ” would not be valid ForTheL.

The following chapters of Part I of the library introduce the *powerset axiom* (chapter 2), the *axiom of regularity* (chapter 3), the symmetric difference and its algebraic properties (chapter 4), ordered pairs via Kuratowski’s definition (chapter 5) and finally Cartesian products (chapter 6). In this last chapter the Cartesian product of two sets x and y is defined as follows:

Definition 6.2. $x \times y$ is the set z such that $z = \{(u, v) \mid u \in x \text{ and } v \in y\}$.

To ensure that this definition is well-formed, i.e. that a unique such set z actually exists, it is preceded by the following lemma granting the existence of z – uniqueness follows immediately from Naproche’s built-in extensionality axiom for classes.

Lemma 6.1. There exists a set z such that

$$z = \{(u, v) \mid u \in x \text{ and } v \in y\}.$$

Within Naproche’s ontology, we could also use a slightly different way of defining the Cartesian product. Namely, we could define $x \times y$ as the class $\{(u, v) \mid u \in x \text{ and } v \in y\}$ and show afterwards that this class is actually a set. Whereas this order of first defining an object and proving afterwards that it is of a certain type is the more common approach in everyday-mathematics, we decided to do it the other way around (i.e. first proving that an object of a certain type exists and afterwards defining something to be this object). This ensures that the assertion that the Cartesian product of two sets is again a set is already part of the definition which makes the proof search in Naproche a little bit more efficient.

Part II of the library begins with a chapter on functions (chapter 7). It states the *axiom of replacement*, introduces the notions of injectivity, surjectivity, bijectivity, identity function and constant functions and defines composition and restriction operations. For instance, the axiom of replacement, like the axiom of separation in ZF only expressible as an axiom schema, is formulated using the notion of maps:

Axiom 7.8 (Replacement). Let f be a map and x be a set. There exists a set y such that $y = \{f(u) \mid u \in \text{dom}(f) \text{ and } u \in x\}$.

The remaining chapters of Part II deal with images and preimages of functions (chapter 8), invertible functions (chapter 9), the behaviour of the symmetric difference under images and preimages (chapter 10), subset preserving functions

and the Knaster–Tarski fixed point theorem (chapter 11) and finally the notion of equinumerosity (chapter 12).

Similar to Naproche’s mechanism for defining classes in a proof, there is also a built-in mechanism for defining maps. For instance consider the following proof of the existence of the preimage of a set z under some function f – note that z was pretyped as a set earlier in the formalization:

Lemma 8.9. Let f be a function. There exists a set y such that $y = \{ u \in \text{dom}(f) \mid f(u) \in z \}$.

Proof.

⋮

Case $f(u) \notin z$ for some $u \in \text{dom}(f)$. Take $w \in \text{dom}(f)$ such that $f(w) \notin z$. Define

$$g(u) = \begin{cases} u & : f(u) \in z \\ w & : f(u) \notin z \end{cases}$$

for $u \in \text{dom}(f)$.

⋮

Take $y = \text{range}(g) \setminus \{ w \}$. Then $y = \{ u \in \text{dom}(f) \mid f(u) \in z \}$. End. \square

Here we define a map g on the set $\text{dom}(f)$ which Naproche can show on its own to be a function. The range of this function g is then used to define a set y which can be expressed by the comprehension term in the assertion of the lemma.

5.2 A Foundational Library for Natural Number Arithmetic

The arithmetic library⁵ is a self-contained elaboration of Peano Arithmetic with addition, multiplication, exponentiation and factorial with detailed proofs of their arithmetic properties. It provides the standard ordering on the natural numbers and proves various rules about its interplay with the arithmetical operations. Based on this ordering, some variants of induction are provided and also a (partial) subtraction operation. The library defines divisibility, Euclidean division and modular arithmetic. Finally, the notion of prime numbers is given together with proofs of some basic number-theoretic facts.

A typical example of a theorem provided by the library is its formulation of the Euclidean division theorem:

Proposition 14.1. For all natural numbers n, m such that m is nonzero there exist natural numbers q, r such that $n = (m \cdot q) + r$ and $r < m$.

Chapter 1 of the library introduces the notion of natural numbers as a new sort of objects together with a constant ‘0’ and a unary successor operation ‘succ’.

