Paper-1764955035930.pdf

Universität des Saarlandes

Dokumentdetails

Einreichungs-Nr.
trn:oid:::3741:334180211

Einreichungsdatum
5. Dez. 2025, 18:18 MEZ

Datum des Herunterladens
5. Dez. 2025, 18:22 MEZ

Dateiname
Paper-1764955035930.pdf

Dateigröße
3.3 MB

74 Seiten

36.247 Wörter

174.560 Zeichen

9% Ähnlichkeit insgesamt

Die Gesamtsumme aller Übereinstimmungen, einschließlich sich überschneidender Quell...

Übereinstimmungsgruppen

284 Nicht zitiert und wörtlich zitiert 8%
Übereinstimmungen, die weder im Text zitiert noch in Anführungszeichen gesetzt werden

22 Fehlende Zitate 0%
Übereinstimmungen, die weiterhin eine große Ähnlichkeit mit dem Quellmaterial aufweisen

13 Fehlende Quellenangaben 1%
Übereinstimmungen mit Anführungszeichen, aber ohne Verweise im Text

0 Zitiert und wörtlich zitiert 0%
Übereinstimmungen, bei denen im Text zitiert wird, aber keine Anführungszeichen vorhanden sind

Topquellen

6% Internetquellen
9% Publikationen
0% Eingereichte Arbeiten (Studentenarbeiten)

Integritätsmarkierungen

1 Integritätsmarkierung zur Überprüfung

Ersetzte Zeichen
103 verdächtige Zeichen auf 22 Seiten
Buchstaben werden gegen ähnliche Zeichen aus einem anderen Alphabet

Unsere Systemalgorithmen untersuchen Dokumente umfassend auf Inkonsistenzen, die sie von einer typischen Übermittlung unterscheiden. Wenn wir etwas als ungewöhnlich einstufen, weisen wir mit einer Markierung darauf hin, dass eine Überprüfung erforderlich ist.

Eine Markierung weist nicht unbedingt auf ein Problem hin. Wir empfehlen jedoch, dass Sie Ihre Aufmerksamkeit auf diese Bereiche richten.

Übereinstimmungsgruppen

284 Nicht zitiert und wörtlich zitiert 8%
Übereinstimmungen, die weder im Text zitiert noch in Anführungszeichen gesetzt werden

22 Fehlende Zitate 0%
Übereinstimmungen, die weiterhin eine große Ähnlichkeit mit dem Quellmaterial aufweisen

13 Fehlende Quellenangaben 1%
Übereinstimmungen mit Anführungszeichen, aber ohne Verweise im Text

0 Zitiert und wörtlich zitiert 0%
Übereinstimmungen, bei denen im Text zitiert wird, aber keine Anführungszeichen vorhanden sind

Topquellen

6% Internetquellen
9% Publikationen
0% Eingereichte Arbeiten (Studentenarbeiten)

Topquellen

Die Quellen mit der höchsten Anzahl von Übereinstimmungen innerhalb des Beitrags. Sich überschneidende Quellen w...

1	Internet	
drops.dagstuhl.de		3%
2	Publikation	
Bingling Dai, Yuhang Song, Yibin Jiang, Cheng Wang. "Data to Physics in Minutes: ...		<1%
3	Publikation	
"ECAI 2020", IOS Press, 2020		<1%
4	Internet	
theses.hal.science		<1%
5	Publikation	
Failing, David Michael. "Commutative, idempotent groupoids and the constraint ...		<1%
6	Internet	
filippoc.people.uic.edu		<1%
7	Publikation	
Pavel Pudlák. "Logical Foundations of Mathematics and Computational Complexi...		<1%
8	Publikation	
"Handbook of Satisfiability", IOS Press, 2021		<1%
9	Publikation	
"Relational and Algebraic Methods in Computer Science", Springer Science and B...		<1%
10	Publikation	
"Don Pigozzi on Abstract Algebraic Logic, Universal Algebra, and Computer Scien...		<1%

11	Publikation	Elliott Mendelson. "Introduction to Mathematical Logic", Chapman and Hall/CRC, ... <1%
12	Internet	leandrovendramin.org <1%
13	Publikation	"Automated Reasoning and Mathematics", Springer Nature, 2013 <1%
14	Publikation	Lecture Notes in Computer Science, 2005. <1%
15	Publikation	"Automated Deduction – CADE 30", Springer Science and Business Media LLC, 2025 <1%
16	Publikation	"Automated Deduction – CADE 27", Springer Science and Business Media LLC, 2019 <1%
17	Publikation	"Automated Deduction – CADE 28", Springer Science and Business Media LLC, 2021 <1%
18	Publikation	Kalle Kaarli, Alden F. Pixley. "Polynomial Completeness in Algebraic Systems", Ch... <1%
19	Publikation	Lecture Notes in Computer Science, 1984. <1%
20	Publikation	"Automated Reasoning", Springer Science and Business Media LLC, 2006 <1%
21	Publikation	Universitext, 2015. <1%
22	Internet	pdfcoffee.com <1%
23	Publikation	Araújo, João and Konieczny, Janusz. "A method for finding new sets of axioms for ... <1%
24	Publikation	"Foundations of Software Science and Computation Structures", Springer Science... <1%

25 Publikation
"Automated Reasoning", Springer Science and Business Media LLC, 2024 <1%

26 Publikation
Wieser, Eric Francis. "Formalizing Clifford Algebras and Related Constructions in t... <1%

27 Publikation
Endre Tóth. "Solution sets and centralizers", University of Szeged, 2025 <1%

28 Publikation
Lecture Notes in Computer Science, 2004. <1%

29 Publikation
Lorenz Halbeisen, Regula Krapf. "Gödel's Theorems and Zermelo's Axioms", Sprin... <1%

30 Publikation
Chow, Choiwah. "Finite Model Enumeration", Universidade Aberta (Portugal), 2024 <1%

31 Publikation
Patterson, Evan. "The Algebra and Machine Representation of Statistical Models."... <1%

32 Publikation
Lahrani, Hitakshi. "Classification of Finite Topological Quandles and Shelves via P... <1%

33 Publikation
Willsey, Max. "Practical and Flexible Equality Saturation.", University of Washingt... <1%

34 Publikation
Ian Stewart, David Tall. "Algebraic Number Theory and Fermat's Last Theorem", C... <1%

35 Publikation
"Logic for Programming, Artificial Intelligence, and Reasoning", Springer Science ... <1%

36 Publikation
Parsert, Julian. "Machine Learning for Function Synthesis.", University of Oxford (... <1%

37 Publikation
R. Sujatha, H. N. Ramaswamy, C. S. Yogananda. "Math Unlimited - Essays in Math... <1%

38 Publikation
"Automated Deduction – CADE-21", Springer Science and Business Media LLC, 2007 <1%

39 Publikation
Sushanta K. Mitra, Suman Chakraborty. "Microfluidics and Nanofluidics Handboo... <1%

40 Publikation
Paravicini, Walther Dietrich(Echterhoff, Siegfried). "KK-Theory for Banach Algebra... <1%

41 Übermittelte Arbeiten
saarlandes on 2025-01-30 <1%

42 Publikation
Bax, Joshua. "Disproving in First-Order Logic with Definitions, Arithmetic and Fini... <1%

43 Publikation
Erin C McKiernan. "Imagining the 'open' university: Sharing scholarship to improv... <1%

44 Publikation
Gwynne, Matthew. "Hierarchies for efficient clausal entailment checking: With ap... <1%

45 Publikation
Kotelnikov, Evgenii. "Automated Theorem Proving with Extensions of First-Order ... <1%

46 Publikation
Logistics Information Management, Volume 16, Issue 6 (2006-09-19) <1%

47 Publikation
Yestrepsky, Joseph M.. "How Employee Motive Attributions Shape the Relationshi... <1%

48 Publikation
Goertzel, Zarathustra Amadeus. "Learning Inference Guidance in Automated The... <1%

49 Publikation
Haym Benaroya, Seon Mi Han, Mark Nagurka. "Probabilistic Models for Dynamica... <1%

50 Publikation
Antao Chen, Edmond Murphy. "Broadband Optical Modulators - Science, Technol... <1%

51 Publikation
Duarte, André. "Efficient Reasoning in Equational Theories", The University of Ma... <1%

52 Publikation
Hendrix, Joseph D. "Decision procedures for equationally based reasoning", Proq... <1%

53 Publikation

Karen Ciaccio Secco. "A construção da identidade profissional do coach e a perce... <1%

54 Publikation

Stanley A Mulaik. "Foundations of Factor Analysis", Chapman and Hall/CRC, 2019 <1%

55 Publikation

Louise Grinstein, Sally I. Lipsey. "Encyclopedia of Mathematics Education", Routle... <1%

The Equational Theories Project: Advancing Collaborative Mathematical Research at Scale

Matthew Bolan, Joachim Breitner, Jose Brox, Nicholas Carlini, Mario Carneiro, Floris van Doorn, Martin Dvorak, Andrés Goens, Aaron Hill, Harald Husum, Hernán Ibarra Mejia, Zoltan Kocsis, Bruno Le Floch, Amir Livne Bar-on, Lorenzo Luccioli, Douglas McNeil, Alex Meiburg, Pietro Monticone, Pace P. Nielsen, Emmanuel Osalotioman Osazuwa, Giovanni Paolini, Marco Petracci, Bernhard Reinke, David Renshaw, Marcus Rossel, Cody Roux, Jérémy Scanvic, Shreyas Srinivas, Anand Rao Tadipatri, Terence Tao, Vlad Tsyrklevich, Fernando Vaquerizo-Villar, Daniel Weber, Fan Zheng

Abstract. We report on the *Equational Theories Project* (ETP), an online collaborative pilot project to explore new ways to collaborate in mathematics with machine assistance. The project successfully determined all 22 028 942 edges of the implication graph between the 4694 simplest equational laws on magmas, by a combination of human-generated and automated proofs, all validated by the formal proof assistant language *Lean*. As a result of this project, several new constructions of magmas satisfying specific laws were discovered, and several auxiliary questions were also addressed, such as the effect of restricting attention to finite magmas.

Contents

Date: December 5, 2025.

1. Introduction

The purpose of this paper is to report on the *Equational Theories Project* (ETP)[1], a pilot project launched[2] in September 2024 to explore new ways to collaboratively work on mathematical research projects using machine assistance. The project goal, in the area of universal algebra, was selected[3] to be particularly amenable to crowdsourced and computer-assisted techniques, while still being of mathematical research interest.

The project achieved its primary goal on 14 April 2025, when the $4694 \times (4694 - 1) = 22\,028\,942$ implications between the test set of 4694 equational laws were completely determined, with proofs or refutations formalized in *Lean*. This required coordinating the efforts of a large number of participants contributing both human-written formalizations and automatically generated proofs from various computer tools. In this paper, we report on both the scientific outcomes of the project, as well as the organizational issues that came up with organizing a mathematical project of this scale.

1.1. Magmas and equational laws.

In order to describe the mathematical goals of the ETP, we need some notation. A *magma* $\mathcal{M} = (M, \diamond)$ is a set M (known as the *carrier*) together with a binary operation $\diamond\colon M \times M \to M$. An *equational law* for a magma, or *law* for short, is an identity involving $\diamond$ and some formal indeterminates, which we will typically denote using the Roman letters $\mathrm{x}, \mathrm{y}, \mathrm{z}, \mathrm{w}, \mathrm{u}, \mathrm{v}$, as well as the formal equality symbol $\simeq$ in place of the equality symbol $=$ to emphasize the formal nature of the law. If M is finite, we refer to its cardinality as the *size* of the magma $\mathcal{M}$.

An *equational theory* is a collection of equational laws; in this paper we will primarily be concerned with theories generated by a single such law, although it is certainly of interest to explore larger theories as well. Equational theories are one of the simplest non-trivial examples of a theory in the model-theoretic sense; they also arise in various areas of computer science, such as term rewriting systems [8], automated theorem proving [43], and in the Dolev–Yao model [22] of interactive cryptographic protocols.

In the ETP, a unique number was assigned to each equational law, via a numbering system that we describe in Appendix A. For instance, the *commutative law* $\mathrm{x} \diamond \mathrm{y} \simeq \mathrm{y} \diamond \mathrm{x}$ is assigned to the equation number E43, while the *associative law* $\mathrm{x} \diamond (\mathrm{y} \diamond \mathrm{z}) \simeq (\mathrm{x} \diamond \mathrm{y}) \diamond \mathrm{z}$ is assigned to the equation E4512. A list of all equations referred to by number in this paper is also provided in Appendix A.

A magma $\mathcal{M} = (M, \diamond)$ satisfies a law E if the law E holds for all possible assignments of the indeterminates to elements of M, in which case we write $\mathcal{M} \models \mathrm{E}$. Thus, for instance $\mathcal{M} \models$ E43 if one has $x \diamond y = y \diamond x$ for all $x, y \in M$. Note that the formal indeterminate symbols x, y in E43 are now replaced by concrete elements x, y of the carrier M.

[1] https://teorth.github.io/equational_theories/

[2] https://terrytao.wordpress.com/2024/09/25

[3] The specific mathematical goal was inspired by the MathOverflow question "Is there an identity between the associative identity and the constant identity? ", posed on July 17, 2023.

We say that a law E *entails* or *implies* another law E′ if every magma that satisfies E, also satisfies E′: $(\mathcal{M} \models \mathrm{E}) \implies (\mathcal{M} \models \mathrm{E}')$. We write this relation as $\mathrm{E} \models \mathrm{E}'$. We say that two laws are *equivalent* if they entail each other. For instance, the constant law $\mathrm{x} \diamond \mathrm{y} \simeq \mathrm{z} \diamond \mathrm{w}$ (E46) can easily be seen to be equivalent to the law $\mathrm{x} \diamond \mathrm{x} \simeq \mathrm{y} \diamond \mathrm{z}$ (E41). It is clear that $\models$ is a pre-order, that is to say a partial order after one quotients by equivalence.

In this entailment pre-ordering, the maximal element is given by the trivial law $\mathrm{x} \simeq \mathrm{x}$ (E1), and the minimal element is given by the singleton law $\mathrm{x} \simeq \mathrm{y}$ (E2), thus $\mathrm{E2} \models \mathrm{E} \models \mathrm{E1}$ for all laws E.

We also define a variant: we say that E *entails* E′ *for finite magmas*, and write $\mathrm{E} \models_{\mathrm{fin}} \mathrm{E}'$, if every *finite* magma that satisfies E, also satisfies E′. Clearly, the relation $\mathrm{E} \models \mathrm{E}'$ implies $\mathrm{E} \models_{\mathrm{fin}} \mathrm{E}'$; but, as observed by Austin [7], the converse is not true in general.

The *order* of an equational law is the number of occurrences of the magma operation, and can be viewed as a crude measure of complexity of the law. For instance, the commutative law E43 has order 2, while the associative law E4512 has order 4. We note some selected laws of small order that have previously appeared in the literature:

- The *central groupoid law* $\mathrm{x} \simeq (\mathrm{y} \diamond \mathrm{x}) \diamond (\mathrm{x} \diamond \mathrm{z})$ (E168) is an order-3 law introduced by Evans [25] and studied further by Knuth [34] and many further authors, being closely related to central digraphs (also known as unique path property digraphs), and leading in particular to the discovery of the Knuth-Bendix algorithm [35]; see [39] for a more recent survey.
- *Tarski's axiom* $\mathrm{x} \simeq \mathrm{y} \diamond (\mathrm{z} \diamond (\mathrm{x} \diamond (\mathrm{y} \diamond \mathrm{z})))$ (E543) is an order-4 law that was shown by Tarski [63] to characterize the operation of subtraction in an abelian group; that is to say, a magma $\mathcal{M} = (M, \diamond)$ satisfies E543 if and only if there is an abelian group structure on $\mathcal{M}$ for which $x \diamond y = x - y$ for all $x, y \in M$.
- In a similar vein, it was shown in [47] (see also [48]) that the order-4 law $\mathrm{x} \simeq (\mathrm{y} \diamond \mathrm{z}) \diamond (\mathrm{y} \diamond (\mathrm{x} \diamond \mathrm{z}))$ (E1571) characterizes addition (or subtraction) in an abelian group of exponent 2; it was shown in [44] that the order-6 law $\mathrm{x} \simeq (\mathrm{y} \diamond ((\mathrm{x} \diamond \mathrm{y}) \diamond \mathrm{y})) \diamond (\mathrm{x} \diamond (\mathrm{z} \diamond \mathrm{y}))$ (E345169) characterizes the Sheffer stroke in a boolean algebra, and it was shown in [28] that the order-8 law $\mathrm{x} \simeq \mathrm{y} \diamond ((((\mathrm{y} \diamond \mathrm{y}) \diamond \mathrm{x}) \diamond \mathrm{z}) \diamond (((\mathrm{y} \diamond \mathrm{y}) \diamond \mathrm{y}) \diamond \mathrm{z}))$ (E42323216) characterizes division in a (not necessarily abelian) group.

Some further examples of laws characterizing well-known algebraic structures are listed in [43].

The Birkhoff completeness theorem [8, Th. 3.5.14] implies that an implication $\mathrm{E} \models \mathrm{E}'$ of equational laws holds if and only if the left-hand side of E′ can be transformed into the right-hand side by a finite number of substitution rewrites using the law E. However, the problem of determining whether such an implication holds is undecidable in general [46]. Even when the order is small, some implications[4] can require lengthy computer-assisted proofs; for instance, it was noted in [32] that the order-4 law $\mathrm{x} \simeq (\mathrm{y} \diamond \mathrm{x}) \diamond ((\mathrm{x} \diamond \mathrm{z}) \diamond \mathrm{z})$ (E1689) was equivalent to the singleton law $\mathrm{x} \simeq \mathrm{y}$ (E2), but all known proofs were found

[4] Another contemporaneous example of this phenomenon was the solution of the Robbins problem [42].

with computer assistance.[5] Furthermore, for the finite magma implication relation $E \models_{\text{fin}} E'$, no analogue of the Birkhoff completeness theorem is available.

1.2. **The Equational Theories Project.** As noted in Appendix A, there are 4694 equational laws of order at most 4. The primary mathematical goal of the ETP was to completely determine the *implication graph* for these laws, in which there is a directed edge from E to E′ precisely when $E \models E'$. As the project progressed, an additional goal was added to determine the slightly larger *finite implication graph*, in which there is a directed edge from E to E′ precisely when $E \models_{\text{fin}} E'$.

Such systematic determinations of implication graphs have been seen previously in the literature; for instance, in [56], the relations between 60 identities of Bol–Moufang type were established, and in the blog post [68, §17], some initial steps towards generating this graph for the first hundred or so laws on our list were performed. However, to our knowledge, the ETP is the first project to study such implications at the scale of thousands of laws.

The ETP requires the determination of the truth or falsity of $4694^2 = 22\,033\,636$ implications (for both arbitrary magmas and finite magmas), or $4694 \times (4694 - 1) = 22\,028\,942$ if the reflexive implications $E \models E$ are removed; while one can use properties such as the transitivity of entailment to reduce the work somewhat, this is clearly a task that requires significant automation. It was also a project highly amenable to crowdsourcing, in which different participants could work on developing different techniques, each of which could be used to fill out a different part of the implication graph. In this respect, the project could be compared with a Polymath project [27], which used online forums such as blogs and wikis to openly collaborate on a mathematical research problem. However, the Polymath model required human moderators to review and integrate the contributions of the participants, which clearly would not scale to the ETP which required the verification of over twenty million mathematical statements. Instead, the ETP was centered around a GitHub repository in which the formal mathematical contributions had to be entered in the proof assistant language *Lean*, where they could be automatically verified. In this respect, the ETP was more similar to the recently concluded Busy Beaver Challenge[6], which was a similarly crowdsourced project that computed the fifth Busy Beaver number $BB(5)$ to be 47 176 870 through an analysis of about 180 million Turing machines, with the halting analysis being verified in a variety of computer languages, with the final formal proof written in the proof assistant language *Coq* [65, 66]. One of the aims of the ETP was to explore potential workflows for such collaborative, formally verified mathematical research projects that could serve as a model for future projects of this nature.

Secondary aims of the ETP included the possibility of discovering unusually interesting equational laws, or new experimental observations about such laws, that had not previously been noticed in the literature; and to develop benchmarks to assess the performance of automated theorem provers and other AI tools.

[5] We improved such a proof to make it human-readable, see the blueprint of the ETP.

[6] `https://bbchallenge.org/`

FIGURE 1. A Hasse diagram of all the equational laws implied by E854 (for unrestricted magmas). An edge in this diagram indicates that the lower equation implies the higher one. Rounded rectangles indicate groups of equivalent laws. This graph was produced by the visualization tool *Graphiti*, which was developed for this project.

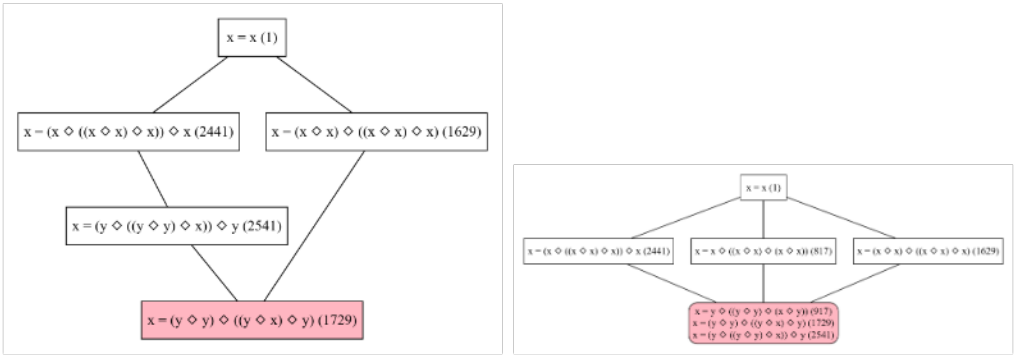


FIGURE 2. A Hasse diagram of all the equational laws implied by E1729, both for unrestricted magmas (left) and finite magmas (right). Note the slightly larger number of implications in the latter.