⁵ Located in `$ISABELLE_HOME/contrib/naproche-20212111/examples/arithmetic`

independent of any other foundational formalizations. In contrast to setting up Peano Arithmetic on the basis of, e.g., the set theory library presented above, this stand-alone approach was chosen to avoid potential efficiency drawbacks concerning the verification in Naproche which a large overhead of formalizations it would otherwise depend on might cause.

The structure of the natural numbers is characterized by the following three axioms:

Axiom 1.5 (1st Peano axiom). If $\text{succ}(n) = \text{succ}(m)$ then $n = m$.

Axiom 1.6 (2nd Peano axiom). 0 is not the direct successor of any natural number.

Axiom 1.7 (3rd Peano axiom). Let P be a class. Assume $0 \in P$ and for all natural numbers n we have $n \in P \implies \text{succ}(n) \in P$. Then every natural number is an element of P .

(Note that in the first axiom, n and m are pretyped as natural numbers and in the second one, “the direct successor of ...” is just a synonym for “ $\text{succ}(\dots)$ ”). As in the set theory library we used Naproche’s built-in notion of classes to formulate the induction schema of Peano Arithmetic.

The following chapters of Part I of the library provide some operations on the natural numbers, namely addition (chapter 2), multiplication (chapter 3), exponentiation (chapter 4) and factorial (chapter 5), together with detailed proofs of their computation laws. These operations are all introduced axiomatically, e.g.:

Signature 4.1. n^m is a natural number.

Axiom 4.2 (1st exponentiation axiom). $n^0 = 1$.

Axiom 4.3 (2nd exponentiation axiom). $n^{m+1} = n^m \cdot n$.

Of course, every time a new operation is introduced this way, the axiom system of the whole formalization is extended which increases the potential of accidentally getting some inconsistency the larger this system grows. Thus in the further development of this library, more robust foundations for Peano Arithmetic will be evaluated, for instance by merging it into the set theory library, which would allow to define all such operations on the basis on Dedekind’s recursion theorem.

Part II of the library introduces the standard ordering on the natural numbers (chapter 6) and presents some facts about the interplay between this ordering and the addition, multiplication and exponentiation operations (chapters 7, 8, 9, resp.). Moreover, some equivalent formulations of the induction axiom are given (chapter 10), followed by a chapter on standard exercises in natural number arithmetic (chapter 11). Finally, chapter 12 introduces a (partial) subtraction operation on the natural numbers.

Part III deals with the divisibility of natural numbers. This notion is defined in chapter 13, together with proofs of its basic algebraic properties. Chapter 14 presents a proof of the possibility of Euclidean division within the natural

numbers, on the basis of which modular arithmetic is introduced in chapter 15. Finally, in chapter 16 the notion of prime numbers is introduced together with some basic facts from number theory.

5.3 Usage in Other Formalizations

The two described libraries are used to serve as a foundation for a formalization of a proof of the Cantor–Schröder–Bernstein theorem, i.e. the assertion that two sets are equipollent if they can be embedded into each other, and of Furstenberg’s proof of the infinitude of primes [4].

The former ⁶ is based on a version of Knaster’s proof of the Cantor–Schröder–Bernstein theorem as stated in [14]. It uses Naproche’s `readtex` instruction to import some chapter (and its dependencies) of the set theory library:

```
[readtex set-theory/sections/02.functions/06_equipollency.ftl.
tex]
```

(To avoid confusion: Naproche provides two instructions for importing files, `read` and `readtex`. The former is used to import files in the `.ftl` format, whereas the latter is used for files in the `.ftl.tex` format.) Since Naproche’s import functionality is very limited, there is no option for only importing those definitions and theorems we need to state and prove the Cantor–Schröder–Bernstein theorem. Instead we have to import a full chapter of the set theory library. Having imported a chapter of a library, we can use L^AT_EX’s `\ref` command to cite any definition, theorem, etc. it provides for the proof of the Cantor–Schröder–Bernstein theorem. For instance after a certain function h is defined which is shown to be subset-preserving, the proof states the existence of a fixed point c of h by referring to theorem 11.5 of the set theory library:

Hence we can take a fixed point c of h (by 11.5).