1.3. **Outcomes.** The ETP achieved almost all of its primary objectives, with all of the 22 033 636 implications $E \models E'$ and non-implications $E \not\models E'$ formalized in the proof assistant language *Lean*, and can be found on the ETP GitHub repository. See Figure 1, Figure 2 and Figure 3 for some small fragments of the implication graphs produced. The 4694 laws organized into 1415 equivalence classes, with by far the largest class being the class of 1496 equations equivalent to the singleton law E2.

For the finite implication graph $E \models_{\text{fin}} E'$, we could similarly formalize all but two implications. Specifically, we were unable to obtain either a human-readable or formalized proof or disproof of the implication E677 $\models_{\text{fin}}$ E255 (or its equivalent dual E2910 $\models_{\text{fin}}$ E47), despite extensive efforts from the participants of the project; we tentatively conjecture this implication to be false (i.e., that there exists a finite magma satisfying E677 but not E255), but the refutation appears to be "immune" to most of the techniques that we developed for the project. (We were however able to establish that the corresponding implication E677 $\models$ E255 for arbitrary magmas was false, using the greedy construction discussed in Section 5.5.)

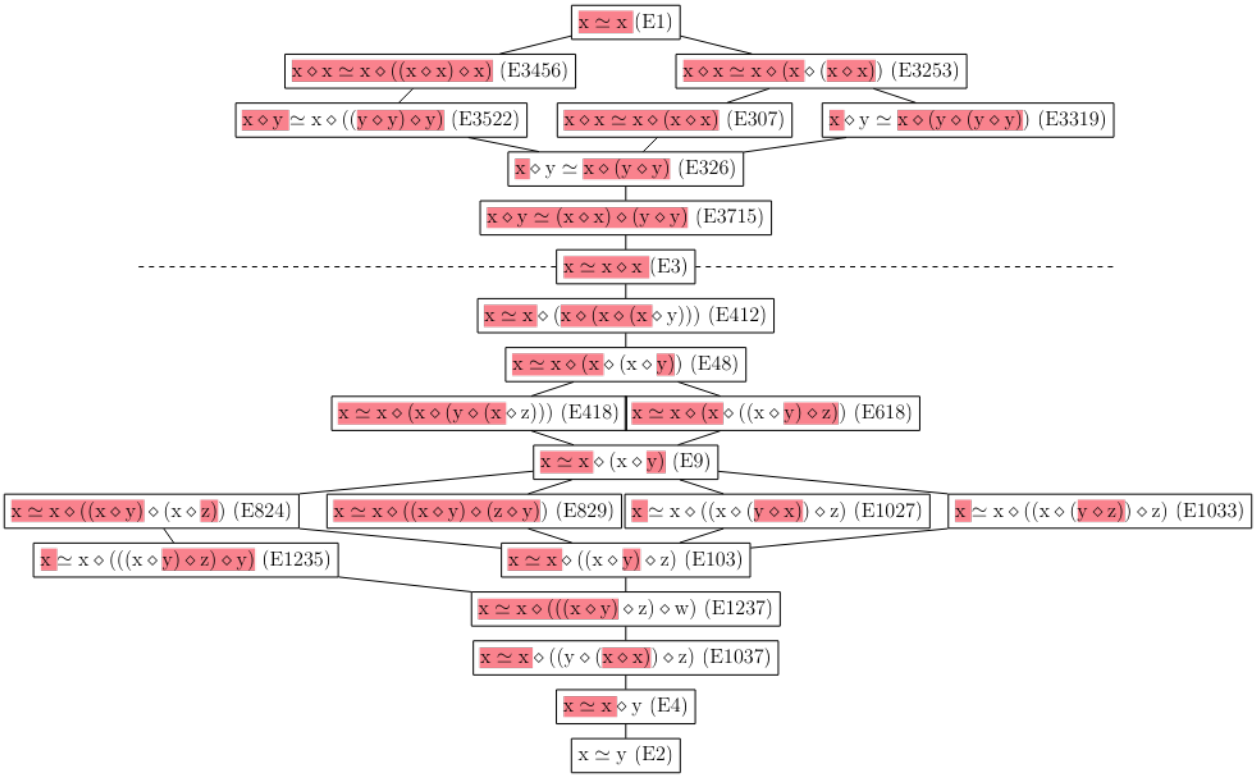


FIGURE 3. Longest chains of implications (length 15) between inequivalent laws in the implication graph. The parts above/below law E3 can be independently dualized.

Of the 22 033 636 possible implications $\mathrm{E} \models \mathrm{E}'$, 8 178 279 (or 37.12%) would end up being true; for an additional set of either 820 or 822 pairs E, E', the weaker implication $\mathrm{E} \models_{\mathrm{fin}} \mathrm{E}'$ also held. To establish such positive implications $\mathrm{E} \models \mathrm{E}'$ or $\mathrm{E} \models_{\mathrm{fin}} \mathrm{E}'$, the main techniques used were as follows:

- A very small number of positive implications were established and **formalized by hand**, mostly through direct rewriting of the laws; but this approach would not scale to the full project.
- **Simple rewriting rules**, for instance based on the observation that any law of the form $\mathrm{x} \simeq f(\mathrm{y}, \mathrm{z}, \dots)$ was necessarily equivalent to the singleton law E2, could already reduce the size of potential equivalence classes by a significant fraction. We discuss this method in Section 6.1.
- The preorder axioms for $\models$, as well as the "duality" symmetry of the preorder with respect to replacing a magma operation $x \diamond y$ with its opposite $x \diamond^{\mathrm{op}} y := y \diamond x$, can be used to significantly cut down on the number of implications that need to be proven explicitly; ultimately, only 10 657 (0.13%) of the positive implications needed a direct proof.
- To obtain additional implications for finite magmas, heavy reliance was made on the fact that for functions $f\colon M \to M$ on a finite set M, surjectivity was equivalent to injectivity. Some more sophisticated variants of this idea can lead to additional implications; see Section 5.1.

- **Automated Theorem Provers** (ATP) could be deployed at extremely fast speeds to establish a complete generating set of positive implications; see Section 7.

More challenging were the 13 855 357 (62.88%) implications that were false, $\mathrm{E} \not\models \mathrm{E}'$, and particularly the slightly smaller set of 13 854 535 or 13 854 537 implications that were false even for finite magmas, $\mathrm{E} \not\models_{\mathrm{fin}} \mathrm{E}'$. Here, the range of techniques needed to refute such implications were quite varied, and may be of independent interest:

- **Small finite magmas**, which can be described explicitly by multiplication tables, could be tested by brute force computations to provide a large number of finite counterexamples to implications, or by ATP-assisted methods. See Section 5.1.
- **Linear models**, in which the magma operation took the form $x \diamond y = ax + by$ for some (commuting or noncommuting) coefficients a, b, allowed for another large class of counterexamples to implications, which could be automatically scanned for, either by brute force or by Gröbner basis type calculations; many of these examples could also be made finite. See Section 5.2.
- **Translation invariant models**, in which the magma operation took the form $x \diamond y = x + f(y - x)$ on an additive group, or $x \diamond y = x f(x^{-1} y)$ on a noncommutative group, reduce matters to analyzing certain functional equations; see Section 5.3.
- To each equation E one can associate a "**twisting semigroup**" S_{E}. If S_{E} is larger than $S_{\mathrm{E}'}$, then this can often be used to disprove the implication $\mathrm{E} \models \mathrm{E}'$; see Section 5.4.
- **Greedy methods**, in which either the multiplication table $(x, y) \mapsto x \diamond y$ or the function f determining a translation-invariant model are iteratively constructed by a greedy algorithm subject to a well-chosen ruleset, were effective in resolving many implications not easily disposed of by preceding methods. See Section 5.5.
- Starting with a simple base magma $\mathcal{M}$ satisfying both E and E′, and either **enlarging** it to a larger magma $\mathcal{M}'$ containing $\mathcal{M}$ as a submagma, **extending** it to a magma $\mathcal{N}$ with a projection homomorphism $\pi : \mathcal{N} \to \mathcal{M}$, or *modifying the multiplication table* on a small number of values, also proved effective when combined with greedy methods or with a "**magma cohomology**" construction. See Section 5.6.
- **Syntactic methods**, such as observing a "matching invariant" of the law E that was not shared by the law E′, could be used to obtain some refutations. For instance, if both sides of E had the same order, but both sides of E′ did not, this could be used to syntactically refute $\mathrm{E} \models \mathrm{E}'$. Similarly, if the law E was confluent, enjoyed a complete rewriting system, or otherwise permitted some understanding of the free magma associated to that law, one could decide the assertions $\mathrm{E} \models \mathrm{E}'$ for all possible laws E′, or at least a significant fraction of such laws. We discuss these methods, and the extent to which they can be automated, in Section 6.
- Some **ad hoc models** based on existing mathematical objects, such as infinite trees, rings of polynomials, or "Kisielewicz models" utilizing the prime factorization of the natural numbers, could also handle some otherwise difficult cases. In some cases, the magma law induced some relevant and familiar structures, such as a directed graph or a partial order, which also helped guide counterexample constructions. We will not detail these diverse examples here, but refer the reader to the ETP blueprint for more discussion.

- **Automated theorem provers** were helpful in identifying which simplifying axioms could be added to the magma without jeopardizing the ability to refute the desired implication $\mathrm{E} \models \mathrm{E}'$ or $\mathrm{E} \models_{\mathrm{fin}} \mathrm{E}'$.

While the vast majority of negative implications could be quickly resolved by one of the above techniques, either with human input or in a completely automated fashion, there were perhaps two dozen such negative results that required quite delicate and *sui generis* constructions. The hardest such implication, E1729 $\not\models$ E817, took several months to establish and then formalize (using a combination of many of the above constructions), with the final proof in *Lean* requiring just over 4000 dedicated lines of code from multiple contributors.

In the course of completing the implication graph, some interesting new algebraic structures were discovered. One such example concerns the magmas satisfying E1485, which we refer to as *weak central groupoids* as they contain the central groupoids (satisfying E168) as a subclass. In [34] it was observed that all finite central groupoids have order equal to a perfect square n^2; empirically, we have found that finite weak central groupoids always have order n^2 or $2n^2$, although we have no rigorous proof of this claim; they also have a graph-theoretic interpretation analogous to the interpretation of central groupoids as digraphs with the unique path property. For these and other observations we refer the reader to the blueprint of the ETP.

The objective of using the data from the ETP to establish well-calibrated benchmarks to evaluate ATPs remains an interesting open problem; the participants of this project did not have the required expertise to develop and test such benchmarks to the standards expected in the area. However, in Section 7 we present a more informal "field report" of our experiences using ATPs in the project, in the hope that this will provide some useful guidance to other researchers seeking to incorporate ATPs into their own research.

1.4. **Further directions.** While the primary objective of the ETP was being completed, some additional related results were generated as spinoffs. Specifically:

- In the blueprint on the ETP web site, we report some partial progress on classifying which of the 57 882 distinct laws of order 5 are equivalent to the singleton law E2, either with or without the requirement that the magma be finite.
- In Section 9 we report on the determination of laws with full spectrum, i.e., with magmas satisfying them of all finite sizes.
- In Section 10 we report on classifying the laws of order 8 that are equivalent to the Higman-Neumann law E42323216.

2. Notation and mathematical foundations

If $\mathcal{M} = (M, \diamond)$ is a magma, we define the left and right multiplication operators $L_a, R_a \colon M \to M$ for $a \in M$ by the formula

$$L_x y = R_y x := x \diamond y. \tag{1}$$

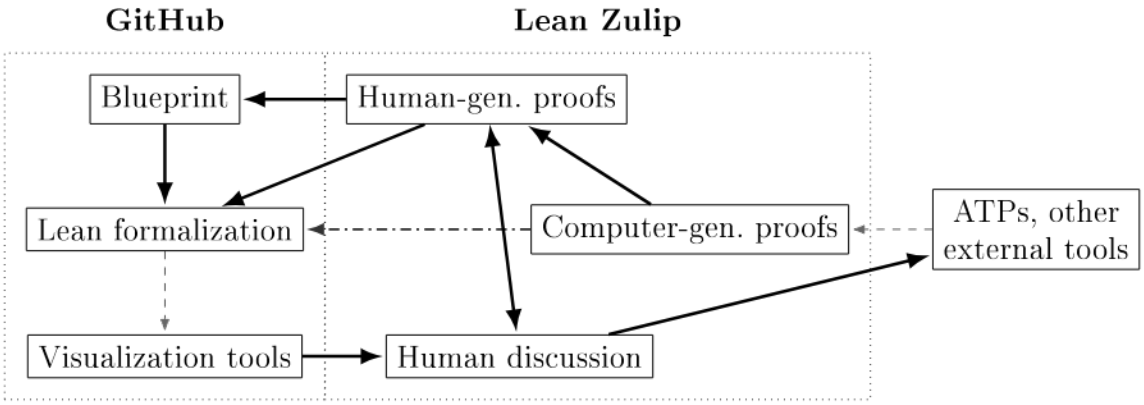


FIGURE 4. Some of the main dynamics in which proofs were generated, discussed within the Lean Zulip channel and then formalized in the Github repository. Boldface arrows indicate human activities, such as proposing an automated attack on outstanding implications, converting a computer-generated proof into a human-readable format, formalizing a human readable proof directly, or first creating a more precise blueprint for other collaborators to work on. Dashed arrows indicate fully automated processes, while the partly dashed line indicated a semi-automated process requiring human supervision.

We also define the squaring operator $S\colon M \to M$ by

$$Sx := x \diamond x = L_x x = R_x x. \tag{2}$$

A *homomorphism* $f\colon \mathcal{M} \to \mathcal{M}'$ between two magmas $\mathcal{M} = (M, \diamond)$, $\mathcal{M}' = (M', \diamond')$ is a function $f\colon M \to M'$ such that $f(x \diamond y) = f(x) \diamond' f(y)$ for all $x, y \in M$. An *isomorphism* is a homomorphism that is invertible (which implies that the inverse is also a homomorphism). An *endomorphism* is a homomorphism from a magma to itself.

If X is an alphabet, we let $\mathcal{M}_X = (M_X, \diamond)$ denote the free magma generated by X, thus an element of $\mathcal{M}_X$ is either a letter in X, or of the form[7] $w_1 \diamond w_2$ with $w_1, w_2 \in M_X$. Every function $f\colon X \to M$ into a magma $\mathcal{M} = (M, \diamond)$ extends to a unique homomorphism $\varphi_f\colon \mathcal{M}_X \to \mathcal{M}$. Formally, an equational law with some indeterminates in X can be written as $w_1 \simeq w_2$ for some $w_1, w_2 \in M_X$; a magma $\mathcal{M} = (M, \diamond)$ then satisfies this law if and only if $\varphi_f(w_1) = \varphi_f(w_2)$ for all $f\colon X \to M$. We also define the order of a word $w \in M_X$ to be the number of occurrences of $\diamond$ in the word, thus letters in X are of order 0, and the order of $w_1 \diamond w_2$ is the sum of the orders of w_1, w_2, plus one.

A *theory* is a collection Γ of equational laws; we say that a magma $\mathcal{M}$ *satisfies* a theory, and write $\mathcal{M} \models \Gamma$, if every law in Γ is satisfied by $\mathcal{M}$. If E is an equational law, we write $\Gamma \models \mathrm{E}$ if every magma that satisfies Γ also satisfies E. A *free magma* $\mathcal{M}_{X,\Gamma} = (M_{X,\Gamma}, \diamond)$ for such a theory Γ and an alphabet X is a magma satisfying Γ together with a map $\iota_{X,\Gamma}\colon X \to M_{X,\Gamma}$ which is universal in the sense that every function $f\colon X \to \mathcal{M}$ to a magma $\mathcal{M}$ satisfying Γ uniquely determines a homomorphism $\varphi_{f,\Gamma}\colon \mathcal{M}_{X,\Gamma} \to \mathcal{M}$ such that $\varphi_{f,\Gamma} \circ \iota_{X,\Gamma} = f$. This

[7]Strictly speaking, one should use parentheses and write $(w_1 \diamond w_2)$ to avoid ambiguity, but to reduce clutter we shall abuse notation by omitting parentheses when no ambiguity is caused by doing so.

magma is unique up to isomorphism; a canonical way to construct it is as the quotient $\mathcal{M}_X/\sim_\Gamma$ of the free magma $\mathcal{M}_X$ by the equivalence relation $\sim_\Gamma$ given by declaring $w \sim_\Gamma w'$ if $\Gamma \models w \simeq w'$ [8, Theorem 3.5.6]. If $\Gamma = \{\mathrm{E}\}$ consists of a single law E, we write $\mathcal{M}_{X,\mathrm{E}}$, $\sim_\mathrm{E}$, $\varphi_{f,\mathrm{E}}$ for $\mathcal{M}_{X,\{\mathrm{E}\}}$, $\sim_{\{\mathrm{E}\}}$, $\varphi_{f,\{\mathrm{E}\}}$ respectively.

In general, the free magma $\mathcal{M}_{X,\Gamma}$ is difficult to describe in a tractable form, but for some theories, one has a simple description. We give two simple examples here:

Example 2.1 (Commutative and associative free magma)**.** The free magma $\mathcal{M}_{X,\{\mathrm{E43,E4512}\}}$ for the commutative law E43 and the associative law E4512 is the free abelian semigroup generated by X (with $\iota_{X,\{\mathrm{E43,E4512}\}}$ the obvious embedding map).

Example 2.2 (Left-absorptive free magma)**.** The free magma $\mathcal{M}_{X,\{\mathrm{E4}\}}$ for the left-absorptive law E4 is the magma with carrier X and operation $x \diamond y = x$ (with $\iota_{X,\mathrm{E4}}$ the identity).

Every magma $\mathcal{M}$ has an opposite $\mathcal{M}^{\mathrm{op}}$, which has the same carrier but the opposite operation $x \diamond^{\mathrm{op}} y := y \diamond x$. A magma $\mathcal{M}$ satisfies an equational law E if and only if its opposite $\mathcal{M}^{\mathrm{op}}$ satisfies the dual law E^*, defined by reversing all the operations. For instance, the dual of $\mathrm{x} \diamond \mathrm{y} \simeq \mathrm{x} \diamond (\mathrm{y} \diamond \mathrm{z})$ (E327) is $\mathrm{y} \diamond \mathrm{x} \simeq (\mathrm{z} \diamond \mathrm{y}) \diamond \mathrm{x}$, which in our numbering system we rewrite in normal form as $\mathrm{x} \diamond \mathrm{y} \simeq (\mathrm{z} \diamond \mathrm{x}) \diamond \mathrm{y}$ (E395).

We then see that the implication graph has a duality symmetry: given two equational laws $\mathrm{E}_1, \mathrm{E}_2$, we have $\mathrm{E}_1 \models \mathrm{E}_2$ if and only if $\mathrm{E}_1^* \models \mathrm{E}_2^*$.

3. Formal foundations

All proofs in the ETP were ultimately formalized in the proof assistant language *Lean*, though in many cases the proofs were first written in an informal human document, which was then incorporated into the human-readable *blueprint*[8] that accompanied the formalization. Many of the computer-assisted proofs were also first generated as computer output from a source other than *Lean*, such as an ATP, and later converted to a *Lean* proof by a separate program custom-written for this task.

The project relied on *Lean*'s extensive *Mathlib* library, for instance to provide support for algebraic concepts such as the free group that arose in some of the more difficult constructions. Additional extensions to *Lean*, such as *duper* or *egg*, were employed by some participants in external forks of the repository, but we did not incorporate them into the master repository to simplify the version control process. As a consequence, some manual translation of proofs produced using such extensions to a proof that avoided such extensions were needed at various stages of the project.

The concept of a magma could be modeled by existing *Mathlib* classes such as `Mul`; however we chose early in the project to define a custom magma class `Magma` instead, as for some magma constructions the magma operation (which we denoted $\diamond$) was distinct from an existing multiplication structure $*$ on the same carrier. Most components of the *Lean* codebase were placed in namespaces to avoid collisions with each other, and with *Mathlib*.

[8] https://github.com/PatrickMassot/leanblueprint

```
@[equational_result]
theorem _root_.Equation1437_not_implies_Equation4269 :
∃ (G : Type) (_ : Magma G), Equation1437 G ∧ ¬ Equation4269 G := by
    use ℕ × Fin 3, ⟨op⟩
    constructor
    · intro x y z
      simp [op, add_assoc]
    · simp only [not_forall, op]
      use (0, 0), (2, 0)
      decide
```

FIGURE 5. A sample proof of a formalized implication, in this case that E1437 $\not\models$ E4269.

```
@[equational_result]
theorem «Facts from All4x4Tables [[1,2,3,4,5,0],[4,1,2,5,0,3],[3,0,5,2,1,4],
[0,5,4,3,2,1],[5,4,1,0,3,2],[2,3,0,1,4,5]]» :
  ∃ (G : Type) (_ : Magma G) (_: Finite G), Facts G [1316, 2863] [411, 680,
  817, 1020, 1426, 2035, 2441, 2644, 2853, 2855, 2865, 2872, 2947, 3050,
  3253, 3456, 4270, 4283, 4290, 4380, 4598, 4605, 4656] :=
⟨Fin 6, «All4x4Tables [[1,2,3,4,5,0],[4,1,2,5,0,3],[3,0,5,2,1,4],[0,5,4,3,2,1],
[5,4,1,0,3,2],[2,3,0,1,4,5]]», Finite.of_fintype _, by decideFin!⟩
```

FIGURE 6. A computer generated `Facts` theorem, using an explicit finite magma of order 6 to refute several implications at once.

Equational laws in the project were implemented both syntactically - as a structure `LawX` containing two words in a free group - as well as semantically, as a predicate `EquationX` that could be applied to a magma. Here `X` is the number assigned to the law. The semantic formulation (`EquationX`) was more convenient for proving or refuting specific implications, while the syntactic formulation (`LawX`) was preferred for implementing metatheorems, such as the use of duality between laws. *Lean*'s metaprogramming features proved to be vital to relate the two representations. A custom command, `equation`, was created for specifying equational laws. Elaborating the `equation` command generated both `EquationX` and `LawX` definitions from this description, as well as theorems relating them to each other. A similar construction was used to generate dual laws, where the dual law was given explicitly for simplicity.

To facilitate the automatic generation of an implication graph from the *Lean* codebase, a custom `@[equational_result]` tag was formed to attach to propositions in *Lean* to indicate that they were proving or refuting one or more implications; see Figure 5. A `conjecture` keyword was also created for implications or refutations which we wished to identify as having an informal proof that had yet to be formalized in *Lean*.

A single construction of a magma could satisfy multiple laws $E_1, E_2, \ldots$ and not satisfy others $E'_1, E'_2, \ldots$, leading to a large number of refutations of the form $E_i \not\models E'_j$. A custom `Facts` command was designed to organize such information efficiently; see Figure 6.

As an additional precaution against "exploit"-based proofs (such as those that might be contributed by an AI tool) *lean4checker* was used to ensure that no axioms were used in *Lean* outside of a small trusted set. In particular, *Lean* tactics such as `native_decide` that relied on external tools were not permitted into the codebase.

Explicitly formalizing all 22 028 942 implications as theorems would lead to an infeasible compilation time in *Lean*. Instead, a reduced generating set of 10 657 positive implications and 586 925 negative implications were formalized, with the latter in turn mostly organized into a smaller number of `Facts` theorems as discussed above. The extension of these results to the rest of the implication graph via transitivity and duality is currently done by programs external to *Lean*, although in principle one could create an "end-to-end theorem" which completely establishes the implication graph within *Lean*.

Some lemmas generated in the project were suitable for upstreaming back to *Mathlib*, as well as several technical improvements to the *LeanBlueprint* software.

4. Project management

This project is, among other things, an experiment on how to organise large scale collaborations for mathematical work. In this section, we describe several aspects of the organisation of the collaborative effort.

4.1. **Problems of scale in mathematical collaboration.** In order to understand the scaling issues that can arise in large scale collaborations, it helps to revisit the mechanics of traditional mathematical collaborations and their limitations. While every collaboration is unique, there are some general patterns. A small number of contributors, usually under ten, who may know each other, join forces to tackle some class of problems. Typically the collaborators are almost all academics who share substantial amounts of common knowledge. They discuss the problem at hand together, typically with some shared written medium such as a whiteboard. After several rounds of discussion and refinement, different members of the collaboration come up with different pieces of a solution. These pieces are then put together via discussion and merging of write-ups over several iterations. Once the collaborators are reasonably confident about the correctness of their work, including theorem statements and proofs, they submit the paper for peer review. Thus the correctness of mathematical research relies on this basic cycle of discuss, solve, write, cross-check, and revise, followed by peer review. Ultimately the authors take responsibility for the contents of their research article. This joint responsibility for authorship is formally enshrined by mathematical societies. For instance, see point 4 of the EMS code of practice for joint responsibility [61].

However this project involved over fifty contributors spread across the world with diverse academic and professional backgrounds. They collaborated across several timezones and countries over the internet. The aforementioned process does not scale. Collaborators do not usually know each other nearly as well as they would in a traditional project. Thus such a collaboration does not have the same level of mutual trust. Further, as the number of contributors grows beyond the single digits, it becomes increasingly difficult to ensure the robustness of each other's results, because of the sheer volume of material produced. Even

delegating responsibility for the various pieces of mathematical work and integrating them into a coherent whole becomes difficult. Concretely, the scaling challenge manifests in several ways:

- Partitioning and allocating tasks to voluntary contributors, keeping track of progress on the respective subtasks, and ensuring that everybody gets a fair chance at contributing without conflicting submissions for the same subproblems.
- Homogenising the mathematical content generated across multiple discussions spanning various forums into a coherent piece of work.
- Tracking progress relative to the goals of the project.
- Verifying the correctness of contributions made by more than fifty people with diverse backgrounds who might not share a common mathematical vocabulary, and collaborate across multiple timezones, using a diverse set of tools.

Of the challenges mentioned above, this section deals with the first, second, and last. We briefly address the third challenge of tracking progress, the tools for which are described in Section 12. We spend a lot of time on the last point of trust and verification of results for two reasons. On the one hand, use of tools like *Lean* is fairly new in mathematical research, and while the community researching theorem provers is familiar with their guarantees and limitations, a clear academic exposition targeted at mathematics researchers will be a helpful resource for future reference, to fill a gap that is currently covered by online forums and folklore. We also describe the important role played by a number of other tools in the project.

4.2. **The Blueprint tool.** The formalization of proofs is an act of careful engineering. It is therefore helpful to have a blueprint with detailed natural language lemmata, definitions, and proof sketches in *Lean*. In the *Lean* community it has been conventional to use the *Lean blueprint* tool by Patrick Massot et al. [1]. The typical formalization project has a clearly defined set of target theorems, and the authors of the project work with a known proof, to produce a clear roadmap for the formalization. The *Lean blueprint* tool is capable of linking each piece of this natural language document to its *Lean* encoding, tracking the dependency of definitions and theorems, and progress through them, by producing a key coloured dependency graph. Thus the managers of the formalization project can not only organise the project to distribute tasks among contributors, but also track when various pieces of the formalization are complete.

In this project, we were entering uncharted mathematical territory. We had a clear list of tasks to accomplish, namely to prove the implication or anti-implication between every pair of equational laws, up to transitivity and duality. At the same time there was no clearly known pen and paper proof available for any of these beforehand. This meant that we could not prepare the blueprint of the project in advance and organise the formalization around it. Thus the traditional roles played by the blueprint were replaced by a number of other tools and mechanisms. In particular, the dependency graph did not play its traditional role in formalization projects. We developed a number of visual tools to track our progress in the project in terms of remaining open implications and anti-implications (see Section 12). Within *Lean*, every equational result was tagged with the `@[equational_result]` attribute

to identify the theorem as one of the project goals, and this attribute was used to collect the status of all the goal theorems of the project. Instead of covering the dependency graph node by node, progress in the project happened as various contributors uncovered some structural ideas or heuristics that helped ATPs solve one or more pairs of laws.

The blueprint tool played a very important role in recording our progress and formalizing these classes of implications or anti-implications. It is the only comprehensive record of all the techniques that were employed in the project. Further at the level of specific implications and anti-implications, the blueprint and formalization evolved as in other projects, hand in hand. As an example, the formalization of the anti-implication E1729 $\not\models$ E817 proceeded through several iterations of refinement of the blueprint and formalization.

In conclusion, when using ITPs for tackling open problems, especially at scale, we observed that the role of the blueprint changed, but it still remained an important way to track and document our progress at a local level across the project.

4.3. **The project template.** When working on a formalization project, there are many moving pieces that need to work in concert. At the core level, there is the project set up by *Lean*'s build and dependency management system *lake*. But in addition to that, there are several pieces, including the aforementioned blueprint tool, as well as scripts that a user may choose to run to visualise various aspects of the project, or check the project in specific ways, or compile documentation automatically as the project advances. These additional tasks are accomplished by a number of external tools, and combining them in a mutually compatible way can be challenging. We side-stepped most of these issues by using the GitHub template repository of Pietro Monticone [49]. At the same time, when we began the project, the template in place was suited for more conventional formalization projects and the tooling they required. It also did not include the scripts that enabled automated project managment support that we added, as well as support for deploying our visualisation tools and the paper. Over the course of the project, the `leanproject` template in-turn received substantial new additions. One elementary example is the addition of git pre-push hooks, which are scripts that perform a basic sanity check on the local working copy of a contributor before pushing their contributions to the central GitHub repository.

4.4. **The *Lean* Zulip chat forum.** The *Lean* community traditionally congregates on the leanprover Zulip chat forum[9]. Our project was coordinated and organised primarily from this forum. At the beginning we created a channel called `Equational`. Zulip allows the creation and management of discussion topics within the scope of a channel. We made extensive use of the Zulip channel for several purposes. In the beginning it became the gathering point for new contributors. The new contributions process was designed and discussed on this forum. Later, topics were created for each specific technical topic, including the metatheory and its formalization, specific design decisions, specific implications and anti-implications, design of tools, etc. As shown in Figure 7, the Zulip chat served as the beginning of the contributions process for each piece of the project. Contributors first discussed their proposed contributions

[9] `leanprover.zulipchat.com`

or specific problems they tackled on Zulip before following the steps of claiming tasks on GitHub, writing a blueprint write up and/or formalization.

4.5. **Organising the collaboration : the precedent set by the PFR project.** When five people collaborate in person, splitting up the research on a question into subtasks and assigning them to collaborators can be accomplished by discussion and consensus. When there are more than fifty collaborators working together online, a more systematic approach is required. In previous formalization projects such as the formalization of the proof of the Polynomial Freiman–Rusza (PFR) conjecture [21], tasks were managed over the *Lean* zulipchat forum. The organiser of the project, Terence Tao, posted a series of message threads. Each thread corresponded to a list of outstanding tasks. These tasks were then claimed by collaborators on Zulip. The claims were recorded on a first-come first-served basis by the organiser by tagging the respective users against the tasks. Contributors could claim any open task and disclaim tasks if they couldn't finish it, with the organiser keeping track of these requests. This system allowed contributors to take their time to flesh out their work, without worrying about competing claims to the same task. Further, it helped the organisers track the task assignment and communicate with the respective collaborators to track and ascertain progress. Unfortunately, this involved a lot of manual and time-consuming management of the task list by organisers. In this project, we automated several pieces of this approach. This freed up organisers to help contributors and review their contributions.

4.6. **Organizing the collaboration in this project.** We adopted tools that are familiar to software engineers as ticket systems but are also known in the wider world of industrial production, such as the kanban system. Our project dashboard was built using the GitHub projects feature. We were able to encode some pieces of our automation using the standard GitHub-provided interface. For the rest, we relied on *continuous integration* scripts (hereon CI). The exact flow of contributions is specified in the `CONTRIBUTING.md` file of the project repository [13]. Briefly,

(1) Tasks were proposed by organisers. A contributor might start a discussion on Zulip or raise an issue on GitHub to prompt the organisers to launch tasks.
(2) Contributors could then claim tasks with a comment under the task. The CI ensured that at most one contributor could claim a task at any time.
(3) Contributors could then work on the task and propose a corresponding pull request.
(4) Upon completion of the task, the pull request received reviews, while the CI automatically checked that the project compiled and passed additional checks such as *Lean*'s environment replay tool *leanchecker* and the semi-external checker *lean4lean* [16].
(5) If all was well, the PR was merged onto the main branch of the project repository.

At any point in this process, the contributor could disclaim the task or replace a proposed PR with an alternative. In addition, organisers could always step in to fix any errors that occurred and follow up with contributors. Each of the steps described above happened automatically, triggered by a well-defined set of actions described in the `CONTRIBUTING.md` file. The typical workflow of this process is shown in the flowchart in Figure 7. The figure omits error handling and situations where organisers might manually intervene. The user

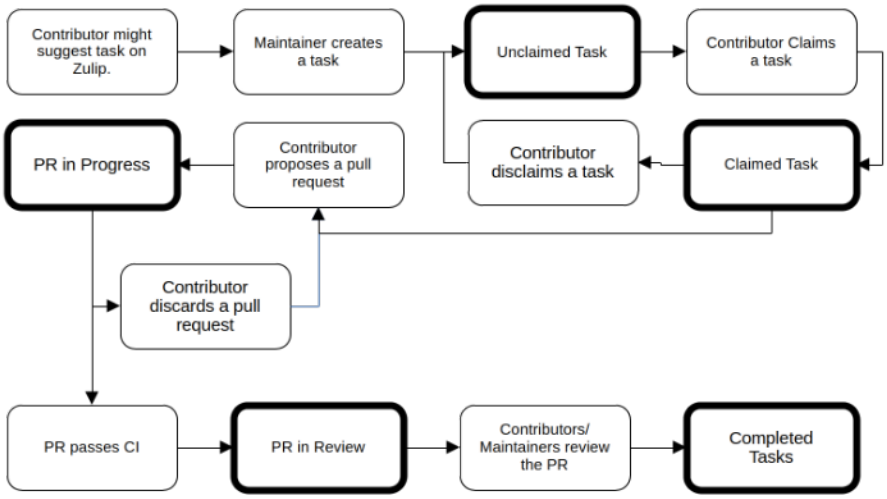


FIGURE 7. A partial flowchart of the automated task management process. Each task corresponds to an issue. A pull request is created to resolve tasks and once a pull request linked to a task is merged, the task is considered complete. The thick boxes represent states of the project dashboard represented by task columns. The movement of tasks between these states is automated by the CI which is triggered upon specific actions performed by contributors on the respective GitHub issues and pull requests. A more detailed description is found in the contributions file of the GitHub file, named `CONTRIBUTING.md` by convention.

interface to this project management is the GitHub project dashboard, of which we include a snapshot in Figure 8

We note that our method has since been adopted by other major formalization projects including the one to formalize Fermat's Last Theorem [15].

4.7. **Trusting ITPs to scale collaboration.** In our project we used the interactive theorem prover (hereon ITP) *Lean* 4 [50] precisely to address these issues of scaling. The contents of this section are common knowledge in the ITP and ITP-adjacent research communities. The exposition is intended to be useful to a user of ITPs.

At its core, an interactive theorem prover implements an expressive logic, encoded in a suitable choice of mathematical foundations. *Lean* 4 has the calculus of constructions extended by inductive types as its core logic. This logic is sufficiently powerful to express mathematical definitions and theorems for almost all areas of mathematical interest, while being relatively spartan and easy to write proof checkers for. Additionally, modern ITPs provide a convenient programming language which helps express mathematical ideas in a

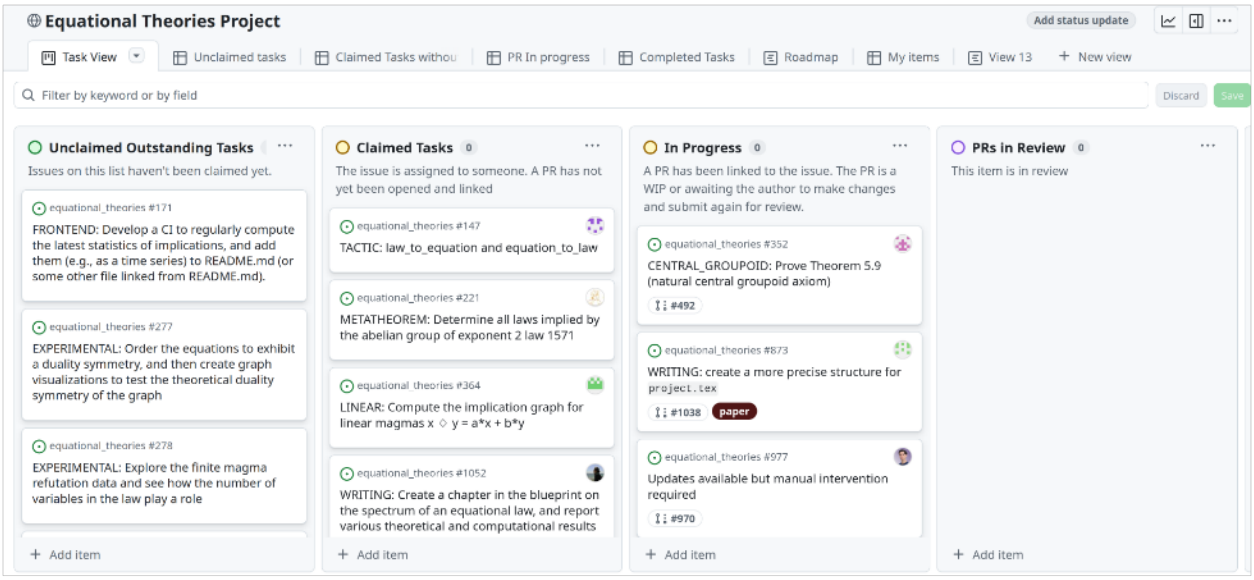


FIGURE 8. A snapshot of the project dashboard as of July 30th 2025

syntax closer to a mathematician's intuition than would be permitted by raw logical terms. A subset of ITPs like *Lean*, *Rocq* (formerly *Coq*), and *Isabelle* go one step further and provide the means to generate proofs through so-called *tactics*. There are usually numerous tactics, each specialised for specific proof generation methods. Among other things, they search mathematical libraries, simplify expressions, and identify lemmas and hypotheses to make progress in proofs. The proofs generated by this overlying programming machinery are terms in the core logical calculus which are checked mechanically by the proof checker. But in a large project, there is more to trust. It helps to understand the nature and limits of trust one can place on ITPs.

For our purposes, *Lean* consists of three pieces:

(1) A core proof checker called the *kernel*. This checker encodes a typed λ-calculus that is sufficiently expressive for most mathematical purposes. Without getting into details, theorems are encoded as a formal specification of the intended theorem statement. Proofs are encoded as deduction trees using known lemmata, acceptable axioms, and inference rules. The kernel checks whether this deduction tree constitutes a correct deduction in the context of existing theorems and lemmata, called the *environment*. Once a theorem's proof has been checked, it is added to the environment and can be used for constructing subsequent proofs. In reality, *Lean* allows users some flexibility in adding declarations into its environment without checking their types.
(2) A sophisticated programming language in which users express their definitions and theorems. Programs in this language are said to be *elaborated* to produce definitions and proofs in the core logic that the kernel can check.
(3) A compiler which compiles executable *Lean* code into reasonably efficient C code.

Of these three pieces, *the kernel is the smallest and most trusted piece.* Programs in the programming language are translated by an *elaborator* to the spartan language of the kernel. In reality the elaborator does much more, but the key takeaways are the following:

- The kernel only verifies programs of a very simple barebones language. A proof verified by *Lean* can be trusted *modulo* the correctness of the kernel implementation. This caveat can be usually dropped because, the kernel is also one of the most battle-hardened pieces of an ITP.
- The kernel checks proofs against a given specification. This means that if the formal theorem statement itself is flawed or incorrectly uses other definitions, a correct and verified proof of this theorem would be mathematically meaningless. The formal statement of a theorem gives expression to a mathematician's intuition and intention, and as such, the only check against mistakes in this area come from human review. *Lean* cannot offer any guarantees against false statements written by its users or artificial intelligence tools.
- The higher level programming language can in-principle, generate arbitrary deductions. Their correctness is ultimately validated by the kernel. This gives the higher level programming language more leeway in generating possibly incorrect deductions.
- Ideally the environment only consists of definitions and lemmata already checked by the kernel. Thus their correct usage in the proof of subsequent theorems is valid. However this assumption is often not entirely true in practical implementations of theorem provers for efficiency reasons. In this case, trust in the kernel is restored by replaying each assumption from scratch in the kernel. This can be accomplished by external checkers. Such checkers, which are ideally independent implementations of the *kernel*, can also guard against implementation mistakes in the kernel. In this project we used the environment replay tool *lean4checker* and the *lean4lean* [16] tool. To our knowledge this was the first such use of the *lean4lean* tool.

Remark 4.1. A bug in the kernel that allows false statements to be proved is usually called a *soundness* bug. Concretely, a soundness bug can be exploited to produce a kernel-certified proof of the proposition `False`. Such bugs are rare but not entirely non-existent. This is distinct from being able to proof `False` by simply assuming contradictory or false statements. A proof of the proposition `False` implies a proof of any statement by *ex falso quod libet.*

Users of *Lean* only interact directly with the higher level programming language and usually get confirmation of the correctness of their proofs through the editor interface. Further, collaborators on a project such as ours are likely to deploy automated tools such as SAT solvers, SMT solvers, first-order theorem provers, and perhaps even modern AI tools. These tools usually produce proof certificates which are imported or inserted into a *Lean* source file. Some modern AI tools are integrated into code editors which might automatically produce or edit even the statements of theorems and the definitions they deal with. Given the limits to trust mentioned above, a productive and useful collaboration using *Lean* also requires a collaboration and verification infrastructure combining human effort and automated tools. It is in this context that we discuss the project infrastructure. It is a concrete answer to the questions posed by one of the authors at the beginning of this project [62] that combines tools from the ITP community and the software engineering community. All these tools already

exist. The goal of this exposition is to explain how they address the concerns described above.

The non-Lean pieces: While *Lean* can check proofs of theorems up to the limitations described above, a project of this scale involves the use of several non-*Lean* tools. For example there are tools which extract the proven implications and anti-implications. There are tools which construct various visualisations. There are also metaprograms which call external automated theorem provers, and extract proof certificates from them to construct a *Lean* proof out of them. In keeping with the garbage-in garbage-out principle, if these tools get spurious inputs and throw spurious outputs, *Lean* can only tell us that the proofs are incorrect after the formal proofs have been translated to *Lean*. It cannot, for instance, stop us from generating a large number of spurious conjectures that misguide contributors because of a simple index error in an array. Further, the continous integration scripts that run *Lean* and check the *Lean* code with external checkers are not formally verified. Such bugs can only be unconvered by empirical testing and user reports. This highlights a basic caveat when using interactive theorem provers. These tools check something highly specific, a proof, against a specification. Contributors are responsible for the correctness of everything else. This makes the role of organisers and maintainers especially important.

4.8. **The Role of organisers and maintainers.** As mentioned before, the correctness of the project depends on a lot of moving pieces, many of which cannot be guaranteed to be correct or functional by *Lean*. In this section, we give a brief description of the variety of tasks that need to be performed by maintainers. We wish to emphasize that the role played by maintainers is akin to that played by the principal investigator and senior postdoctoral researchers in a large experimental project, in that they need to understand the big picture and mathematical details of the project to a reasonable extent and be capable of making highly technical decisions, either themselves, on in consultation with subject experts. Further, they are likely to have limited time to get involved in highly specific details of the project, and while they might make technical contributions, most of their time will be spent managing and organising tasks for other contributors. The role requires a combination of mathematical research skills and capacity with software engineering tools.

The organisers and maintainers have several tasks in a project such as ours.

- They are responsible for monitoring the Zulip chat and onboarding new contributors.
- They are responsible for creating new tasks based on requirements and Zulip discussions, and ensuring that they are properly assigned.
- They are responsible for ensuring that all the project automation functions smoothly and step in when an issue is detected. It greatly helps to have a geographically distributed set of maintainers across timezones to help fix issues at any time of the day.
- They are responsible for reviewing all code, both *Lean* proofs and theorems, and non-*Lean* scripts. This includes ensuring that the automation to build and check proofs, compile the documentation and blueprint, and test and run scripts works smoothly. When necessary they must be willing to step in and build or repair the automation.