The L^AT_EX source of this line looks as follows:

```
Hence we can take a fixed point  $c$  of  $h$  (by \ref{
SetTheory_02_05_636019}).
```

The L^AT_EX command `\ref{SetTheory_02_05_636019}` creates a clickable link which points to the theorem with label `SetTheory_02_05_636019` (which is theorem 11.5) in the PDF file of the set theory library.

6 Further Plans

Our approach to libraries is exploratory and so far limited by Naproche’s rudimentary `read` instruction. We are working on an import mechanism that makes

⁶ see `$ISABELLE_HOME/contrib/naproche-20212111/examples/cantor-schroeder-bernstein.ftl.tex`

it possible to build up a graph of theories and provides more control over the visibility of theorems. We are also experimenting with options for natural language syntax and the reuse of common L^AT_EX commands (e.g. `\cite`) for theory imports.

Presently importing mathematical structures and associated material from other formalizations is rather inflexible. Although Naproche can deal with structures (see e.g. [9]), they can only be imported “verbatim”. For example, changing a group operation from \times to \star requires cumbersome notation and incurs a considerable amount of overhead (making proof automation less effective). A logical next step is a ForTheL/Naproche implementation of mathematical structures similar to Isabelle’s locales which shall address these issues and significantly improve reusability of future formalizations.

Since Naproche formalizations omit many details and heavily rely on proof automation, checking speeds have always been slower than established proof assistants. Running Naproche can be compared to continuously invoking Sledgehammer for every explicit and implicit claim in a document. We are working on more robust local caching and efficient usage of external provers so that Naproche remains usable on mid-range hardware.



Performance problems have also been exacerbated by basing formalizations on general libraries and moving towards richer ontologies. While Kelley–Morse class theory with urelements is a strong and expressive foundation for mathematics, we have also noticed some trade-offs compared to ZF. Presently first-order formulas exported to ATPs are burdened with type guards for classes and elements, making proof search more difficult. In practice this meant that Naproche proofs had to become more detailed and some previously working proofs had to be refactored. We also observed that proof obligations stemming from the class-set-distinction (e.g. having to prove that a certain comprehension forms a set) can be a major stumbling block for students working with Naproche.

References

1. Avigad, J.: Mathematics and language. In: Davis, E., Davis, P.J. (eds.) *Mathematics, Substance and Surmise: Views on the Meaning and Ontology of Mathematics*, pp. 235–255. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-21473-3_12
2. De Lon, A., Koepke, P., Lorenzen, A.: A natural formalization of the mutilated checkerboard problem in Naproche. In: Cohen, L., Kaliszyk, C. (eds.) *12th International Conference on Interactive Theorem Proving (ITP 2021)*. *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 193, pp. 16:1–16:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). <https://doi.org/10.4230/LIPIcs.ITP.2021.16>
3. Frerix, S., Koepke, P.: Automatic proof-checking of ordinary mathematical texts. *Proceedings of the Workshop Formal Mathematics for Mathematicians* (2018)
4. Furstenberg, H.: On the infinitude of primes. *The American Mathematical Monthly* **62**(5), 353–353 (1955)

5. Ganesalingam, M.: The Language of Mathematics - A Linguistic and Philosophical Investigation, Lecture Notes in Computer Science, vol. 7805. Springer (2013). <https://doi.org/10.1007/978-3-642-37012-0>
6. Isabelle contributors: The Isabelle2021-1 release (December 2021), <https://isabelle.in.tum.de/website-Isabelle2021-1/index.html>
7. Kelley, J.L.: General Topology. Springer New York (1955)
8. Koepke, P.: Textbook mathematics in the Naproche-SAD system. Joint Proceedings of the FMM and LML Workshops (2019)
9. Koepke, P., Penquitt, J., Schütz, M., Sturzenhecker, E.: Formalizing foundational notions in Naproche-SAD. In: NFM 2020 - Workshop on Natural Formal Mathematics (at CICM 2020) (2020)
10. Lon, A.D., Koepke, P., Lorenzen, A., Marti, A., Schütz, M., Wenzel, M.: The Isabelle/Naproche Natural Language Proof Assistant. In: Platzzer, A., Sutcliffe, G. (eds.) Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12699, pp. 614–624. Springer (2021). https://doi.org/10.1007/978-3-030-79876-5_36
11. Lyaletski, A.V., Verchinine, K.: Evidence Algorithm and System for Automated Deduction: A retrospective view. In: Autexier, S., Calmet, J., Delahaye, D., Ion, P.D.F., Rideau, L., Rioboo, R., Sexton, A.P. (eds.) Intelligent Computer Mathematics, 10th International Conference, AISC 2010, 17th Symposium, Calculemus 2010, and 9th International Conference, MKM 2010, Paris, France, July 5-10, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6167, pp. 411–426. Springer (2010). https://doi.org/10.1007/978-3-642-14128-7_35
12. Naproche on GitHub, <https://github.com/naproche/naproche>
13. Rubin, J.E.: Set Theory for the Mathematician. Holden-Day, 1st edn. (1967)
14. Schröder, B.S.W.: The fixed point property for ordered sets. Arabian Journal of Mathematics **1**, 530 (2012). <https://doi.org/10.1007/s40065-012-0049-7>
15. The Stacks project, <https://stacks.math.columbia.edu/>
16. Verchinine, K., Paskevich, A.: ForTheL — The Language of Formal Theories. International Journal of Information Theories and Applications **7**(3), 120–126 (2000)
17. Naproche-Webinterface, <https://naproche.github.io/#/>
18. Wiedijk, F.: Mizar’s soft type system. In: Schneider, K., Brandt, J. (eds.) Theorem Proving in Higher Order Logics, 20th International Conference, TPHOLs 2007, Kaiserslautern, Germany, September 10-13, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4732, pp. 383–399. Springer (2007). https://doi.org/10.1007/978-3-540-74591-4_28

OEIS Semantified – A Tetrapodal Dataset

Michael Wagner  and Michael Kohlhase 

Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstraße 3, 91058 Erlangen, Germany
`michael.kohlhase@fau.de`

Abstract. The OEIS is an important resource for mathematicians. The database is well-structured and rich in mathematical content but is informal in nature, so interaction with the database is restricted to humans supported by basic search services.

In this paper, we provide the result of semantifying – i.e. recovering machine-actionable representations – the OEIS data as an open dataset. We hope that this dataset will stimulate the development of novel user interfaces, derived data, connections to other mathematical datasets and libraries, and advance knowledge management services for the mathematical community.

Keywords: OEIS · machine-actionable · tetrapod model · dataset.

1 Introduction

The OEIS (On-line Encyclopedia of Integer Sequences) is a community driven effort for curating and collecting knowledge about integer sequences. Applications range from being able to access information about a specific graph problem while researching the lifetime of multicomponent models in engineering [8], to treating the search for new sequences as an entertaining puzzle [11]. Interested parties can submit their own sequences and contribute to existing sequences. Submissions are manually reviewed and upon acceptance, the new sequence receives a new unique identifier: an A-number. Behind each A-number, of which there are over 350000, is an entry, containing mathematical knowledge about that sequence. This includes, but is not limited to, the name, sequence values, generating functions, references, scripts and comments. The data is saved internally as a text file, with each line starting with an identifier, that defines the lines' category. A line from the comments section starts with "%C", continues with the comment and ends in a line break. It is not obvious which lines belong to each other, something a human can recognize by context, a machine less so. Furthermore, the line content is only partially structured, depending on the category it originates from.

Over the last decade, our research group has invested considerable effort on extracting machine-actionable representations of the OEIS data. This paper makes the fruit of this labor available to the mathematical community. The semantized OEIS data can be used to derive new insights into integer

sequences. For instance, the semantized generating functions (otherwise available only as strings in the OEIS) led to the automated discovery of ≥ 300000 relations between sequences, most of them previously unknown [9] (and still unpublished in the OEIS). Note that the generation of the relations was the work of two afternoons with SageMath [10], whereas the semantization effort involved weeks/months of manual parser optimization. This shows the value of a curated dataset like the one presented in this paper.

The OEIS dataset is also interesting from another angle: In contrast to other data, that typically consist of one or two aspects, it contains content from all five aspects of mathematical knowledge postulated by the tetrapod model in [6].