- They are responsible for reviewing and developing the basic definitions and theorems. As mentioned before, *Lean* takes definitions and theorems for granted. Thus maintainers need to be familiar with both the mathematical content and good ways of expressing this content in *Lean*. Being proficient in *Lean* internals is helpful for maintainers in identifying anti-patterns like the use of certain tactics that lead to trusting the *Lean* compiler, such as `native_decide`.
- Maintainers are responsible for helping contributors who might get stuck in a proof. Other contributors may also assist in such matters, but ultimately it is up to the maintainers to ensure progress.
- Experienced maintainers might also offer suggestions and guidance on how to produce shorter or more elegant proofs.
- They are responsible for ensuring that some basic standards are met in proof blocks that make proofs robust to upstream changes. For instance, non-terminal uses of the `simp` tactic must be replaced with the `simp only` tactic with an explicit list of lemmas used. Otherwise, changes in the behaviour of upstream libraries can change how the tactic works and affect the correctness of the proof, when the *Lean* toolchain is updated.
- They are responsible for maintaining some record of the progress of the project, Projects on this scale can take a long time and it can become hard to remember how the project progressed. It would be extremely tedious to try to reconstruct real-time impressions long after the fact just from the GitHub commit history and Zulip chat archive. Thus it is extremely helpful if maintainers keep and publish regular logs of activities at a frequency that is proportionate to the scale of activity in the project. In our case, the main organiser maintained a daily log of activities during the busiest part of the project, which became less frequent as the progress rate slowed towards the end.

In this project, we were still learning a lot of the aforementioned lessons. A key learning from this project is that it really helps to have a maintainer structure in place before the project begins, rather than inventing one on the fly. Of course, new contributors can be onboarded as maintainers as necessary. But a small maintainer team in the initial stages can hamper proper review processes and result in suboptimal design choices in the *Lean* formalization that become hard to undo later; for instance, we developed a theory of free magmas before realizing that Mathlib already had this concept, but by that point we did not feel it worth the effort to refactor the existing code to use the Mathlib version. As one example, the initial list of equations were translated and put into one *Lean* file as opposed to several. This created excessively large *Lean* files which could have been managed better with a better file organisation.

In conclusion, we wish to emphasize that as projects scale, the administrative aspects of the project assume non-trivial importance. They require people who are proficient in at least part of the research topic, technical tools, and administrative matters. Setting these processes well in advance of announcing the project and inviting contributors should lead to a smoother project.

5. Counterexample constructions

In this section we collect the various techniques developed in the ETP to construct counterexamples to various implications $\mathrm{E} \models \mathrm{E}'$.

5.1. **Finite magmas.** A finite magma $\mathcal{M}$ of size n can be assumed without loss of generality to have carrier $\{1, \dots, n\}$ and described by specifying the multiplication table $\diamond\colon \{1, \dots, n\} \times \{1, \dots, n\} \to \{1, \dots, n\}$. By generating a list of all the equational laws $\mathrm{E}j$, $j = 1, \dots, 4694$ satisfied by this magma, one can create refutations: if $\mathcal{M} \models \mathrm{E}j$ and $\mathcal{M} \not\models \mathrm{E}k$, then clearly $\mathrm{E}j \not\models_{\mathrm{fin}} \mathrm{E}k$ and hence also $\mathrm{E}j \not\models \mathrm{E}k$. (As mentioned previously, these statements were organized in Lean using the `Facts` statement.) It is feasible to brute force over all $\sum_{n=2}^{4} n^{n^2} \approx 4.3 \times 10^9$ non-trivial magmas of size at most 4 to obtain many refutations of this type. By performing brute force over all magmas up to size 4, a total of 13 632 566 implications (61.9% of all implications, and 96.3% of the false ones) can be refuted with 524 distinct magmas. Of these implications, 13 345 053 were refuted with magmas of size[10] 3, with the remaining 415 293 requiring magmas of size 4. Performing this search took 165 CPU-hours.

However, it is not feasible to exhaustively search over the $5^{5^2} \approx 3 \times 10^{17}$ magmas of size 5, even after quotienting out by isomorphism and symmetry (which roughly saves a factor of $5! \times 2 = 240$). Randomly sampling such magmas did not produce significant refutations, as random magmas of size 5 tended to satisfy few laws, and the set of laws covered were usually also exhibited by smaller magmas. A more fruitful approach was to randomly sample from magmas with additional properties that encouraged satisfiability of a greater set of laws. These included linear and quadratic magmas (discussed below), and cancellative magmas. On the other hand, some classes of magmas, such as commutative magmas, ended up producing a disappointingly small number of additional refutations.

For specific refutations, it was sometimes possible to locate a finite example with an ATP, particularly if one also imposed additional axioms (e.g., an idempotence axiom $x = x \diamond x$) that one suspected would be useful; see Section 7 for further discussion. For medium-sized magmas (of size $n = 5, 6, 7, 8$), this appeared to be a more efficient approach than brute force exhaustion of all such magmas.

It is a result of Kisielewicz [33] that every law $\mathrm{E}n$ with $n \leq 4694$ is either equivalent to the singleton law E2, or else has a non-trivial finite model; in other words, the implications $\mathrm{E}n \models \mathrm{E}2$ and $\mathrm{E}n \models_{\mathrm{fin}} \mathrm{E}2$ are equivalent for $n \leq 4694$. Our brute force search revealed that in the latter case there is always a model of size $2 \leq n \leq 5$, with the lone exception of E1286 (and its dual E2301), for which the smallest non-trivial finite model was of size 7, as presented in Example 5.2 below. In fact, most of the 4694 laws either only had trivial models, or had a size 2 model, as shown in Table 1.

Remark 5.1. The laws satisfied by a given finite magma $\mathcal{M}$ need not be finitely axiomatizable. The smallest example is the three-element magma $\{0, 1, 2\}$ with $1 \diamond 2 = 1$, $2 \diamond 1 = 2 \diamond 2 = 2$, and $x \diamond y = 0$ for all other x, y [51]. It was also shown in [52] that "almost

[10] For an earlier computer exploration of size 3 magmas, see [11].

TABLE 1. Number of laws of order at most 4 whose smallest non-trivial model (if any) is of a given size.

Size of smallest non-trivial model	Number of laws
Trivial only	1496
2	3136
3	32
4	14
5	14
7	2

all" magmas $\mathcal{M}$ (in a certain precise sense) are idemprimal[11], which implies that their laws are finitely axiomatizable, and all other finite magmas satisfying these laws are isomorphic to powers of $\mathcal{M}$.

5.2. **Linear models.** As it turns out, a particularly fruitful source of counterexamples is the class of *linear magmas*, where the carrier M is a ring (which may be commutative or noncommutative, finite or infinite), and the operation $\diamond$ is given by $x \diamond y = ax + by$ for some coefficients $a, b \in M$; one can also generalize this slightly to *affine magmas*, in which the operation is given by $x \diamond y = ax + by + c$, but for simplicity we shall focus on linear magmas here. It is easy to see that in a linear magma, any word $w(x_1, \ldots, x_n)$ of n indeterminates also takes the linear form

$$w(x_1, \ldots, x_n) = \sum_{i=1}^{n} P_{w,i}(a, b) x_i$$

for some (possibly noncommutative) polynomial $P_{w,i}$ in a, b with integer coefficients. Thus, a linear magma will satisfy an equational law $w_1 \simeq w_2$ if and only if the pair (a, b) lies in the (possibly noncommutative) variety

$$V_{w_1,w_2}(M) := \{(a, b) \in M \times M : P_{w_1,i}(a, b) = P_{w_2,i}(a, b) \text{ for all } i\} \subseteq M^2. \tag{3}$$

As such, a necessary condition for such a law $w_1 \simeq w_2$ to entail another law $w_1' \simeq w_2'$ is that one has the inclusion

$$V_{w_1,w_2}(M) \subseteq V_{w_1',w_2'}(M)$$

for all rings M. For commutative rings, this criterion can be checked in an automatable fashion by standard Gröbner basis techniques; in the noncommutative case one can use methods such as the diamond lemma [10].

Example 5.2 (Commutative counterexample)**.** For the law $x \simeq y \diamond (((x \diamond y) \diamond x) \diamond y)$ (E1286), the variety (3) can be computed to be

$$\{(a, b) \in M \times M : 1 = ba^3 + bab, 0 = a + ba^2b + b^2\}$$

[11] A magma is idemprimal if every idempotent function is expressed by a term. This is weaker than being primal (in which *every* function is expressible by a term), but stronger than being quasiprimal (in which the *discriminator* $f(a, b, c)$, defined to equal c if $a = b$ and a otherwise, is expressed by a term); quasiprimality is sufficient to show that all other finite magmas satisfying the laws of $\mathcal{M}$ are isomorphic to products of submagmas of $\mathcal{M}$. By combining these observations with [55], one can also show that quasiprimal finite magmas can be axiomatized by a single equation. We thank Stanley Burris for these comments, as well as the other references and observations in this remark.

while the variety for the idempotent law E3 is

$$\{(a,b) \in M : a+b=1\}.$$

Thus, to show that E1286 does not entail E3, it suffices to locate elements a, b of a ring M for which one has $1 = ba^3 + bab$, $0 = a + ba^2b + b^2$, and $a + b \neq 1$. Here one can take a commutative example, for instance when $M = \mathbb{Z}/p\mathbb{Z}$ and $(p, a, b) = (11, 1, 7)$.

Example 5.3 (Noncommutative counterexample)**.** For the law $x \simeq y \diamond ((y \diamond (x \diamond z)) \diamond z)$ (E1117), the variety (3) can be computed to be

$$\{(a,b) \in M \times M : 1 = baba, 0 = a + ba^2, 0 = bab^2 + b^2\}$$

while the variety for $x \simeq (x \diamond ((x \diamond x) \diamond x)) \diamond x$ (E2441) is

$$\{(a,b) \in M \times M : 1 = a^2 + aba^2 + abab + ab^2 + b\}.$$

Observe that if $ba = -1$, then (a, b) automatically lies in the first set, and lies in the second set if and only if $(ab+1)(b-1) = 0$. One can then show that E1117 does not imply E2441 by setting $a = L$, $b = -R$ where L, R are the left and right shift operators respectively on the ring of integer-valued sequences $\mathbb{Z}^{\mathbb{N}}$. With some *ad hoc* effort one can convert this example into a less linear, but simpler (and easier to formalize) example, namely the magma with carrier $\mathbb{Z}$ and operation $x \diamond y = 2x - \lfloor y/2 \rfloor$.

Remark 5.4. As essentially observed in [6], if there is a commutative linear counterexample to an implication $\mathrm{E} \models \mathrm{E}'$, then by the Lefschetz principle this counterexample can be realized in a finite field $\mathbb{F}_q$ for some prime power q (and by the Chebotarev density theorem one can in fact take q to be a prime, so that the carrier is of the form $\mathbb{Z}/p\mathbb{Z}$ for some prime p), so that one also has $\mathrm{E} \models_{\mathrm{fin}} \mathrm{E}'$. As such, we have found that an effective way to refute implications by the commutative linear magma method is to simply perform a brute force search over linear magmas $x \diamond y = ax + by$ in $\mathbb{Z}/p\mathbb{Z}$ for various triples (p, a, b).

On the other hand, the refutations obtained by noncommutative linear constructions need not have a finite model. For instance, consider the refutation E1117 $\not\models$ E2441 from Example 5.3. The law E1117 can be rewritten as $L_y R_z L_y R_z x = x$. This implies that R_z is injective and L_y is surjective for all y, z. For finite magmas $\mathcal{M}$, this then implies that the L_y, R_z are in fact invertible, and hence we have also $R_z L_y R_z L_y x = x$, which implies E2441 by setting $x = y = z$. Thus, the refutation E1117 $\not\models$ E2441 is "immune" to finite counterexamples.

Remark 5.5. One can also consider nonlinear magma models, such as quadratic models $x \diamond y = ax^2 + bxy + cy^2 + dx + ey + f$ in a cyclic group $\mathbb{Z}/N\mathbb{Z}$. For small values of N, we have found such models somewhat useful in providing additional refutations of implications $\mathrm{E} \models_{\mathrm{fin}} \mathrm{E}'$ beyond what can be achieved by the linear or affine models. However, as the polynomials associated to a word $w(x_1, \ldots, x_n)$ tend to be of high degree (exponential in the order of the word), it becomes quite rare for such models to satisfy a given equation E when N is large.

Remark 5.6. One can also consider the seemingly more general linear model $x \diamond y = ax + by$, where the carrier M is now an abelian group, and a, b act on M by homomorphisms, that is to say that they are elements of the endomorphism ring $\mathrm{End}(M)$. However, this leads to exactly the same varieties (3) (where M is now replaced by the endomorphism ring $\mathrm{End}(M)$) and so does not increase the power of the linear model for the purposes of refuting implications.

On the other hand, there are certainly some refutations E $\not\models$ E′ of implications that are "immune" to both commutative and noncommutative linear models, in the sense that all such models that satisfy E, also satisfy E′. One such example is the refutation E1485 $\not\models$ E151, which we discuss further in Section 5.4 below.

5.3. **Translation-invariant models.** It is natural to look for counterexamples amongst magmas that satisfy a large number of symmetries. One such class of counterexamples are *translation-invariant models*, in which the carrier M is a group, and the left translations of this group form isomorphisms of the magma $\mathcal{M}$. In the case of an abelian group $M = (M, +)$, such models take the form

$$x \diamond y = x + f(y - x) \tag{4}$$

for some function $f\colon M \to M$; in the case of a non-abelian group $M = (M, \cdot)$, such models instead take the form

$$x \diamond y = x f(x^{-1} y). \tag{5}$$

For such models, the verification of an equational law in n variables corresponds to a functional equation for f in $n-1$ variables, as the translation symmetry allows one to normalize one variable to be the identity (say). This can simplify an implication to the point where an explicit counterexample can be found. These functional equations are trivial to analyze when $n = 1$. For $n = 2$, they are not as trivial, but still quite tractable, and has led to several refutations in practice. The method does not appear to be particularly effective for $n > 2$ due to the complexity of the functional equations.

Example 5.7 (Abelian example)**.** For the law $x \simeq (x \diamond y) \diamond ((x \diamond y) \diamond y)$ (E1648), we apply the abelian translation-invariant model (4) with $y = x + h$ to obtain

$$x \diamond y = x + f(h)$$
$$(x \diamond y) \diamond y = x + f(h) + f(h - f(h))$$
$$(x \diamond y) \diamond ((x \diamond y) \diamond y) = x + f(h) + f(f(h - f(h)))$$

so that this model satisfies E1648 if and only if the functional equation

$$f(h) + f(f(h - f(h))) = 0$$

holds for all $h \in M$. Similarly, the law $x \simeq (x \diamond (x \diamond y)) \diamond y$ (E206) is satisfied if and only if

$$f(f(h)) + f(h - f(f(h))) = 0$$

for all $h \in M$. One can now check that the function $f\colon \mathbb{Z} \to \mathbb{Z}$ defined by $f(h) := -\mathrm{sgn}(h)$ (thus $f(h)$ equals -1 when h is positive, $+1$ when h is negative, and 0 when h is zero) satisfies the first functional equation but not the second, thus establishing that E1648 $\not\models$ E206.

Example 5.8 (Non-abelian example)**.** We now obtain the opposite refutation E206 $\not\models$ E1648 to Example 5.7 using the non-abelian translation-invariant model. By similar calculations to before, we now seek to find a function $f\colon M \to M$ on a non-abelian group $(M, \cdot)$ that satisfies the functional equation

$$f(f(h)) f(f(f(h))^{-1} h) = 1 \tag{6}$$

for all $h \in M$, but fails to satisfy the functional equation

$$f(h) f(f(f(h)^{-1} h)) = 1 \tag{7}$$

for at least one $h \in M$. Now take M to be the group generated by three generators a, b, c subject to the relations $a^2 = b^2 = c^2 = 1$, or equivalently the group of reduced words in a, b, c with no adjacent letters in the word equal. We define

$$f(1) = 1, f(a) = b, f(b) = c, f(c) = a$$

and then $f(aw) = a$ for any non-empty reduced word w not starting with a, and similarly for b and c. The equation (6) can be checked directly for $h = 1, a, b, c$. If $h = aw$ with w non-empty, reduced, and not starting with a, then $f(f(h))^{-1} = f(f(h)) = b$ and $f(f(f(h))^{-1}h) = f(baw) = b$, giving (6) in this case, and similarly for cyclic permutations. Meanwhile, (7) can be checked to fail for $h = a$.

Remark 5.9. The construction in Example 5.8 also has the following more "geometric" interpretation. The carrier M can be viewed as the infinite 3-regular tree, in which every vertex imposes a cyclic ordering on its 3 neighbors (for instance, if we embed M as a planar graph, we can use the clockwise ordering). For $x, y \in M$, we then define $x \diamond y$ to equal x if $x = y$. If y is instead a neighbor of x, we define $x \diamond y$ to be the next neighbor of x in the cyclic ordering. Finally, if y is distance two or more from x, we define $x \diamond y$ to be the neighbor of x that is closest to y. One can then check that this model satisfies (6) but not (7).

Remark 5.10. These constructions are necessarily infinitary in nature, because E206 and E1648 can be shown to be equivalent for finite magmas. Indeed, E206 can be written as $x = R_y L_x L_x y$, which implies that R_y is surjective, hence injective, on a finite magma; writing $x = R_y z$ we conclude that $R_y z = R_y L_{z \diamond y} L_{z \diamond y} y$ and hence $z = L_{z \diamond y} L_{z \diamond y} y$, giving E1648. The opposite implication is similar (using E1648 to show that R_y is injective, hence surjective), and is left to the reader.

Some refutations $\mathrm{E} \not\models \mathrm{E}'$ are "immune" to translation-invariant models, in the sense that any translation-invariant model that satisfies E, also satisfies E′. One obstruction is that for such models, the squaring map S is necessarily an invertible map, since $Sx = x + f(0)$ in the abelian case and $Sx = xf(1)$ in the non-abelian case. On the other hand, adding the assumption of invertibility of squares can sometimes force the implication $\mathrm{E} \models \mathrm{E}'$ to hold. For instance, the commutative law $\mathrm{x} \diamond (\mathrm{y} \diamond \mathrm{y}) \simeq (\mathrm{y} \diamond \mathrm{y}) \diamond \mathrm{x}$ (E4482) for a square and an arbitrary element will imply the full commutative law E43 for translation-invariant models due to the surjectivity of S, but does not imply it in general (as one can easily see by considering models where S is constant).

5.4. **The twisting semigroup.** Suppose one has a magma $\mathcal{M}$ satisfying a law E, that also enjoys some endomorphisms $T, U : \mathcal{M} \to \mathcal{M}$. Then one can "twist" the operation $\diamond$ by T, U to obtain a new magma operation

$$x \diamond' y := Tx \diamond Uy. \tag{8}$$

If one then tests whether this new operation $\diamond'$ satisfies the same law E as the original operation $\diamond$, one will find that this will be the case provided that T, U satisfy a certain set of relations. The semigroup generated by formal generators T, U with these relations will be called the *twisting semigroup* $\mathrm{Twist}_{\mathrm{E}}$ of E. This can be best illustrated with some examples.

Example 5.11. We compute the twisting semigroup of $x \simeq (y \diamond x) \diamond (x \diamond (z \diamond y))$ (E1485). We test this law on the operation (8), thus we consider whether

$$x = (y \diamond' x) \diamond' (x \diamond' (z \diamond' y))$$

holds for all $x, y, z \in M$. Substituting in (8) and using the homomorphism property repeatedly, this reduces to

$$x = (T^2 y \diamond TUx) \diamond (UTx \diamond (U^2 Tz \diamond U^3 y)).$$

If we impose the conditions $TU = UT = \mathrm{id}$, $T^2 = U^3$, then this equation would follow from E1485 (with x, y, z replaced with $TUx = UTx = x$, $T^2 y$, and $U^2 Tz$ respectively). Thus the twisting semigroup $\mathrm{Twist}_{\mathrm{E1485}}$ of E1485 is generated by two generators T, U subject to the relations $\mathrm{TU} = \mathrm{UT} = 1$, $\mathrm{T}^2 = \mathrm{U}^3$. This is a cyclic group of order 5, since the relations can be rewritten as $\mathrm{T}^5 = 1$, $\mathrm{U} = \mathrm{T}^{-1}$.

Now consider $x \simeq (x \diamond x) \diamond (x \diamond x)$ (E151). Applying the same procedure, we arrive at

$$x = (T^2 x \diamond TUx) \diamond (UTx \diamond U^2 x)$$

so the twisting group $\mathrm{Twist}_{\mathrm{E151}}$ is generated by two generators T, U subject to the relations $\mathrm{TU} = \mathrm{UT} = \mathrm{T}^2 = \mathrm{U}^2 = 1$. This is a cyclic group of order 2, since the relations can be rewritten as $\mathrm{T}^2 = 1$, $\mathrm{U} = \mathrm{T}$.

Suppose the twisting semigroup $\mathrm{Twist}_{\mathrm{E}}$ is not a quotient of $\mathrm{Twist}_{\mathrm{E}'}$, in the sense that the relations that define $\mathrm{Twist}_{\mathrm{E}'}$ are not satisfied by the generators of $\mathrm{Twist}_{\mathrm{E}}$. Then one can often disprove the implication $\mathrm{E} \models \mathrm{E}'$ by attempting the following procedure.

- First, locate a non-trivial magma $\mathcal{M} = (M, \diamond)$ satisfying the law E. Then the Cartesian power $M^{\mathrm{Twist}_{\mathrm{E}}}$ of tuples $(x_W)_{W \in \mathrm{Twist}_{\mathrm{E}}}$, with the pointwise magma operation, will also satisfy E.
- Furthermore, this Cartesian power admits two endomorphisms T, U defined by
$$T(x_W)_{W \in \mathrm{Twist}_{\mathrm{E}}} = (x_{W\mathrm{T}})_{W \in \mathrm{Twist}_{\mathrm{E}}}; U(x_W)_{W \in \mathrm{Twist}_{\mathrm{E}}} = (x_{W\mathrm{U}})_{W \in \mathrm{Twist}_{\mathrm{E}}},$$
which satisfy the relations defining $\mathrm{Twist}_{\mathrm{E}}$.
- We now twist the magma operation $\diamond$ on $M^{\mathrm{Twist}_{\mathrm{E}}}$ by T, U to obtain a new magma operation $\diamond'$ defined by (8), that will still satisfy law E.
- Because T, U will not satisfy the relations defining $\mathrm{Twist}_{\mathrm{E}'}$, it is highly likely that this twisted operation will not satisfy E′, thus refuting the implication $\mathrm{E} \models \mathrm{E}'$. If M and the twisting semigroup were finite, this approach should also refute $\mathrm{E} \models_{\mathrm{fin}} \mathrm{E}'$.