The tetrapod model proposes that “doing math” involves five aspects (**computation**: typically programs or program fragments, **tabulation**: concrete data, typically sequences, matrixes, time series; **inference**: deduction, typically proofs; **narration**: typically text fragments or document structure fragments (e.g. sections, chapters, ...); **organization**: relational data, usually RDF triples jointly describing a graph).

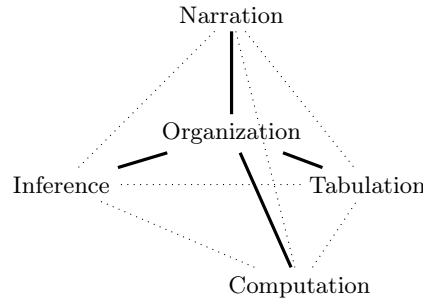


Fig. 1. A visual representation of the Tetrapod model.

In the OEIS we have

- *computational content* in the form of programs that compute the sequences,
- *tabulated content* in the form of concrete initial prefixes of the sequences,
- *inferential content* in the form of generating functions,
- *narrative content* in the form of descriptions, and
- *organizational content* in the form of curated identifiers, author/publication references, etc.

The tetrapodal structure makes the curation of the semantified OEIS dataset into a test case of the tetrapod model and the potential basis of coming tetrapodal services like the multi-aspect search.

2 OEIS Semantization and Dataset Format

In [12], a harvesting of the OEIS and semantification pipeline for the OEIS data is established. This iterates through all entries and extracts machine-actionable data. Depending on the kind of mathematical data represented in the OEIS, the pipeline deploys a different harvester/semantics extraction approach. We will discuss the semantization process and the format of the resulting dataset by the aspects in the tetrapod model.

The OEIS dataset (THODS) (Tetrapodally Harvested OEIS DataSet) is a zip file with six subdirectories, five for each of the tetrapodal aspects and a source folder. When necessary, a subdirectory is further subdivided into folders to avoid overloading the filesystem with over 350000 files in one folder.

Computational content. The OEIS contains well-structured scripts and programs that compute values for many of the sequences. In contrast to the other tetrapodal aspects (tabulation, inference, narration and organization), the OEIS representation contains an additional prefix in the form of the programming language’s name in brackets. Using this syntax, a harvester can differentiate which lines belong semantically together. The challenge of machine actionability lies in not knowing how to execute these programs. The first difficulty is that the version of their language and libraries is not explicitly stated. The second challenge concerns the intent of use: some scripts print their sequence to the console, others provide a function, which takes an index and returns the sequence value at that index. The initial step in [12] was to extract metadata. This includes the author, the date of the last change and the standard file extension of the script’s language.

In the OEIS Dataset, computational content is saved as XML-files named after the sequences A-number, e.g. the path of the Fibonacci numbers (A000045) is `/computation/A000/A000045.xml`. A single file holds all code snippets that can be found in the sequence plus the (extracted) metadata; namely:

- The “*author*” element, that contains the author’s name. The name is extracted by searching for the underscores that encompass an author’s name in the internal OEIS format.
- “*language_tag*” holds the OEIS script prefix. It states the programming/scripting language of the code snippet.
- Inside the “*file_ending*” element is the filename extension specific to the language given in the “*language_tag*”. It is generated via a manually generated conversion table.
- The content of the “*date*” is the date of the last modification. Since the date is often written in proximity to the author’s name, the harvester starts its search for it with the name as a starting point.
- Inside the “*source_code*” XML-element is the code snippet itself. As stated above, the way how it generates the sequence may vary.

Tabulated Content. is the simplest aspect in the OEIS. The first n integers of a sequence are split into three lines there, each with its own prefix flag. We join them into a single sequence, that is saved as a CSV-file, where each line is in the format *index, element of sequence at index*.

Inferential Content. The OEIS contains generating functions, which are defined as “a formal power series of some given form that generates a sequence” [4]. This dataset focuses on harvesting two types of generating functions:

1. “*direct generating functions*” – expressions that return the value of a sequence a at index n (possibly in relation to other sequences), e.g. $a(n) = A000688(n) + A060689(n)$
2. “*ordinary and exponential generating functions*” – algebraic expressions $g(x)$ in terms of a single variable x , such that the coefficients of the Taylor expansion of $g(x)$ “generate” the sequences.
 - (a) ordinary generating functions are of the form $G_{\{a_n\}}(x) = \sum_{n=0}^{\infty} a_n x^n$
 - (b) exponential ones are of the form $E_{\{a_n\}}(x) = \sum_{n=0}^{\infty} a_n \frac{x^n}{n!}$.