For instance, a non-trivial finite model for E1485 is given by the finite field $\mathbb{F}_2$ of two elements with the NAND operation $x \diamond y := 1 - xy$. If we twist $\mathbb{F}_2^5$ by the left shift $T(x_i)_{i=1}^5 = (x_{i+1})_{i=1}^5$ and right shift $U(x_i)_{i=1}^5 = (x_{i-1})_{i=1}^5$, where we extend the indices periodically modulo 5, then the resulting operation

$$(x_i)_{i=1}^5 \diamond' (y_i)_{i=1}^5 := (1 - x_{i+1} y_{i-1})_{i=1}^5$$

on $\mathbb{F}_2^5$ still satisfies E1485, but does not satisfy E151, thus showing that E1485 $\not\models_{\mathrm{fin}}$ E151 and hence E1485 $\not\models$ E151. This particular implication does not seem to be easily refuted by any of the other methods discussed in this paper.

5.5. **Greedy constructions.** We have found *greedy extension methods*, or *greedy methods* for short, to be a powerful way to refute implications, especially when the carrier M is allowed to be infinite. Such constructions have a long history in model theory, with possibly the earliest[12] such construction due to Skolem [60]. A basic implementation of this method is as follows. To build a magma operation $\diamond\colon M \times M \to M$ that satisfies one law E but not another E′, one can first consider *partial magma operations* $\diamond\colon \Omega \to M$, defined on some subset Ω of $M \times M$. Thus $x \diamond y$ is defined if and only if $(x, y) \in \Omega$. A magma operation is then simply a partial operation which is *total* in the sense that $\Omega = M \times M$. We say that a partial magma operation is *finitely supported* if Ω is finite.

In the language of first-order logic (in which functions and relations must be total), it is convenient to view a partial magma operation as a ternary relation $R(x, y, z)$ on M with the axiom that $R(x, y, z) \wedge R(x, y, z') \implies z = z'$ for all $x, y, z \in M$. The support Ω is then the set of (x, y) for which $R(x, y, z)$ holds for some (necessarily unique) z, which one can then take to be the definition of $z = x \diamond y$.

We say that one partial operation $\diamond'\colon \Omega' \to M$ *extends* another $\diamond\colon \Omega \to M$ if Ω' contains Ω, and $x \diamond y = x \diamond' y$ whenever $x \diamond y$ (and hence $x \diamond' y$) are defined. Given a sequence $\diamond_n\colon \Omega_n \to M$ of partial operations, each of which is an extension of the previous, we can define the *direct limit* $\diamond_\infty\colon \bigcup_n \Omega_n \to M$ to be the partial operation defined by $x \diamond_\infty y := x \diamond_n y$ whenever $(x, y) \in \Omega_n$.

Abstractly, the greedy algorithm strategy can now be described as follows.

Theorem 5.12 (Abstract greedy algorithm)**.** *Let* E, E' *be equational laws, and let* Γ *be a theory of first-order sentences regarding a partial magma operations* $\diamond\colon \Omega \to M$ *on a carrier* M*. Assume the following axioms:*

- *(i) (Seed) There exists a finitely supported partial magma operation* $\diamond_0\colon \Omega_0 \to M$ *satisfying* Γ *that contradicts* E'*, in the sense that there is some assignment of variables in* E' *in* M *such that both sides of* E' *are defined using* $\diamond_0$*, but not equal to each other.*
- *(ii) (Soundness) If* $\diamond_n\colon \Omega_n \to M$ *is a sequence of partial magma operations satisfying* Γ *with each* $\diamond_{n+1}$ *an extension of* $\diamond_n$*, and the direct limit* $\diamond_\infty$ *is total, then this limit satisfies* E*.*
- *(iii) (Greedy extension) If* $\diamond\colon \Omega \to M$ *is a finitely supported partial magma operation satisfying* Γ*, and* $a, b \in M$*, then there exists a finitely supported extension* $\diamond'\colon \Omega' \to M'$ *of* $\diamond$ *to a possibly larger carrier* M'*, and also satisfying* Γ*, such that* $a \diamond' b$ *is defined.*

Then $\mathrm{E} \not\models \mathrm{E}'$*.*

We remark that this greedy method seems to be inherently infinitary in nature, and does not seem well adapted to refute finite magma implications $\mathrm{E} \models_{\mathrm{fin}} \mathrm{E}'$.

Proof. We work on the countably infinite carrier $\mathbb{N}$. By embedding the finitely supported operation $\diamond_0$ from axiom (i) into $\mathbb{N}$, we can assume without loss of generality that $\diamond_0$ has

[12] We thank Stanley Burris for this reference.

carrier $\mathbb{N}$. By similar relabeling, we can assume in (iii) that $M' = M$ when $M = \mathbb{N}$, since any elements of $M' \backslash \mathbb{N}$ that appear in Ω' can simply be reassigned to natural numbers that did not previously appear in Ω. We well-order the pairs in $\mathbb{N} \times \mathbb{N}$ by (a_n, b_n) for $n = 1, 2, \dots$. Iterating (iii) starting from $\diamond_0$, we can thus create a sequence of finitely supported magma operations $\diamond_0, \diamond_1, \dots$ on $\mathbb{N}$ satisfying Γ, with each $\diamond_{n+1}$ an extension of $\diamond_n$, and $a_n \diamond_n b_n$ defined for all $n \geq 1$. Then the direct limit $\diamond_\infty$ of these operations is total, and does not satisfy E′ thanks to axiom (i). On the other hand, by axiom (ii) it satisfies E, and the claim follows. □

We refer to Γ as the *rule set* for the greedy extension method. To apply Theorem 5.12 to obtain a refutation $\mathrm{E} \models \mathrm{E}'$, we have found the following trial-and-error method to work well in practice:

1. Start with a minimal rule set Γ that has just enough axioms to imply the soundness property for the given hypothesis E.
2. Attempt to establish the greedy extension property for this rule set by setting $a \diamond' b$ equal to a new element $c \notin M$, and then defining additional values of $\diamond'$ as necessary to recover the axioms of Γ'.
3. If this can be done in all cases, then locate a seed $\diamond_0$ refuting the given target E′, and `STOP`.
4. If there is an obstruction (often due to a "collision" in which a given operation $x \diamond' y$ is required to equal two different values), add one or more rules to Γ to avoid this obstruction, and return to Step 2.

As an example, we present

Proposition 5.13 (E73 $\not\models$ E4380)**.** *The law* $\mathrm{x} \simeq \mathrm{y} \diamond (\mathrm{y} \diamond (\mathrm{x} \diamond \mathrm{y}))$ (E73) *does not imply* $\mathrm{x} \diamond (\mathrm{x} \diamond \mathrm{x}) \simeq (\mathrm{x} \diamond \mathrm{x}) \diamond \mathrm{x}$ (E4380).

Proof. To build a rule set Γ that will imply E73 when total, a natural first choice would be the single rule

1. If $y \diamond (x \diamond y)$ is defined, then $y \diamond (y \diamond (x \diamond y))$ is defined and equal to x.

However, the greedy algorithm will fail just with this rule: if the partial operation has $x \diamond y$ and $z \diamond y$ both equal to some w for some $x \neq z$, then any attempt to assign a value to $y \diamond (y \diamond w)$ will lead to a contradiction, as the above rule will force $y \diamond w$ to equal both x and z. Indeed, it is clear that E73 forces all the right translation operators R_y to be injective. We therefore add this as an additional rule:

2. If $x \diamond y$ and $z \diamond y$ are defined and equal, then $x = z$.

To avoid some unwanted edge cases, it is also convenient to impose the additional rule

3. If $x \diamond y$ is defined, it is not equal to y.

Unlike Rule 2, this rule is not forced by E73, but can be enforced as part of the greedy construction.

The ruleset clearly satisfies the soundness axiom (ii) of Theorem 5.12. We now verify the greedy extension axiom (iii). Let Ω, a, b be as in that axiom. We may assume that $a \diamond b$ is undefined, since we are done otherwise. We adjoin a new element c to M to create M', and set $a \diamond' b = c$. If we also have $b = d \diamond a$ for some d (unique by Rule 2, and only possible for $a \neq b$ by Rule 3), set $a \diamond' c = d$ (this is necessary to retain Rule 1). Of course, we also set $x \diamond' y = x \diamond y$ whenever $x \diamond y$ is already defined.

Since $c \notin M$, it is clear that $\diamond'$ is a finitely supported partial magma operation on M'. It is also clear that $\diamond'$ satisfies Rule 2 and Rule 3. Now we case check Rule 1:

- Case 1: $x = c$ or $y = c$. Not possible since no left multiplication with c is defined.
- Case 2: $x \diamond' y = c$. Only possible when $x = a$, $y = b$, but then $y \diamond' (x \diamond' y)$ is undefined since $y = b \neq a$ if d is defined.
- Case 3. $y \diamond' (x \diamond' y) = c$. Only possible when $y = a$ and $x = d$, and holds in this case.
- Case 4: $x, y, x \diamond' y, y \diamond' (x \diamond' y) \neq c$: In this case $\diamond' = \diamond$ on all pairs, so the claim reduces to Rule 1 for $\diamond$, which holds by the induction hypothesis.

To conclude, we need to locate a seed $\diamond_0$ satisfying Rules 1,2,3 and containing a counterexample to E4380. One simple example is the carrier $\{0, 1, 2, 3\}$ with $0 \diamond_0 0 = 1$, $0 \diamond_0 1 = 2$, $0 \diamond_0 2 = 0$, $1 \diamond_0 0 = 3$. □

This method is not guaranteed to halt in finite time, as there may be increasingly lengthy sets of rules one has to add to Γ to avoid collisions. However, in practice we have found many of the refutations that could not be resolved by simpler techniques to be amenable to this method (or variants thereof, as discussed below).

One can automate the above procedure by using ATPs (or SAT solvers) to locate new rules that are necessary and sufficient to resolve any potential collision (and which, *a posteriori*, can be seen to be necessarily consequences of the law E). The seed-finding step (Step 3) is particularly easy to automate, and can also often be done by hand. In some cases, the SAT solver calculations provided by these methods were difficult to formalize efficiently in Lean, and so we elected in some cases to replace computer-generated rulesets with shorter human-generated versions in preparation for the formalization step.

However, in some cases we have found it necessary to add "inspired" choices of rules that were not forced by the initial hypothesis E, but which simplified the analysis by removing problematic classes of collisions from consideration. We were unable to fully automate the process of guessing such choices; however, we found ATPs very useful for testing any proposed such guess. In particular, if an ATP was able to show that the existing ruleset, together with a proposed new rule A, implied E′, then this clearly indicated that one should not add A to the rule set Γ. Conversely, if an ATP failed to establish such an implication, this was evidence that this was a "safe" rule to impose.

We also found that human verification of the greedy extension property was a highly error-prone process, as the case analysis often included many delicate edge cases that were easy to overlook. Both ATPs and the Lean formalization therefore played a crucial role in verifying the human-written greedy arguments, often revealing important gaps in those arguments that required either minor or major revisions to the rule set.

The greedy method can also be combined with the translation-invariant method, both in abelian and non-abelian settings. For instance, we can modify the proof of Theorem 5.12 to obtain the following variant:

Theorem 5.14 (Noncommutative translation-invariant greedy algorithm)**.** *Let F, F' be functional equations on groups, and let Γ be a theory of first-order sentences regarding a partial function $f\colon \Omega \to G$ on a group $(G, \cdot, \cdot^{-1}, 1)$. Assume the following axioms:*

(i) (Seed) There exists a finitely supported partial function $f_0\colon \Omega_0 \to G$ satisfying Γ that contradicts F', in the sense that there is some assignment of variables in F' in G such that both sides of F' are defined using f_0, but not equal to each other.
(ii) (Soundness) If $f_n\colon \Omega_n \to G$ is a sequence of partial functions satisfying Γ with each f_{n+1} an extension of f_n, and the direct limit f_∞ is total, then this limit satisfies F.
(iii) (Greedy extension) If $f\colon \Omega \to G$ is a finitely supported partial function satisfying Γ, and $a \in G$, then there exists a finitely supported extension $f'\colon \Omega' \to G'$ of f to a possibly larger group G', and also satisfying Γ, such that $f'(a)$ is defined.

Then $F \not\models F'$.

One can of course also develop an abelian analogue of the above theorem, in which $(G, +, -, 0)$ and $(G', +, -, 0)$ are now required to be abelian. We can then give an alternate proof of Proposition 5.13 as follows:

Second proof of Proposition 5.13. (Sketch) The functional equations associated to E73 and E4380 are $f^2(h^{-1}f(h)) = h^{-1}$ and $f^2(1) = f(1)f(f(1)^{-1})$ respectively. We apply Theorem 5.14 with the following ruleset:

1. If $f(h^{-1}f(h))$ is defined, then $f^2(h^{-1}f(h))$ is defined and equal to h^{-1}.
2. If $h^{-1}f(h)$ and $k^{-1}f(k)$ are defined and equal, then $h = k$.
3. If $f(h)$ is defined, it is not equal to h.

Axiom (ii) is clear. To verify axiom (iii), we can assume $f(h)$ is undefined, then adjoin an element c freely to G to create a larger group G', and set $f'(h) = c$. If $h = k^{-1}f(k)$ for some k (which is unique by Rule 2, and only possible for $h \neq 1$ by Rule 3), then also set $f'(c) = k^{-1}$. One can then check that axiom (iii) is satisfied. For axiom (i), take G to be a free cyclic group with one generator a, and set $f(1) = a$, $f(a) = a^3$, $f(a^3) = 1$, $f(a^{-1}) = a^3$ (say). □

More complex (and *ad hoc*) variants of the greedy algorithm are possible. In some cases, we were not able to preserve the finitely supported nature of the partial operation or partial

function, and needed to extend that partial object at an infinite number of values at each step. In other cases, one also had to add additional temporary data during the greedy process to record tasks that one wished to attend to at a later stage of the process, but could not handle immediately because it was awaiting some other operation to become well-defined. We will not attempt to survey all possible variants of this method here, but refer the reader to the ETP blueprint for further examples.

5.6. **Modifying base models.** A general technique that we have found useful in obtaining a refutation such as $\mathrm{E} \not\models \mathrm{E}'$ is to start with a simple base model $\mathcal{M} = (M, \diamond)$ that satisfies both E and E′, and modify it in various ways to preserve E, but create a violation of E′. There are many such possible modifications, but three general ways that have proven effective are as follows:

(i) Modify the magma operation $\diamond\colon M \times M \to M$ on a small set in order to violate E′, and then make further modifications as needed to recover E.
(ii) Construct an *extension* $\mathcal{N}$ of $\mathcal{M}$, equipped with a surjective magma homomorphism $\pi : \mathcal{N} \to \mathcal{M}$, and defined in terms of some additional data. Then solve for that data in such a way that $\mathcal{N}$ satisfies E but not E′.
(iii) Construct an *enlargement* $\mathcal{M}' = (M', \diamond')$ of $\mathcal{M} = (M, \diamond)$, which is a magma that contains $\mathcal{M}$ as a submagma. One needs to construct the multiplication table $\diamond$ on $(M' \times M')\backslash(M \times M)$ in order to retain E but disprove E′.

One appealing case of (ii) that our project discovered, involving a "magma cohomology" analogous to (abelian) group cohomology, is that of an *affine* extension of a magma $\mathcal{G} = (G, \diamond_G)$ by another magma $(M, \diamond_M)$ which is an abelian group M with a linear magma operation $s \diamond_M t := as + bt$ for some endomorphisms $a, b \in \mathrm{End}(M)$. One can then consider extensions with carrier $G \times M$ and magma operation

$$(x, s) \diamond (y, t) := (x \diamond_G y, s \diamond_M t + f(x, y)) \tag{9}$$

for some function $f\colon G \times G \to M$. If $(M, \diamond_M)$ and $(G, \diamond_G)$ already satisfy a law E, then this extension will also satisfy E if and only if f satisfies a certain "cocycle equation", which is a linear equation on f. One can then sometimes use linear algebra to locate an f that satisfies the cocycle equation for one law E but not another E′, thus refuting the implication $\mathrm{E} \models \mathrm{E}'$. An example is as follows:

Proposition 5.15 (E1110 $\not\models$ E1629)**.** *The law* $\mathrm{x} \simeq \mathrm{y} \diamond ((\mathrm{y} \diamond (\mathrm{x} \diamond \mathrm{x})) \diamond \mathrm{y})$ (E1110) *does not imply* $\mathrm{x} \simeq (\mathrm{x} \diamond \mathrm{x}) \diamond ((\mathrm{x} \diamond \mathrm{x}) \diamond \mathrm{x})$ (E1629)*, even for finite magmas.*

Proof sketch. Using the linear ansatz, we find that E1110 has a model $\mathcal{M}$ with carrier $\mathbb{F}_5$ (the finite field $\mathbb{Z}/5\mathbb{Z}$) with operation $x \diamond y = 3x - y$. We then apply the ansatz (9) with $G = M$. One then finds that this operation satisfies E1110 if $f\colon \mathbb{F}_5 \times \mathbb{F}_5 \to \mathbb{F}_5$ the cocycle equation

$$3f(x, x) - 3f(y, 2x) - f(3y - 2x, y) + f(y, 3y - x) = 0$$

for all $x, y \in \mathbb{F}_5$, and satisfies E1629 if f satisfies the cocycle equation

$$f(2x, 0) - f(2x, 2x) = 0$$

for all $x \in \mathbb{F}_5$. A routine symbolic algebra package computation reveals that the space of f that satisfies the former equation is a six-dimensional vector space over $\mathbb{F}_5$, which is not contained in the solution space of the latter equation, giving the claim. In fact, since these equations preserve the space of homogeneous polynomials of a fixed degree, one can use linear algebra to locate an example that is a homogeneous polynomial; one explicit choice is

$$f(x,y) = y^5 + xy^4 + x^2y^3 + 3x^3y^2 + 3x^4y. \qquad \square$$

It may be of interest to develop this theory of "magma cohomology" further, for instance by defining higher order magma cohomology groups.

Now we give an example of how method (ii) can be combined with method (i).

Proposition 5.16 (E1659 $\not\models$ E4315)**.** *The law* $x \simeq (x \diamond y) \diamond ((y \diamond y) \diamond z)$ (E1659) *does not imply* $x \diamond (y \diamond x) \simeq x \diamond (y \diamond z)$ (E4315).

Proof. There are two simple models for E1659: the model G with carrier $\mathbb{Z}/2\mathbb{Z}$ and operation $x \diamond y = x+1$, and the model $\mathcal{M}$ with carrier $\mathbb{Z}$ and operation $x \diamond y = x$. Using the ansatz (9), one can soon discover that one obtains a magma operation $\diamond : (G \times \mathbb{Z}) \times (G \times \mathbb{Z}) \to G \times \mathbb{Z}$ with $f(0,0) = f(1,0) = 0$, $f(0,1) = -1$, and $f(1,1) = 1$. This model still satisfies E4315. However, we can create a modification $\diamond'$ of $\diamond$ as follows. We will seek to violate E4315 at $x = (0,0)$, $y = (0,0)$, $z = (1,0)$, thus we want

$$(0,0) \diamond' ((0,0) \diamond' (0,0)) \neq (0,0) \diamond' ((0,0) \diamond' (1,0)).$$

We have $(0,0) \diamond (0,0) = (1,0)$ and $(0,0) \diamond (1,0) = (1,-1)$. One can try to force the counterexample by setting $(0,0) \diamond' (1,0)$ to equal $(0,0)$ instead of $(1,-1)$. However, if this is the only change we make, then we no longer satisfy E1659, since

$$(1,0) \neq ((0,0) \diamond' (1,0)) \diamond' (((1,0) \diamond' (1,0)) \diamond' (1,t))$$

for any $t \in \mathbb{N} \backslash \{0\}$. But these are the only counterexamples created involving elements in the subset $G \times \mathbb{N}$ of $G \times \mathbb{Z}$; and if one then sets $(0,0) \diamond' (1,t) = (0,0)$ for *all* $t \in \mathbb{N}$, and then restricts to $G \times \mathbb{N}$ (which is now closed under $\diamond'$), then one can check that the modified operation $\diamond'$ on this submagma now satisfies E1659 but not E4315 as required. Incidentally, this submagma is isomorphic to the magma $\mathcal{M}' = (\mathbb{Z}, \diamond'')$ with $m \diamond'' n = -m$ for $n < 0$ and $m \diamond'' n = -m - 1$ for $n \geq 0$ under the bijection that maps $(0,s)$ to s and $(1,s)$ to $-s-1$. $\square$

The specific law $x \simeq x \diamond ((y \diamond z) \diamond (x \diamond z))$ (E854) turned out to be somewhat "mutable", in the sense that one can often change a small number of entries in the multiplication table of a finite magma satisfying this law, or add rows and columns to the table, in ways that preserve the law E854. This makes the law amenable to methods (i) and (iii) to construct new models of this equation that refute various implications E854 $\not\models_{\text{fin}}$ E, for instance by starting with a model that already refuted some stronger law E′, and then attempting to modify it (possibly with ATP assistance) by some combination of methods (i) and (iii) to produce a model that violates E.

Some heuristics loosely inspired by discrete-time dynamical systems proved helpful. The idea is to generate a sequence of magmas, each of which is generated from the previous entry by applying various operations expected to increase the likelihood of finding a refutation.

This is similar to the greedy methods in 5.5, except that we require our resulting magma to be finite and completely defined, and our transformations need not be deterministic.

Since our goal is simply to find a finite model — and any candidate can be checked directly — we are not limited to transformations that can be rigorously justified. SAT solvers like *Glucose* [4, 5] inspired by the earlier *MiniSAT* [24], via convenient interfaces like *PySAT* [29, 30], other fast SAT solvers such as *Kissat* [12], counterexample finders like *Mace4* [41], and more general ATPs like *Vampire* [37] which can be used as solvers, all succeeded in finding useful magmas following this approach.

For example, suppose our goal is to show E854 $\not\models_{\text{fin}}$ E, for some E. We start with a magma table which satisfies both equations, and remove a random subset of the table entries (creating a partially defined magma). We then ask the ATP to find a magma which satisfies E854, filling in the unspecified values and potentially adding new elements. If in fact E is not implied by E854, the ATP might succeed in finding a magma refuting the implication. We may also directly insert a violation of E into the magma cells we have emptied, hoping that a consistent completion still exists. Another method, by analogy with slowly introducing forcing terms into numerical integrations while preserving adiabatic invariants, is to impose selected implications of a given equation without directly enforcing the equation itself. This can gradually drive the magmas in the sequence toward satisfying the equation without immediately imposing it.

Combining several of these techniques allowed us to find proofs of

$$\text{E854} \not\models_{\text{fin}} \text{E413}, \text{E1045}, \text{E1055}, \text{E3316}, \text{E3925}.$$

As an example of the process, we started with a magma that satisfied E854 and all the above equations, but had an E10 violation. We suspected that E10 might be relevant (and later found an argument showing multiple E10 violations would be required in a magma which showed E854 $\not\models_{\text{fin}}$ E1055), and so used *Kissat* and *Vampire* to grow E854 magmas with such violations. These larger magmas were then used as seeds for the random evolution process, and the ATPs eventually succeeded in finding magmas which satisfied E854 but refuted the implications in question.

Note that these approaches have their limitations. To be effective, it must be easy to transition between different states, which usually involves finding a magma satisfying the equation of interest or at least some related one. Equations whose finite magmas are difficult to find (e.g. E677), whether because of absolute rarity or the numerical challenges that our ATPs have in finding them without taking advantage of a structural ansatz, appear resistant to these methods in practice.

Another way to utilize (iii), which proved useful for laws that involved the squaring operator S, was to adopt a "squares first" approach in which one selected a base model $S\mathcal{M} = (SM, \diamond)$ to serve as the set of squares, then extend it to a larger model $\mathcal{M}$ with carrier $M = SM \uplus N$ by first determining what the multiplication map should be on the diagonal $\{(x, x) : x \in N\}$ (i.e., to determine the squaring map $S\colon N \to SM$), together with the values on the blocks $SM \times N$, $N \times SM$, and then finally resolve the remaining values on the $N \times N$ block. Often, versions of the greedy algorithm are useful for each of these stages of the construction. The

precise details are quite technical, particularly for the law $x \simeq (y \diamond y) \diamond ((y \diamond x) \diamond y)$ (E1729), which was the last of the equations whose implications were settled by the ETP. We refer the reader to the ETP blueprint for further details.

6. Syntactic arguments

Many proofs or refutations of implications (or equivalences) between two equational laws E, E' can be obtained from the syntactic form of the equation. We discuss some techniques here that were useful in the ETP.

6.1. **Simple rewrites.** Many equational laws E' can be formally deduced from a given law E by applying the *Lean* `rw` tactic to rewrite E' repeatedly by some forward or backward application of E applied to arguments that match some portion of E. For instance, the commutative law E43 clearly implies $x \diamond (y \diamond z) \simeq (y \diamond z) \diamond x$ (E4531) by a single such rewrite. A brute force application of such rewrite methods is already able to directly generate about 15 000 such implications, including many equivalences to the singleton law E2 and the constant law E46. After applying transitive closure, this generates about four million further such implications.

A simple observation that already generates a reasonable number of equivalences is that any equation of the form $x \simeq f(y, z, \dots)$ necessarily is equivalent to the trivial law $x \simeq y$, by transitivity; similarly, an equation of the form $f(x, y) \simeq g(z, w, \dots)$ implies $f(x, y) \simeq f(x', y')$; and so forth. Equivalences of this form were useful early in the project by cutting down the number of distinct equivalence classes of laws that needed to be studied.

6.2. **Matching invariants.** Fix an alphabet X. A *matching invariant* is an assignment $I\colon M_X \to \mathcal{I}$ of an object $I(w) \in \mathcal{I}$ in some space $\mathcal{I}$ to each word $w \in M_X$ with the property that if an equational law $w_1 \simeq w_2$ has matching invariants $I(w_1) = I(w_2)$, then the same matching $I(w_1') = I(w_2')$ holds for any consequence $w_1' \simeq w_2'$. In particular, if one has $I(w_1) = I(w_2)$ and $I(w_1') \neq I(w_2')$, then the law $w_1 \simeq w_2$ does not imply the law $w_1' \simeq w_2'$.

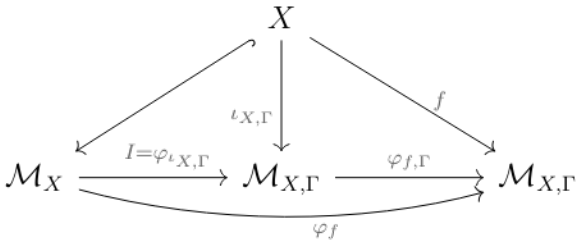
A simple example of a matching invariant is the multiplicity $(n_x)_{x \in X}$ of variables of a word: if w_1, w_2 have all variables x appear the same number of times n_x in both words, then any rewriting of a word w using the law $w_1 \simeq w_2$ will preserve this property. Hence, if w_1', w_2' do not have that each variable appear the same number of times in both words, then $w_1 \simeq w_2$ cannot imply $w_1' \simeq w_2'$. For instance, the commutative law E43 cannot imply the left-absorptive law E4.

One source of matching invariants comes from the free magma $\mathcal{M}_{X,\Gamma}$ of a theory:

Proposition 6.1 (Free magmas and matching invariants)**.** *Let $\iota_{X,\Gamma}\colon X \to M_{X,\Gamma}$ be the map associated to the free magma $\mathcal{M}_{X,\Gamma}$ for a theory Γ. Then the map $I\colon M_X \to M_{X,\Gamma}$ defined by $I(w) := \varphi_{\iota_{X,\Gamma}}(w)$ is an invariant.*

Proof. Suppose that $w_1 \simeq w_2$ entails $w_1' \simeq w_2'$, and that $I(w_1) = I(w_2)$. For any $f\colon X \to \mathcal{M}_{X,\Gamma}$, the two maps $\varphi_f, \varphi_{f,\Gamma} \circ \varphi_{\iota_{X,\Gamma}}\colon \mathcal{M}_X \to \mathcal{M}_{X,\Gamma}$ are both homomorphisms that extend

f, hence agree by the universal property of $\mathcal{M}_X$, as displayed by the following commutative diagram:



In particular, the hypothesis $I(w_1) = I(w_2)$ implies that $\varphi_f(w_1) = \varphi_f(w_2)$ for all $f\colon X \to M_{X,\Gamma}$; that is to say, the magma $\mathcal{M}_{X,\Gamma}$ satisfies the law $w_1 \simeq w_2$, and hence also $w_1' \simeq w_2'$ by hypothesis. Thus $\varphi_{\iota_{X,\Gamma}}(w_1') = \varphi_{\iota_{X,\Gamma}}(w_2')$, which gives $I(w_1') = I(w_2')$ as required. □

Example 6.2. If we take $\Gamma = \{\text{E4}\}$ to be the theory of the left-absorptive law E4 as described in Example 2.2, then the matching invariant $I(w)$ produced by Proposition 6.1 is the left-most letter of the alphabet X appearing in the word; for instance $I((\mathrm{x} \diamond \mathrm{y}) \diamond \mathrm{z}) = \mathrm{x}$. Thus, for example, the left-absorptive law E4 cannot imply the right-absorptive law E5.

Example 6.3. If we take $\Gamma = \{\text{E43}, \text{E4512}\}$ to be the theory of the commutative law E43 and the associative law E4512, then by Example 2.1, the associated invariant $I(w) = \sum_{x\in X} n_x e_x$ is the formal sum of all the generators e_x appearing in the word w, in the free abelian semigroup generated by those generators. This recovers the preceding observation that the multiplicities $(n_x)_{x\in X}$ form a matching invariant.

Example 6.4. Let $n \geq 1$ be a positive integer, and consider the theory $\Gamma = \{\text{E43}, \text{E4512}, \text{E}_n\}$ consisting of the previous theory $\{\text{E43}, \text{E4512}\}$ together with the order-n law $L_x^n y = y$. One can check that the free magma $\mathcal{M}_{X,\Gamma}$ can be described as the free abelian group of exponent n with generators $e_x, x \in X$, with associated map $\iota_{X,\Gamma}\colon x \mapsto e_x$. The associated matching invariant $I(w) = \sum_{x\in X} n_x e_x$ is essentially the multiplicities $(n_x \bmod n)_{x\in X}$ modulo n, which gives a slightly stronger criterion than the preceding matching invariant for refuting implications. For example, the cubic idempotent law $\mathrm{x} \simeq (\mathrm{x} \diamond \mathrm{x}) \diamond \mathrm{x}$ (E23) has matching invariants $e_x = 3e_x$ in the $n = 2$ case, and hence does not imply the idempotent law $\mathrm{x} \simeq \mathrm{x} \diamond \mathrm{x}$ (E3) since $e_x \neq 2e_x$ in the $n = 2$ case.

In practice, we found that these invariants could be used to establish a significant fraction of the non-implications in the implication graph, although in most cases these non-implications could also be established by other means, for instance through consideration of small finite counterexamples, especially small models of Γ.

Remark 6.5. One can also obtain matching invariants from the free objects associated to theories that involve additional operations beyond the magma operation $\diamond$, such as an identity element or an inverse operation. We leave the precise generalization of Proposition 6.1 to such theories to the interested reader.

6.3. **Canonization.** One possible way of obtaining refutations of a given implication $\mathrm{E} \models \mathrm{E}'$ between equational laws is by building a special kind of syntactic model, via certain involutions on elements of the free magma $\mathcal{M}_X$ we call *canonizers*.

Definition 6.6. A function $C\colon M_X \to M_X$ is a *canonizer* for an equation E if

(1) C is computable.
(2) if $w_1 \sim_{\mathrm{E}} w_2$, then $C(w_1) = C(w_2)$.

In fact, a canonizer is simply a matching invariant with target in M_X.

We describe a concrete strategy for building such Cs, which will require a number of definitions.

Definition 6.7. Let $R\colon M_X \to M_X$ be an arbitrary function.

- We say that R is *weakly collapsing* if for every word w, $R(w)$ is a (not necessarily proper) sub-word of w.
- A function $\theta\colon X \to M_X$ is called a *substitution*, and we can extend θ to an endomorphism $\varphi_\theta\colon \mathcal{M}_X \to \mathcal{M}_X$ in the usual way. We write $\varphi_\theta w$ instead of $\varphi_\theta(w)$ for application of substitutions.
- If E is the equation $l \simeq r$, and l is not a variable, we say that R is a *weak canonizer* if for any substitution θ, $R(\varphi_\theta l) = \varphi_\theta r$.
- Finally we say that R is *non-overlapping* for E if for every word $w \in M_X$ which is a strict sub-word of l that is not in X, and any substitution θ, $R(\varphi_\theta w) = \varphi_\theta w$.

We can then define $C_R\colon M_X \to M_X$ as follows:

$$C_R(\mathrm{x}) \coloneqq \mathrm{x} \tag{10}$$

$$C_R(w \diamond w') \coloneqq R(C_R(w) \diamond C_R(w')) \tag{11}$$

The following lemma is readily proven by induction on the structure of terms.

Lemma 6.8. *If t is a strict sub-word of l, and R is non-overlapping, then $C_R(\varphi_\theta t) = \varphi_{C_R \circ \theta} t$.*

We now have the following theorem.

Theorem 6.9. *Whenever R weakly collapsing, a weak canonizer, and non-overlapping, then C_R is a canonizer.*

Proof. Assume $w \sim_{\mathrm{E}} w'$. We proceed by induction over the proof of equality.

The only non trivial case is $w = \varphi_\theta l$ and $w' = \varphi_\theta r$ for some substitution θ. The case $l = r$ is clear, so we may assume that $l \neq r$. Then we have

$$\begin{aligned} C_R(\varphi_\theta l) &= R(\varphi_{C_R \circ \theta} l) \\ &= \varphi_{C_R \circ \theta} r \\ &= C_R(\varphi_\theta r) \end{aligned}$$

where the first and third equalities follow from the Lemma, observing that r is a strict subterm of l (since R is weakly collapsing and a weak canonizer), and the second from weak canonicity. □

We mention an example to show why this is a useful theorem. Take E to be the equation

$$\mathrm{y} \diamond (\mathrm{x} \diamond (\mathrm{y} \diamond (\mathrm{y} \diamond \mathrm{y}))) \simeq \mathrm{x}.$$

We can take R to be the transformation which sends a term of the form $w \diamond (v \diamond (w \diamond (w \diamond w)))$ to v for any two words v, w, and leaves all other words unchanged. It is then somewhat easy to show that this transformation satisfies the conditions of Theorem 6.9, and so we have a canonizer C_R. This can be used, e.g. to refute the implication of $\mathrm{x} \simeq (\mathrm{x} \diamond \mathrm{x}) \diamond (\mathrm{x} \diamond (\mathrm{x} \diamond \mathrm{x}))$ from E.

As a conclusion for this section, we note that a very general strategy for building canonizers comes from the theory of *rewrite systems*, see e.g. Baader & Nipkow [8]. In that setting one defines rewriting as a transformation on words (or terms), and if this transformation is *terminating* and *confluent* (intuitively, rewrites cannot go on forever, or diverge forever), then one may simply pick the transformation which sends a term to its normal form as a canonizer.

Though we note that the non-overlapping criterion seems very similar to the notion of orthogonality in rewriting, we leave the investigation of the precise relationship of the classical theory with the above technique as future work.

6.4. **Unique factorization.** In general, the free magma $\mathcal{M}_{X,\mathrm{E}}$ for a given equational law E, which we can canonically define as $\mathcal{M}_X/\sim_{\mathrm{E}}$, is hard to describe explicitly; indeed, from the undecidability of implications between equational laws, such a magma cannot be computably described for arbitrary E. Nevertheless, for some laws it is possible to obtain some partial understanding of $\mathcal{M}_{X,\mathrm{E}}$ from a syntactic perspective. For instance, if we can refute the equivalence $w_1' \sim_{\mathrm{E}} w_2'$ by constructing a counterexample magma $\mathcal{M}$ that satisfies E but not $w_1' \simeq w_2'$, then this implies that the representatives $\iota_{X,\mathrm{E}}(w_1'), \iota_{X,\mathrm{E}}(w_2')$ of w_1', w_2' in $\mathcal{M}_{X,\mathrm{E}}$ are distinct.

We illustrate this approach with equations E of the left-absorptive form

$$\mathrm{x} \simeq \mathrm{x} \diamond f(\mathrm{x}, \mathrm{y}, \mathrm{z}) \tag{12}$$

for some word $f(\mathrm{x}, \mathrm{y}, \mathrm{z})$, that are also known to imply the right-idempotent law E378. An illustrative example is the law E854 depicted in Figure 1. Other examples are listed in Figure 9.

Lemma 6.10. *Equation* E854 *is of the form* (12) *and implies* E378.

Proof. Clearly we have (12) with $f(\mathrm{x}, \mathrm{y}, \mathrm{z}) := (\mathrm{y} \diamond \mathrm{z}) \diamond (\mathrm{x} \diamond \mathrm{z})$. From (12) we have in any magma satisfying E854 that

$$x = x \diamond f(x, x, S^2 x) = x \diamond S(x \diamond S^2 x) = x \diamond S(x \diamond f(x, x, x)) = x \diamond Sx.$$

This implies from a further application of (12) that

$$y = y \diamond f(y, x, y) = (y \diamond Sy) \diamond ((x \diamond y) \diamond Sy) = f(x \diamond y, y, Sy)$$

and hence by (12) again

$$(x \diamond y) \diamond y = x \diamond y$$

giving E378. □

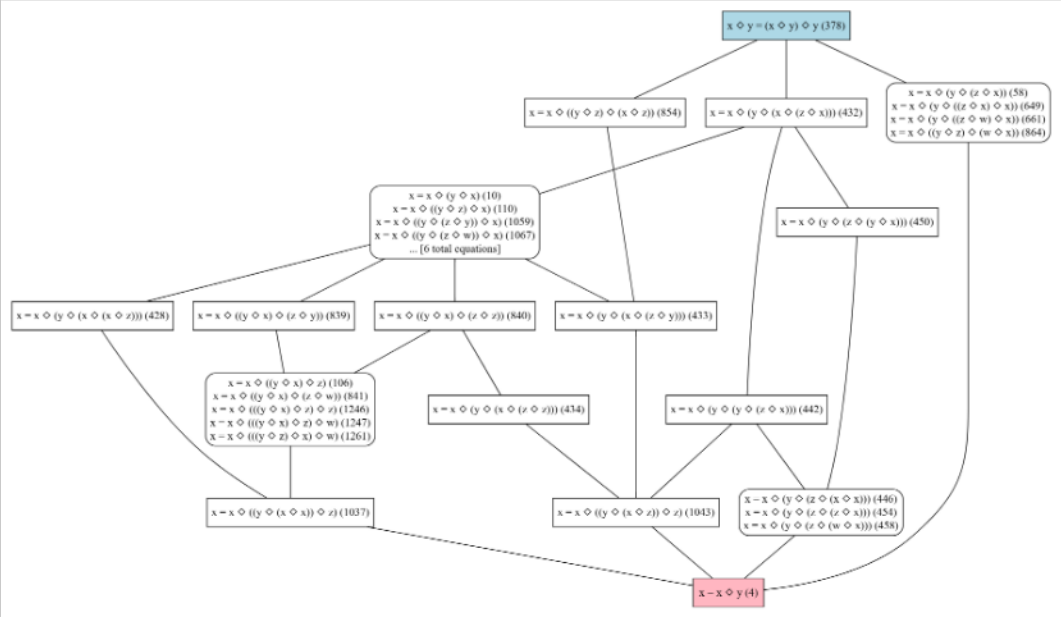


FIGURE 9. Equations similar to E854 that are of the form (12) (possibly involving a fourth indeterminate w) and imply E378. For brevity, 70 equations equivalent to E4 have been omitted.

Let E be a law of the form (12) that implies E378. We define a directed graph $\to_{\mathrm{E}}$ on words in $\mathcal{M}_X$ by declaring $w' \to_{\mathrm{E}} w$ if $w \sim_{\mathrm{E}} w'' \diamond w'$ for some $w' \in M_X$. By E378 (applied to the quotient magma $\mathcal{M}_{X,\mathrm{E}} = \mathcal{M}_X/\sim_{\mathrm{E}}$), this is equivalent to requiring that $w \sim_{\mathrm{E}} w \diamond w'$. In particular, from (12) we have $f(x,y,z) \to x$ for all x, y, z. Furthermore, the relation $\to_{\mathrm{E}}$ factors through $\sim_{\mathrm{E}}$: if $w \sim_{\mathrm{E}} \tilde{w}$ and $w' \sim_{\mathrm{E}} \tilde{w}'$, then $w' \to_{\mathrm{E}} w$ if and only if $\tilde{w}' \to_{\mathrm{E}} \tilde{w}$.

Call a word $w \in M_X$ *irreducible* if it is not of the form $w = w_1 \diamond w_2$ with $w_2 \to_{\mathrm{E}} w_1$. We can partially understand the equivalence relation $\sim_{\mathrm{E}}$ on irreducible words:

Theorem 6.11 (Description of equivalence)**.** *Let* E *be an equation of the form* (12)*. Let* w *be an irreducible word, and let* w' *be a word with* $w \sim_{\mathrm{E}} w'$*.*

(i) If w *is a product* $w = w_1 \diamond w_2$*, then* w' *takes the form*

$$w' = (\dots((w_1' \diamond w_2') \diamond v_1) \diamond \dots) \diamond v_n$$

for some $w_1' \sim_{\mathrm{E}} w_1$*,* $w_2' \sim_{\mathrm{E}} w_2$*, some* $n \geq 0$*, and some words* $v_1, \dots, v_n$ *such that for all* $0 \leq i < n$*,* v_{i+1} *is of the form*

$$v_{i+1} \sim_{\mathrm{E}} f(x_i, y_i, z_i)$$

for some x_i, y_i, z_i *with*

$$x_i \sim_{\mathrm{E}} (\dots((w_1' \diamond w_2') \diamond v_1) \diamond \dots) \diamond v_i.$$

In particular, $v_{i+1} \to_{\mathrm{E}} x_i$*.*

(ii) Similarly, if $w \in X$ *is a generator of* $\mathcal{M}_X$*, then* w' *takes the form*

$$w' = (\dots(w \diamond v_1) \diamond \dots) \diamond v_n$$

for some $n \geq 0$, and some words $v_1, \ldots, v_n$ such that for all $0 \leq i < n$, v_{i+1} is of the form

$$v_{i+1} \sim_{\mathrm{E}} f(x_i, y_i, z_i)$$

for some x_i, y_i, z_i with

$$x_i \sim_{\mathrm{E}} (\ldots(w \diamond v_1) \diamond \ldots) \diamond v_i.$$

In particular, $v_{i+1} \to_{\mathrm{E}} x_i$.

Conversely, any word of the above forms is equivalent to w.

Proof. We just verify claim (i), as claim (ii) is similar. The converse direction is clear from (12) (after quotienting by $\sim_{\mathrm{E}}$), so it suffices to prove the forward claim. By the Birkhoff completeness theorem, it suffices to prove that the class of words described by (i) is preserved by any term rewriting operation, in which a term in the word is replaced by an equivalent term using (12). Clearly the term being rewritten is in w'_1 or w'_2 then the form of the word is preserved, and similarly if the term being rewritten is in one of the v_i. The only remaining case is if we are rewriting a term of the form

$$x_i = (\ldots((w'_1 \diamond w'_2) \diamond v_1) \diamond \ldots) \diamond v_i.$$

If $i > 0$ we can rewrite this term down to x_{i-1}, and this still preserves the required form (decrementing n by one). If $i = 0$ then we cannot perform such a rewriting because of the irreducibility of $w_1 \diamond w_2$ and hence $w'_1 \diamond w'_2$. Finally, we can rewrite x_i to $x_i \diamond v$ where v is of the form

$$v_i = f(x_i, y, z),$$

and after some relabeling we are again of the required form (now incrementing n by one). This covers all possible term rewriting operations, giving the claim. □

Specializing to the case where w, w' are both irreducible, we conclude

Corollary 6.12 (Unique factorization)**.** *Two irreducible words w, w' are equivalent if and only if they are either the same generator of X, or are of the form $w = w_1 \diamond w_2$, $w' = w'_1 \diamond w'_2$ with $w_1 \sim_{\mathrm{E}} w'_1$ and $w_2 \sim_{\mathrm{E}} w'_2$.*

As an application of this corollary, we establish

Proposition 6.13 (E854 $\not\models$ E3316)**.** *Equation* E854 *does not imply* E3316 .