The OEIS contains additional types of generating functions that are not yet available in this dataset, since we started on the types with the most occurrences first. As these are finite representations of the infinite sequences, they are an important tool for the theoretical analysis and thus a prime target for semantization. Unfortunately, the OEIS sources represent the expressions in a syntactically arbitrary form and mixes them with additional information like author, comments or definitions without separators. To extract the generating functions themselves our semantifier must (heuristically) parse all entries, that may contain functions: We use a lexicon of allowed words and grammatical rules on how to combine them. To enhance the quality of the dataset, parsed functions are validated by importing them into SageMath and comparing function evaluation results with the integer sequence given by OEIS. We do this in order to detect errors in the heuristic parsing approach. At the moment 44% (47589 out of 107152) function validations return a perfect sequence match. Validated functions are exported by creating an expression graph using the abstract syntax trees library “AST” [1].

In the dataset, the automatic validation is shared as a MongoDB database, exported in the JSON format (`/computation/gf.json`). The file contains all generating functions, that could be heuristically harvested. An entry for a generating function contains the following name and value pairs:

- **\$oid**: a MongoDB unique ID
- **theory**: The sequence’s A-number
- **validated**: An internal flag that signifies that the function could be imported into SageMath (true for all entries in the current dataset)
- **type**: The type of generating function. Possible types are **an_succ** (direct generating functions), **gf_succ** and **egf_succ** (ordinary and exponential generating functions, respectively)
- **formula**: The original OEIS line without the prefix tag

- `parsed_formula`: The function without any additional “ASCII art”
- `data`: The sequence the function should produce, based on values given by OEIS
- `formula_data`: The sequence the function produces
- `match_ratio`: A float score, that is the result of matching the values of `data` and `formula_data`
- `offset`: It is not mandatory that a generating function starts creating the sequence at index 0 – `offset` is the result of trying to shift sequences to the left and right and trying to find a match.

Successfully parsed generating functions are furthermore exported to OpenMath [5] because the standard allows for storage, publication, web visualization and programmatical usage. Each function from the “gf.json” file is exported to a separate file. They are named “theory” _ “type” _ “\$oid”.

Organizational Content. The organizational harvester tries to enrich data about the sequence author and references by accessing the zbMATH author/document identification APIs using the python libraries *mechanize* [3] and *Beautiful Soup* [2]. If a query returns only one person or article, we link them to the respective item as a semantic identifier.

Organizational content is stored as RDF-triples where the predicates come from the FOAF, Dublin Core and ULO (upper library ontology [7]) ontologies for interoperability. An additional benefit of this format is that triple stores allow for SPARQL queries, and single parts of a triple are separately addressable via unique URIs.

The dataset contains references (`/organization/oeis_references.rdf`), OEIS authors (`/organization/oeis_authors.rdf`) and users (`/organization/oeis_users.rdf`). “Authors” are the result of trying to parse the OEIS author field. Note that the OEIS dataset contains impurities, where the heuristic harvesting approach encountered unexpected content or formatting. “Users”, in contrast, are extracted via web-crawling from the list of all OEIS users. *Author* and *user* triple stores contain the unique zbMATH identifier, if found. Only “users.rdf” entries contain the link to their wiki entry. A reference triple consists of the

- *creator*, consisting of one or more people,
- The *match score* expresses the certainty that the query result is the reference the zbMATH ID refers to.
- A *zbMATH ID* is a unique identifier for a document.
- The *reference* field is filled with the reference, as it was found in the OEIS:
- The *A-number* states which sequence contains this reference.
- And the *title* is the title of the document.

The file `/organization/oeis_a_num_references.csv` lists all mentions of a sequence in all other sequences. It contains three rows:

- *aNum*: the A-number of the sequence
- *atPosition*: the OEIS entry the sequence in *aNum* is mentioned in
- *appearsIn*: an integer declaring at which position the sequence is referenced

Narrative Content. includes all descriptive texts, comments, and keywords in the OEIS. The current partial harvest is saved as a XML-file. It holds the comments section and name of a sequence and authors and dates of program contributions to a sequence. Each entry includes a source reference as a source reference (“srcref”) attribute that contains the start position of the narrational element inside its OEIS entry.