Proof sketch. We work in the free magma $\mathcal{M}_X$ on two generators $X = \{\mathrm{x}, \mathrm{y}\}$. It suffices to show that

$$\mathrm{x} \diamond \mathrm{y} \not\sim_{\mathrm{E854}} \mathrm{x} \diamond (\mathrm{y} \diamond (\mathrm{x} \diamond \mathrm{y})).$$

Suppose this were not the case, then by Corollary 6.12 one of the following statements must hold:

(i) $\mathrm{y} \to_{\mathrm{E854}} \mathrm{x}$.
(ii) $(\mathrm{y} \diamond (\mathrm{x} \diamond \mathrm{y})) \to_{\mathrm{E854}} \mathrm{x}$.
(iii) $\mathrm{y} \diamond (\mathrm{x} \diamond \mathrm{y}) \sim_{\mathrm{E854}} \mathrm{y}$.

If (i) holds, then we have $x \diamond y = x$ must hold in $\mathcal{M}_X/\sim_E$, hence E854 would imply E4. However, it is possible to refute this implication by a finite counterexample.

Similarly, if (iii) held, then E854 would have to imply E10, but this can also be refuted by a finite magma.

Finally, if (ii) held, then the claim

$$x \diamond y \sim_{E854} x \diamond (y \diamond (x \diamond y))$$

to refute simplifies to

$$x \diamond y \sim_{E854} x$$

and we are back to (i), which we already know not to be the case. □

7. Proof automation

In this project we used proof automation in two ways: automated theorem provers (ATPs) and *Lean* tactics. ATPs are generally stand-alone tools that implement a (semi-)decision procedure for a given formal language or related set of languages. For example, *Vampire* [37] is an ATP focused primarily on first-order logic using superposition, which we used extensively in this project. We also made extensive use of *Prover9* and *Mace* [41].

ATPs are complex software that can contain bugs. Instead of trusting ATP output, we used proof certificates, which many ATPs can produce, to reconstruct proofs in *Lean*. The details of proof reconstruction depend on the form of the proof certificate produced by the ATP. We expand on this in Section 7.2.

Tactics in *Lean*, on the other hand, are meta-programs [23] that build proofs. In other words, they essentially take *Lean* code as input and produce *Lean* code as output. In this manner, they look like another keyword in the language, and are tightly integrated by producing proofs directly. Under the hood, their implementation can be arbitrarily complex, from syntactic sugar to full decision procedures. The `duper` tactic [17], for example, implements a superposition calculus similar to *Vampire*'s, but for dependent types — *Lean*'s underlying logical foundation.

In the rest of this section we describe the different proof automation techniques used in this project. We first discuss the different proof methods used: primarily superposition and equational reasoning, we then discuss the integration in *Lean*, and finally we report some basic empirical results from this project.

7.1. Proof techniques.

The two main families of ATPs and tactics we used are based on superposition/saturation and equational reasoning. In this context we also include SMT solvers, which combine specific decision procedures for theories, like congruence closure for equational reasoning, with satisfiability (SAT) solving [18]. Finally, we also used `aesop` [40], which implements a version of tableau search. This was used mainly to help specific constructions in refutations, and is not specific to proving or disproving magma implications in this sense. We describe our use of `aesop` in Section 7.2 below.

Saturation. Most of the ATPs used extensively in this project rely primarily on saturation procedures in the superposition calculus. For example, this is the case for *Vampire* [37].[13] The core idea of these provers is that they take a set of assumptions and a conjecture, expressed in (say) first-order logic. The conjecture is negated and added to the set of assumptions, which are all put into a normal form. The ATP then tries to refute the negation by applying rules of an underlying calculus, until a proof of false (a contradiction) is derived. In this case, the conjecture was (classically) true, and the ATP has found a proof by contradiction, often called a "refutation" or "saturation" proof.

The underlying calculi vary from system to system, but they often have a variant of a resolution clause of the form:

$$\frac{C \vee L \quad D \vee \neg L}{C \vee D}$$

This can be read as $C \vee L$ with $D \vee \neg L$ implies $C \vee D$, where C, D, L are formulas in e.g. first-order logic. Superposition calculi have a variant of this rule that deals with equality directly, and thus are more efficient at reasoning about equality.

In this project we used *Vampire* [37], *Duper* [17] and *Prover9* and *Mace4* [41] which are all based on variants of saturation for proving.

Equational Reasoning. As already discussed in Section 6.3, equational reasoning is a type of reasoning that is based[14] on equational logic and rewriting with congruence [8]. In general, an equational reasoning procedure takes a series of equations and tries to determine whether another equation can be deduced from it. A core tool in equational reasoning are *e-graphs*, a data structure used to represent congruence classes of terms. By themselves, e-graphs provide an efficient means of implementing a decision procedure for congruence closure over ground equations (i.e., equations without variables). Extensions to this procedure, for example by quantifier instantiation via e-matching [19], also allow for a semi-decision procedure for congruence closure over non-ground equations.

SMT solvers like *Z3* [20] use equational reasoning for deciding the theory of equality with uninterpreted functions [38, 19]. On the other hand, equality saturation [67] uses e-graphs by extending congruence closure to a more controlled search, enabling optimization and conditional rewriting. One of the main advantages of using equational reasoning to reason about implications of magma laws is that we get very explicit proofs: a proof that $l \models l'$ is given by a sequence of rewrites that starts at the left-hand side of l' and arrives at the right-hand side through applications of l.

In this project we used *Z3* [20], *Prover9* and *Mace4* [41], a custom ATP *MagmaEgg* for magmas based on egg [67], and the *Lean* `egg` tactic [36, 57], which all work with equational logic. We have also reasoned with manual (custom written) heuristics about simple rewrites.

7.2. **Integration of automation procedures.** While ATPs are very useful for solving theorems in this project, they do not integrate with *Lean* out of the box. ATPs may produce

[13] See also [9] for a gentler exposition.

[14] More precisely, one can formalize this reasoning using Birkhoff's five rules of inference (reflexive, symmetric, transitive, replacement, and substitution); see, e.g., [14].

unsound proofs, or worse, derive incorrect results. Thus, by default, theorems in *Lean* cannot be proven by deferring to the result of an ATP. Instead, the results of an ATP can be used to reconstruct a proof of the form required by *Lean*. Thus, in general, integration of ATPs requires two steps. First, there is the invocation of the ATPs by translating the problem from *Lean* into the languages and logics they use. And second, there is the reconstruction of the ATPs' results as a (persistent) *Lean* proof. These two aspects present different challenges, and require different strategies, depending mostly on the kind of proof strategy the ATP uses.

More generally, we have observed that there are multiple ways of integrating decisions procedures within *Lean*, with different levels of integration.

(1) Using a *Lean* tactic, which calls a decision procedure written in *Lean* (like `aesop` or `duper`).
(2) Using a *Lean* tactic, which calls an existing (external) ATP and reconstructs a proof term from the ATP's result (like `bv_decide` or `egg`).
(3) Using an external script which calls an existing ATP and generates a source file `.lean` which captures the result explicitly.

This project primarily used the least integrated approach, Option 3, as it was fastest to implement and imposed no additional technical requirements on other contributors. The matter of technical requirements caused problems, for example, when integrating the `egg` tactic (Option 2) as it initially expected certain software on the user's machine. Such trade-offs between Option 2 and Option 3 are, however, mutual, as the higher upfront cost of integrating a proof tactic in Option 2 makes the decision procedure easier to use than with Option 3. Additionally, Option 2 can benefit from *Lean*'s meta-programming capabilities when encoding the problem for use with an ATP, and when reconstructing a *Lean* proof from the result.

Proof Reconstruction.

The relative simplicity of the objects used in this project benefit the implementations of proof reconstruction. By focusing on the given problem domain, difficult reconstruction issues, like complex dependent types, could be ignored.

For saturation proofs with *Vampire*, we implemented analogs of the *superpose*, *resolve*, and *subsumption* steps in *Lean*. Proofs can then be reconstructed as sequences of these steps (and additional technicalities) as shown in Figure 10.

For equational proofs from external provers, like *MagmaEgg*, we also used a tailored version of reconstruction. Specifically, the *MagmaEgg* implementation turns *explanations* [53] from *egg* into *Lean* proofs by simple applications of the defining properties of equality as shown in Figure 11.

```
@[equational_result]
theorem Equation999_implies_Equation86 (G : Type*) [Magma G] (h : Equation999 G) : Equation86 G := by
  by_contra nh
  simp only [not_forall] at nh
  obtain (sK0, sK1, sK2, nh) := nh
  have eq9 (X0 X1 X2 X3 : G) : (X1 ◇ ((X2 ◇ X3) ◇ (X1 ◇ X0))) = X0 := mod_symm (h ..)
  have eq10 : sK0 ≠ (sK1 ◇ (sK2 ◇ (sK1 ◇ sK0))) := mod_symm nh
  have eq11 (X3 X4 X5 : G) : (X4 ◇ (X3 ◇ (X4 ◇ X5))) = X5 := superpose eq9 eq9
  have eq21 : sK0 ≠ sK0 := superpose eq11 eq10
  subsumption eq21 rfl
```

FIGURE 10. Example of a proof reconstructed from output of *Vampire*. Note how the proof proceeds by contradiction and uses the `superpose` and `subsumption` steps implemented in *Lean*.

```
private abbrev T := @Eq.trans
private abbrev S := @Eq.symm
private abbrev R := @Eq.refl
private abbrev M := @Magma.op
private abbrev C := @congr_op

@[equational_result]
theorem Equation3973_implies_Equation4023 (G: Type _) [Magma G] (h: Equation3973 G) : Equation4023 G := fun x y z =>
  let v0 := M z x
  let v1 := M z v0
  let v2 := M v1 y
  let v3 := M z v1
  have h4 := R v2
  have h5 := R z
  have h6 := h z x v1
  let v7 := M x (M v1 z)
  T (T (T (T (h x y z) (h (M y v0) z v2)) (C (C h5 (C h4 (T (T (h y v0 v2) (C (C (T (T (T h6 (h v7 v1 v0)) (C (C (R v1)
```

FIGURE 11. Example of a proof reconstructed by *MagmaEgg*. Note the proof only uses reflexivity, symmetry, transitivity, and congruence of equality.

In the case of the `egg` tactic, which also reconstructs proofs from *egg* explanations, the proof could be converted into a more human-readable form by using the `calcify`[15] tactic, as shown in Figure 12

Semi-Automated Counterexample Guidance. Another use of ATPs has been in a semi-automatic fashion, to find counterexamples. The general strategy was to use ATPs to find counterexamples to implications by building magmas iteratively. If we want to build a counterexample to $l \models l'$, we want to construct a magma where l holds but l' does not. In this method, we iteratively strengthen a construction with additional hypotheses, and use the ATP to check whether these hypotheses are not too strong (to imply l') or unsound (to disallow l).

While equational reasoning can also be used in a semi-automatic fashion to prove equations [36], the positive implications in the main implication graph of project were all simple enough that we did not need a semi-automatic approach for them.

[15] `https://github.com/nomeata/lean-calcify`

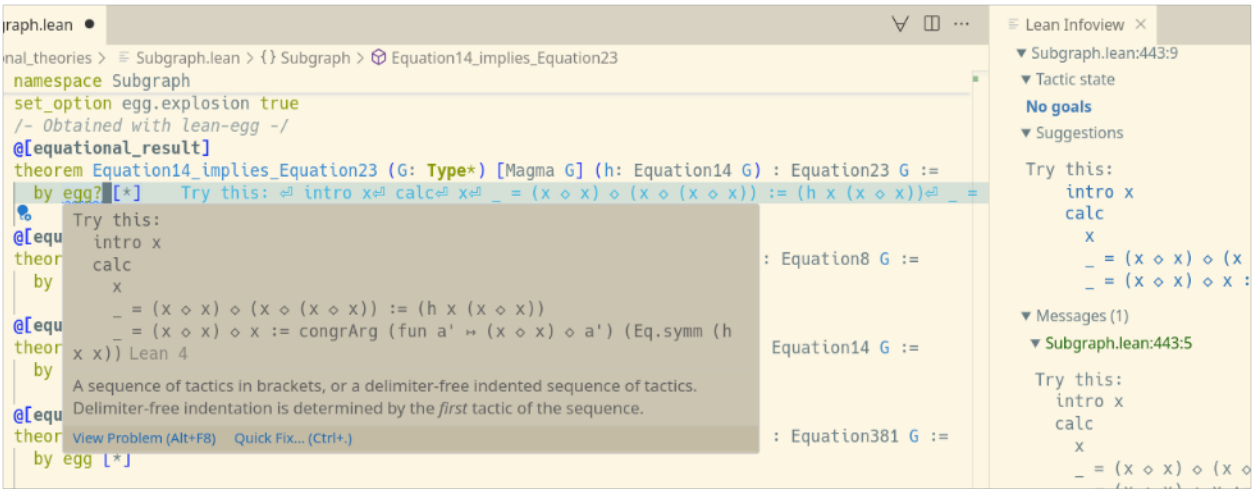


FIGURE 12. Example of the `egg` tactic reconstructing a proof in human-readable form with the help of `calcify` (invoked by the special syntax `egg?`).

7.3. **ATPs usage.** When the project started, contributors had varying degrees of ATPs' knowledge and utilisation skills, with several of us having to start using them from the ground up. With hindsight, several of the project computations could have been approached in better ways, their difficulty diminished due to better ATP expertise. Accordingly, in this section[16] we provide some facts and hints about ATP usage, primarily with the algebraist working with (unsorted) equational theories in mind, based on our experience and available evidence[17] within the ETP. We restrict our attention to *Vampire* and *Prover9-Mace4*, as those have been the main tools used for exploration of implications and anti-implications along the ETP.

7.3.1. *Employing several ATPs.* In general, given a batch of problems on which one wants to work, it is useful to employ several ATPs when they complement each other on the batch–i.e., when they can solve different parts of the batch, so that altogether they can prove more theorems than any of them alone. This is often the case with *Vampire* and *Prover9*: e.g., in a study from 2024 ([3]), it is shown that from a batch of around 770 TPTP problems solved with *Vampire* and *Prover9* together (with some restrictions), around 60% is solved by both, 20% is solved only by *Vampire*, and the remaining 20% is solved only by *Prover9*.[18]

Although older and virtually discontinued, *Prover9* is still very useful with equational logic and algebraic problems. Within the ETP we likewise identified several problems that were easier to solve with *Prover9* than with *Vampire*, and even some solved by *Prover9* but not by *Vampire* (with the configurations we tried). The most salient example is the proof of

[16]Consult the ETP site for a substantially expanded version of these notes.

[17]The timings presented here cannot be taken as benchmarks. Different experiments were executed on different machines, with heterogeneous software and OS environments (Ubuntu, Windows 10...), with varying numbers of parallel processes–both internal and external to the experiment–and, in many cases, under the Linux `nice` command.

[18]In contrast, in said study the problems solved by the E prover happened to be a subset of those proved by *Vampire*, except for one.

E102744082 implying the injectivity of the right multiplication map, used in the Higman-Neumann side project (see Section 10), which was originally found by *Prover9* in several hours with parameters chosen to produce a big search space; an optimized choice of *Prover9* options lowers the runtime to 0.2s. Upon contact with the *Vampire* development team, after several attempts they were able to provide a *Vampire* configuration (inspired by the *Prover9* optimization) which produces a proof in 3s.

In addition, *Mace4* non-SAT algorithm generally makes it faster at finding models than generic SAT-based finite model builders, as those implemented in *Vampire*. For ETP problems in particular, a well-configured *Mace4* is faster than *Vampire*'s finite model builder in both the task of finding a model of a given size and that of exhausting a size with no models. For example, for E677 models, *Mace4* is able to exhaust size 8 in 1.5s, and to find a model of size 9 in 16s, while *Vampire*'s finite model builder is unable to perform any of these tasks in 10 minutes. By contrast, *Vampire*'s `casc_sat` mode can be quicker at finding *some* model of some size for a given problem.

7.3.2. *Basic usage.* When running an ATP or finite model builder on a nontrivial problem, the two key recommendations are: 1) Conduct several runs with different search-parameter settings, and 2) provide enough computational resources for each search: memory[19], number of processing cores[20], number of user instructions executed, and specially, time. Along the ETP, some proofs or models that could not be found in under 1 second with a given configuration, could be obtained in under (say) 8 seconds without altering the configuration; other implications initially required runs lasting several minutes, or even hours. Once a proof is found by some configuration, a short and quick proof can typically be found by tweaking that configuration.

The configuration of the search parameters is a difficult art that has become more sophisticated as ATPs have grown in complexity. To address difficult problems, we strongly recommend acquiring a solid understanding of the available configuration options and their effects, enabling the search space to be tailored as closely as possible to the problem at hand. Currently, options allow control over numerous aspects of each proving stage, including search limits, preprocessing steps, inference rules, formulas ordering and weighting, etc.

7.3.3. *Flow control.* *Vampire* is equipped with a user-friendly and powerful standard mode, the CASC mode (or CASC SAT mode for finite model building) that in many situations avoids the need of configuring specific search options for the given problem. This mode invokes a sequence of strategies (different option configurations) with assigned time limits. The time given to each strategy is important: counterintuitively, giving less time to the CASC mode may end up producing a faster proof: there are proofs than can be found when the time limit is set to less than 5% of the time *Vampire* needs to find a proof without the

[19] If a memory limit is not specified, *Vampire* will try to use as much RAM as possible, but by default *Prover9-Mace4* sets memory limits which are rather low for current standards, so we advise raising them before starting a long computation.

[20] *Vampire* can make use of as many cores as the user specifies. To take advantage of multicore processors with *Prover9-Mace4*, one can run several terminal or GUI instances. Moreover, each GUI window allows to run *Prover9* and *Mace4* simultaneously on the same problem, and they run on different cores.

time limit. At least two factors contribute to this effect: the correct strategy is explored sooner because less time is spent on each strategy, and the behaviour of *Vampire*'s default saturation algorithm, the limited resource strategy (LRS) algorithm.

On the other hand, *Prover9* itself does not have the capability of running several schedules in a row, although it lets the user implement some rudimentary strategies through commands called *actions*. Also, a separate program called *FOF-Prover9* includes a preprocessing step that attempts to reduce the problem to independent subproblems, possibly reducing the overall time significantly.

7.3.4. *Input.* *Prover9-Mace4* formulas can have free variables, which are assumed to be universally quantified at the outermost level of the whole formula, regardless of its parenthesization. This can lead to mistakes for algebraists who are unaware of this issue. For example, suppose we want to formulate that the right multiplication map is injective if and only if it is surjective. If we had both map properties already written separately and without quantifiers, we may copy-paste and join them still without quantifiers (perhaps adding parentheses). But that would quantify all variables at the outermost level, instead of quantifying surjectivity at its desired level, producing an incorrect formula.

Similarly, *Vampire* uses the the TPTP language, in which every variable must be bound by a preceding quantification with adequate scope, with quantifiers having higher precedence than binary connectives. This property may lead to a mistake analogue to the one above, since wrongly placed parentheses can disrupt the quantification scope. Suppose for example that we want to formulate that the right multiplication map is injective if and only if it is surjective. If we had both map properties already written separately with their quantifiers, we may copy-paste and join them by placing each property between parentheses, keeping the quantifiers inside the parentheses. But since quantification has higher precedence than binary connectives, and no opening parenthesis actually restricts the first quantifier scope, in said formula the first quantifier has the whole formula as its scope. We need to place the parentheses after the quantifiers, in order to create the right scope for them.

We have in fact committed these two mistakes over the course of the ETP, thereby giving rise to false proofs of E677 $\models_{\text{fin}}$ E255, our only open implication.

Frequently, it is useful not to run an ATP alone, but to run it in a larger environment permitting multiple calls with different input files, strategies, etc. so that we can provide (automated) semiguidance or (user-guided) interactive guidance. Integrating the ATP into a computer algebra system further allows to leverage the latter's mathematical capabilities, such as preparing input files with operations from sophisticated algebraic structures. For example, in the ETP we have integrated *Prover9-Mace4* with Python and SAGE, and (among several other applications) used SAGE to access GAP's small groups library to search, via *Mace4*, for translation-invariant countermodels with specific groups (see Section 5.3).

An important factor to take into consideration when generating an input file for proving a conjecture is, for both *Vampire* and *Prover9*, that owing to the way the saturation algorithm operates, the original order in which formulas are presented is actually important: it is perfectly possible to have a collection of axioms which produces a proof in some order but

not in another. *Vampire* includes the option `--normalize` to prevent this effect. For *Prover9*, one can use a larger environment to automate the permutations of the input file and make several tries with time restrictions. In contrast, the order of the formulas does not affect *Mace4*'s response.

Moreover, the inclusion of additional formulas redundant with the premises may significantly speed up the proof search, provided that those formulas are not quickly derived by the ATP algorithm (e.g., evaluations mapping different variables to the same one are routinely included).

Finally, the use of demodulators can greatly reduce the search complexity and hence the proof-finding time. *Prover9* `eq_defs` command, when set to `fold`, allows to substitute any specified subexpression by a user-defined symbol, simplifying all further expressions generated by the ATP. In fact, a valid strategy for finding a quick proof of E102744082 implying the injectivity of the right multiplication map consists of setting `fold` and adding the formula `s(x) = x*x` (where $*$ stands for $\diamond$) to the premises (together with a good weight limit, see Section 7.3.5). In *Vampire*, deactivating the `--function_definition_elimination` mode can serve a similar purpose.