3 Conclusion, Future Work, and Availability

We have presented a dataset of semanticised OEIS data. This dataset is comprehensive as it contains data from all five tetrapod aspects – whereas other libraries/datasets from mathematical software systems typically only span one or two: their native aspect and possibly narration.

This papers dataset (THODS) is based on an OEIS snapshot from December 2020. We are currently working with the OEIS community to obtain direct access to the OEIS data so that we can provide an automated periodic semantization process. The current and future versions of (THODS) are available under the DOI [10.5281/zenodo.6809687](https://doi.org/10.5281/zenodo.6809687).

The current dataset is limited by the fact, that it does not cover the full OEIS – we erred on the side of caution and only included the parts where semantization is verifiable. Ideally it would be a complete, tetrapodal representation of the OEIS, so that an OEIS entry reconstruction would be possible from its harvested tetrapodal components. We will work on further semantization, e.g. on (verified) semantizations of reported relations between formulae, e.g. from the “program” in the OEIS representation of A000055:

```
(Python)
# uses function from A000081
def A000055(n): return 1 if n == 0 else A000081(n)-sum(A000081(i)*A000081(n-i) for
    i in range(1, n//2+1)) + (0 if n % 2 else (A000081(n//2)+1)*A000081(n//2)//2) #
    Chai Wah Wu, Feb 03 2022
```

Fig. 2. OEIS program snippet from A000055

References

1. ast - abstract syntax trees, <https://docs.python.org/3/library/ast.html>
2. Beautiful soup: parsing html and xml documents, <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
3. mechanize: Stateful programmatic web browsing in python, <https://mechanize.readthedocs.io/en/latest/>
4. Oeis – generating functions (accessed 2022-05-24), https://oeis.org/wiki/Generating_functions

5. Openmath, <http://www.openmath.org>
6. Carette, J., Farmer, W.M., Kohlhase, M., Rabe, F.: Big math and the one-brain barrier – the tetrapod model of mathematical knowledge. *Mathematical Intelligence* **43**(1), 78–87 (2021). <https://doi.org/10.1007/s00283-020-10006-0>
7. Condoluci, A., Kohlhase, M., Müller, D., Rabe, F., Sacerdoti Coen, C., Wenzel, M.: Relational data across mathematical libraries. In: Kaliszyck, C., Brady, E., Kohlhase, A., Sacerdoti Coen, C. (eds.) *Intelligent Computer Mathematics (CICM)* 2019. pp. 61–76. No. 11617 in LNAI, Springer (2019). <https://doi.org/10.1007/978-3-030-23250-4>, <https://kwarc.info/kohlhase/papers/cicm19-ulo.pdf>
8. Corset, F., Fouladirad, M., Paroissin, C., Remy, E.: A parametric lifetime model for a multicomponents system with spatial interactions. *Applied Stochastic Models in Business and Industry* **35**(2), 221–233 (2019). <https://doi.org/https://doi.org/10.1002/asmb.2449>
9. Luzhnica, E., Kohlhase, M.: Formula semantification and automated relation finding in the OEIS. In: Greuel, G.M., Koch, T., Paule, P., Sommese, A. (eds.) *Mathematical Software - ICMS 2016 - 5th International Congress*. LNCS, vol. 9725. Springer (2016). <https://doi.org/10.1007/978-3-319-42432-3>, <https://kwarc.info/kohlhase/papers/icms16-oeis.pdf>
10. Sagemath, the Sage Mathematics Software System, <http://www.sagemath.org>, [Online; accessed 30 August 2016]
11. Vilz, S.: Oeis hobbyist example: A346180. (accessed 2022-05-09)., <https://oeis.org/A346180>
12. Wagner, M.: Tetrapodal Harvesting of the OEIS – FAIR, Semantic Extraction and Organization. Master’s thesis, Informatik, FAU Erlangen-Nürnberg (2021), https://gl.kwarc.info/supervision/MSc-archive/blob/master/2021/Wagner_Michael.pdf