7.3.5. *The weight-sos limit strategy.* In *Prover9*, each clause is assigned a weight depending on its length (and other customizable parameters), with newly generated clauses exceeding the `max_weight` limit being discarded. In addition, the size of the list of clauses awaiting processing (the SOS list) is controlled with the `sos_limit` parameter, with most newly generated clauses being discarded once the list is full. Consequently, the size and shape of the search space can be partially controlled through these parameters. Ideally, we would want to use the smallest values of `max_weight` and `sos_limit` that still guarantee a proof to be contained in the search space. Thus smaller values are preferable, provided they are not so low that the search is exhausted without finding a proof.

Since there are more parameters affecting the search, and different proofs may be reachable depending on the configuration, the system's behaviour with respect to `max_weight` and `sos_limit` is not straightforward, with several casuistics arising. When `sos_limit` is fixed, the proof-finding time tends to rise with `max_weight` until reaching a plateau, with the length of this increasing phase extending for higher `sos_limit` values (see Figure 13). For this reason we recommend lowering `sos_limit` from its default value of 20000 to substantially smaller values.

On the other hand, when `max_weight` is fixed, the smallest `sos_limit` values that still yield proofs typically produce a chaotic transient phase with higher proof-finding times, after which a better-defined relation between proof-finding time and `sos_limit` emerges, which tends to follow either a near-plateau pattern (Figure 14a)), or a monotonically increasing trend (Figure 14b)). We recommend avoiding an excessively low `sos_limit`, in order to prevent the onset of the transient phase.

Other relevant characteristics related to the complexity of the resulting proof (such as proof level, length, weight, etc.) may also vary in nontrivial ways depending on the chosen `sos_limit` value (see Figure 15).

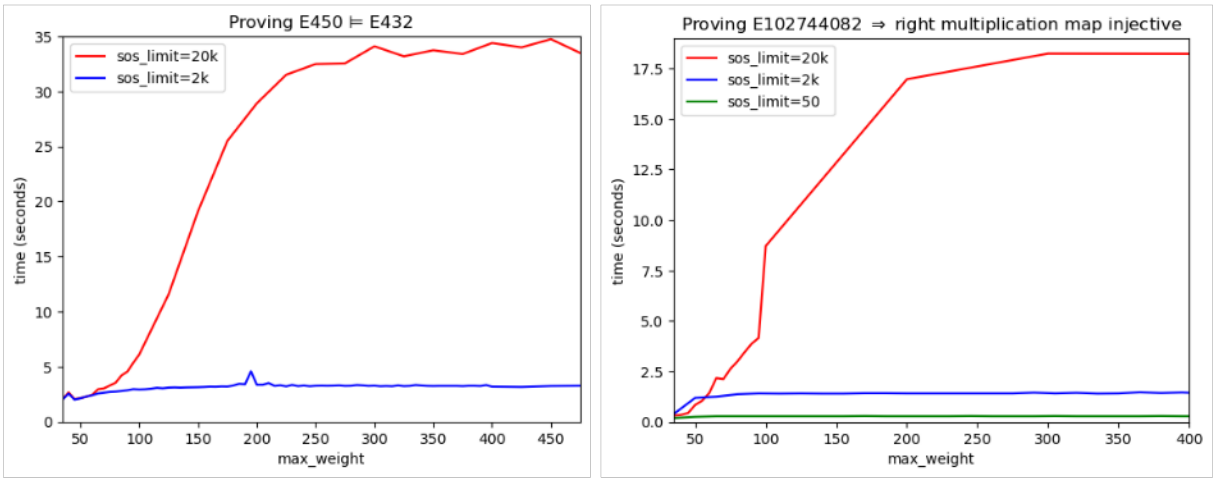


FIGURE 13. Proof-finding time as a function of max_weight, for several values of sos_limit.

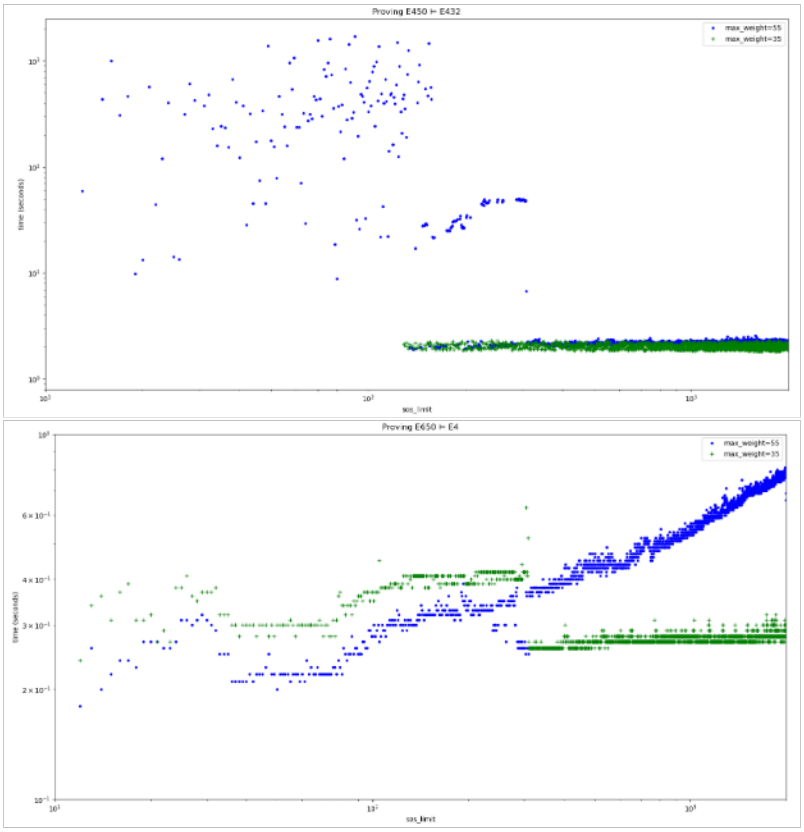


FIGURE 14. Proof-finding time as a function of sos_limit, for different values of max_weight. a) Stabilization as plateau, same for different weights. b) Different behaviours for different weights, with one trend monotonically increasing.

Remarkably, *Prover9* can establish all positive implications between laws of order up to 4 using `max_weight` 55 and `sos_limit` 20000, with `max_weight` 25 being sufficient for almost

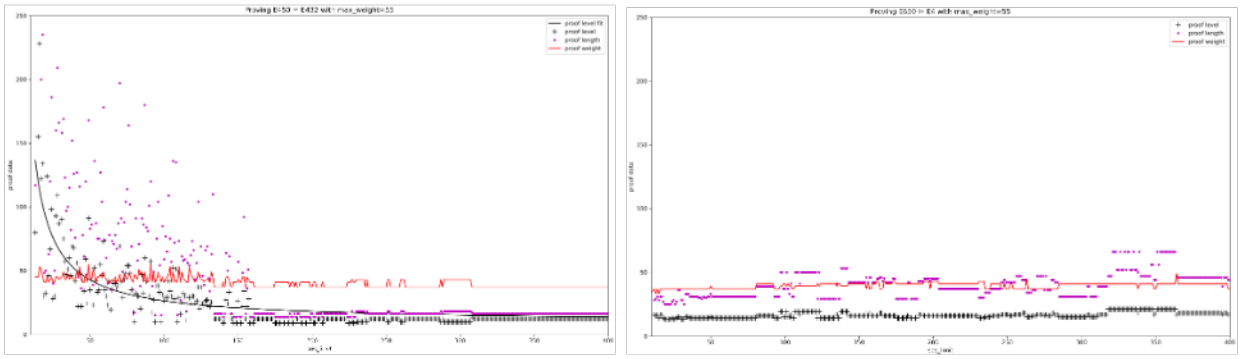


FIGURE 15. Proof-complexity indicators as a function of sos_limit.

every case. With default parameters (`max_weight` 100, `sos_limit` 20000), all consequences of each equation can be proven in at most 1 second per equation, with the exception of laws E450 and E650 (and their duals)[21].

Vampire does not allow the imposition of a user-defined maximum weight, although its default LRS algorithm already implements a dynamic weight limit. According to [31], *Vampire* 5.0 can prove all true implications from the ETP, and can prove 99.97% of them using less than 500 instructions per proof.

7.3.6. *Mace4*. The configuration of *Mace4* likewise has a substantial impact on its running time, with different parameter settings resulting in differences of orders of magnitude. In particular, model-computation time is greatly affected by the `selection_measure` and `skolems_last` parameters. The `selection_order` parameter is also relevant, but our empirical data indicate that `selection_order` 2 is by far the best general choice. It is outperformed by `selection_order` 0 only in a few cases, and even then the improvement in computing time is not that substantial. The effect of setting `skolems_last` can range from slightly detrimental to highly advantageous: as an example, a model of E272260 $\not\models$ E42323216 of size 7 is found in 0.0s with (2,4,Y), and in 4.0s with (2,4,N). Here and in the following we use notation (a,b,S) to mean the configuration `selection_order` a, `selection_measure` b, and `skolems_last` set (S=Y) or clear (S=N).

There is no universally optimal configuration (a,b,E): for each different (potential) implication or theory, configurations may be ranked differently. However, in the ETP context, certain configurations have been preferable due to their consistently better performance. An arguable ordering is (2,3,Y), (2,1,Y), (2,4,Y), (2,3,N), (2,1,N), (2,4,N), (0,1,N), (0,1,Y), with the remaining configurations being largely avoidable. Problems involving several algebraic operations, or presenting some kind of symmetry (see Figure 16 with the example of weak central groupoids), tend to perform better with (2,4,N/Y). Problems may also exhibit considerable sensitivity to changes in formulas that apparently do not alter the problem significantly, for example adding $f^n(x) = x$ with different values of n can result in different optimal configurations.

[21]For further details, see `https://leanprover.zulipchat.com/#narrow/channel/458659-Equational/topic/Which.20implications.20are.20harder.20for.20ATPs/near/547089094`.

Configuration:	(2,4,Y/N)	(0,4,Y/N)	(2,3,Y/N)	(2,2,Y/N)	(2,1,Y/N)
Size:	14 (in 127s)	14 (in 166s)	13 (in 290s)	9 (in 33s)	7 (in 8s)

FIGURE 16. Largest size of E1485 (weak central groupoids) models exhausted by each configuration in under 300s. The skolems_last parameter turns out to be irrelevant for this problem.

In addition, while in the ETP we typically searched for a single model to contradict a given implication, in some cases we sought all models of a certain size for some given theory, either to better understand a collection of base examples in order to extend them, or just to improve our understanding. Accordingly, it should be noted that a configuration that is fast for finding one model of a given size may be slower than others for exhausting the whole size. A notorious example of this is (2,2,Y) for models of E677 of size 9, which finds a single model in 160s, but is unable to exhaust the whole size after 20h (see Figure 17).

Configuration	**One model**	**Exhaust size**
(2,1,N)	16s	190s
(2,1,Y)	16s	210s
(0,1,Y)	20s	149s
(0,1,N)	32s	169s
(2,2,Y)	154s	72000s+
(2,2,N)	180s	43000s+
(2,3,N)	557s	877s
(2,3,Y)	587s	941s
(2,4,Y)	596s	761s
(2,4,N)	620s	730s
(0,3,N)	3746s	4151s
(0,3,Y)	3777s	4254s

FIGURE 17. Time to find one model of E677 of size 9 and to exhaust the same size for different configurations. Some finite-setting formulas were included to speed up the search (see Section 7.3.8). The configurations not listed here were unable to find a model in 4000s.

When undertaking a long-running model search at a large size, it is advisable to determine in advance the potentially optimal configuration(s) by examining smaller sizes. Throughout the ETP, we have employed the following procedure. If time permits, exhaust the previous size with all configurations to make a choice. If that process is prohibitively time-consuming, then set the target number of desired configurations (e.g. for parallel runs), initialize the pool of configurations with all possibilities, and iterate:

(1) Pick the next available smallest size.
(2) Run all configurations on that size under a reasonable time limit and collect their running times, whether for finding a single model, a predetermined number of models, or for exhausting the size.
(3) Based on the smallest collected time, determine a statistically significant time threshold.
(4) Remove from the pool those configurations whose time differs from the best one by more than the threshold (if any). End the loop once the number of configurations contained in the pool is the desired one.

Be aware that this algorithm assumes that once a configuration outperforms another, this advantage carries over to larger sizes. This is not always the case (see Figure 18 for an example). Additionally, the algorithm may be further refined by considering combinations of the configurations with different subsets of formulas.

Size	(2,3,Y) exhaust	(2,4,N) exhaust	(2,3,Y) model	(2,4,N) model
9	0.14s	**0.04s**	0.02s	**0.0s**
10	0.32s	**0.12s**	-	-
11	0.82s	**0.37s**	0.15s	**0.01s**
12	2.17s	**1.2s**	-	-
13	6.18s	**4.35s**	2.74s	**0.91s**
14	15.47s	**14.39s**	-	-
15	**38.82s**	50.87s	-	-
16	**120s**	175s	11.95s	**0.02s**
17	**306s**	660s	-	-
18	**883s**	2382s	-	-
19	**2300s**	3600s+	**846s**	2082s

FIGURE 18. Comparison of the two best configurations for finding certain special 677 models, both for exhausting each size and for finding a single model, with best times in boldface. Note that initially (2,4,N) seems the best configuration, but in the long run, (2,3,Y) outperforms it. The comparison is particularly misleading at size 16 when searching for a single model.

7.3.7. *Search of compatible properties.* When in search of a model, either theoretically of using a model builder, it is useful to be able to identify additional properties to impose on the model, strong enough to simplify the search, yet weak enough to guarantee compatibility with the original axioms. In particular, when seeking a countermodel to implication $E \models E'$, we should avoid properties that, together with E, would imply E′. This search can be greatly aided by an ATP: if we have an educated guess that some property P is compatible with the problem $E \not\models E'$, we can run the ATP on $E \wedge P$ under various strategies to attempt a proof of E′. If, after allowing ample time, no proof is found, this provides heuristic evidence that a countermodel satisfying P exists. Note that even if no proof actually exists, it may well be that there are infinite countermodels but not finite ones.

In the ETP, we successfully applied this approach to several of the outstanding implications. Most notably, we were able to construct an infinite model of $E1323 \not\models E2744$ after heuristically verifying compatibility[22] with a unit element and with closure of the operation over the set of square elements.

7.3.8. *Finite setting.* There are situations in which one wants to work in the finite setting, for example, when proving a finite implication. Indeed, model builders typically search for finite models; therefore, when using them, we can usually assume we are operating in the finite setting. In this context, injective (resp. surjective) maps are bijective, bijections are periodic

[22]Specifically, for each property, we ran *Prover9* for 20 minutes with the default configuration and *Vampire* for 999s in CASC mode.

maps, and so on. In addition, any law of the form $x = f(y) \diamond g(x, y)$ for some maps f, g has left multiplication map $L_{f(y)}$ surjective, hence bijective. Thus by applying the substitution $x \mapsto f(y) \diamond x$ and simplifying $f(y)$ on the left, the law finitely implies $x = g(f(y) \diamond x, y)$.

By this and similar approaches we typically find some multiplication or related map (squaring, etc.) to be injective and surjective, properties that we may add to our initial formulas. In hindsight, adding injectivity to a finite model builder usually increases performance, whereas surjectivity (and other existentially quantified formulas) tend to diminish it. Moreover, *Prover9-Mace4* performs better with operations and equalities than with non-equational formulas. For this reason it is advantageous to convert an injectivity condition from an implication into a new operation. For example, injectivity of the left multiplication map is captured by $x\backslash(x*y) = y$ (where $*$ stands for $\diamond$); this substitution typically yields a threefold speedup. In contrast, this conversion appears to slow down *Vampire* slightly.

Example 7.1. As a case study, let us examine the model search for E1518 $\not\models$ E47 using *Mace4* (with $*$ standing for $\diamond$). It is currently known that the smallest models have size 15. Initially, this search exceeded our ATPs expertise, hence we ended constructing a theoretical model of size 232. If we restrict to the original formulas E1518 and ¬E47, the optimal configuration is (2,3,N), taking 2.5 minutes to exhaust size 10 and already exceeding 7 hours for exhausting size 11. Putting $S(x) := x \diamond x$, since E1518 is $x = (y \diamond y) \diamond (x \diamond (y \diamond x))$ we observe that $L_{S(y)}$ is surjective, hence injective. Accordingly we add `(y*y)*x = (y*y)*z -> x = z` to *Mace4* and then size 11 is exhausted in less than 5 minutes. In addition, from the Equation Explorer we observe that E1518 implies E359, which states $S(x) \diamond x = S(x)$; when added as `(x*x)*x = x*x`, *Mace4* exhausts size 11 in under 1s, with the optimal configuration shifting to (2,1,N). Size 12 is exhausted in 18s. We also note that if $S(x) = S(y)$ then $S(x) \diamond x = S(x) = S(y) = S(y) \diamond y$, yielding $x = y$ by injectivity of $L_{S(x)}$. Therefore S is injective, we can add `x*x = y*y -> x = y`, and exhaust size 12 in 13.5s. Also, in E1518 we can substitute $x \mapsto S(y) \diamond x$ and cancel $S(y)$ on the left to get E320858, with which we exhaust size 12 in under 2s and size 13 in under 1 minute. Moreover, as S is surjective every element is a square, so the injectivity of $L_{S(x)}$ implies that of L_x, and we can add `x*y = x*z -> y = z` to the search. But this actually worsens the performance! After some experimentation, we observe that we should either add the injectivity of $L_{S(x)}$ or that of L_x, but not both. It turns out that for size 13, injectivity of $L_{S(x)}$ performs slightly better, so we retain it. Additionally, the implication formula for this injectivity can be replaced with an operation, by using `x\((x*x)*y) = y`. This further reduces the exhausting time for size 13 to 33s (we also check that injectivity of L_x in operation form still performs slightly worse, and that including both operations is again less effective than using either one alone). Now size 14 is exhausted in 30 minutes, and size 15 is well within reach.

7.3.9. *Discriminators for Isofilter.* *Mace4*'s output may be redundant, in that many of the models found in a search may (and often will) be isomorphic to each other. To solve this problem, *Mace4*'s output can be fed to another tool called *Isofilter*, which returns a representative model of each isoclass. *Isofilter* compares permutations of the models, preceded by the application of some discriminators: by default it only applies the frequency of occurrence of domain element, which is the number of times a domain element appears in the

operation tables[23], but the user can include any number of other discriminators. *Isofilter* "as is" handles models of size up to around 10 with ease; but the combinatorial explosion soon makes computations unfeasible for larger sizes, unless discriminators well suited to the problem are chosen. For example, *Mace4* provides 10 models of size 15 for E1518 $\not\models$ E47, of which 6 belong to different isoclasses. Vanilla *Isofilter* requires 10 days and processes $3 \cdot 10^{13}$ permutations in order to determine these isoclasses, while adding the discriminators `x*x=x. (x*y)*z = x*(y*z). x*y=y. x*y=y*x.` reduces the task to 10^5 permutations, completed in 0.02s.

8. Implications for finite magmas

Many of techniques used to determine the graph of implications E $\models$ E′ can also be used to determine the graph of finite implications E $\models_{\text{fin}}$ E′, with the notable exception of the greedy construction, which appears to be inherently infinitary in nature. On the other hand, when the magma $\mathcal{M}$ is finite, one can prove additional implications by using the fact that any function $f\colon M \to M$ which is surjective, is necessarily injective, or vice versa. We could establish about 200 new implications by providing these two axioms to *Vampire* or to the *Lean* package *Duper*, though in the latter case some human rewriting of the proof was needed to formalize it in the base installation of *Lean*. A small number of additional implications could be resolved by more complicated facts about functions $f, g\colon M \to M$, such as the fact that $f = f \circ f \circ g$ implies $f = f \circ g \circ f$. We refer the reader to the blueprint for examples of such arguments, which were obtained by *ad hoc* methods.

In the end, we were able to establish 820 new implications E $\models_{\text{fin}}$ E′ for which E $\not\models$ E′; and for most other anti-implications E $\not\models$ E′, we were able to strengthen the anti-implication to E $\not\models_{\text{fin}}$ E′. However, there was (up to duality) precisely one finite implication which we could not settle, and leave as an open problem:

Problem 8.1. *Does the law* $x \simeq y \diamond (x \diamond ((y \diamond x) \diamond y))$ (E677) *imply the law* $x \simeq ((x \diamond x) \diamond x) \diamond x$ (E255) *for finite magmas?*

This problem appears to be "immune" to many of our constructions, such as the linear magma construction or the magma cohomology construction; the greedy construction does show that E677 $\not\models$ E255, but the construction is inherently infinite in nature. We tentatively conjecture that E677 $\not\models_{\text{fin}}$ E255; we refer the reader to the blueprint for several partial results in this direction.

9. Spectrum of equational laws

Given an equational law En, one can ask for its spectrum, namely the set of cardinalities of its finite models.[24] The spectrum Spec(En) is a multiplicative subset of $\mathbb{Z}_{>0}$ since the direct

[23]This discriminator is entirely ineffective with models having same number of copies of each element in their tables, such as quasigroups.

[24]The infinite spectrum is uninteresting (assuming the axiom of choice). Let us show that if En is not equivalent to the singleton law E2 then it has models of all infinite cardinalities κ. The free magma on En

product of models is a model. We focus here on the most basic question, that is, which laws (of order up to 4) have spectrum equal to $\mathbb{Z}_{>0}$?

Several extensions will be described in a separate publication: determining the spectrum and not only whether it is full; the spectrum of simple magmas or (sub)directly irreducible magmas; tracking multiplicity, namely counting (or bounding asymptotically) how many finite magmas exist of each size in the spectrum; this is referred to as the *fine spectrum* in [64]. These detailed considerations reveal profound differences in how much an equational law constrains the magma operation, and may help organize the implication graph into different families.

An ATP run shows that 1558 laws of order up to 4 have no model of size 2 and 62 have a model of size 2 but none of size 3. Pushing the search to higher model sizes does not resolve the question for any of the remaining 3074 laws. As we explain next, all of these laws actually have full spectrum.[25]

Our main tool by far to show that a law has full spectrum is to consider the carrying set $\mathbb{Z}/n\mathbb{Z}$, with a linear operation $x \diamond y = ax + by$ with $a, b \in \{-1, 0, 1\}$. If the law holds for some choice of a, b then the law has full spectrum.

- For $(a, b) = (0, 0)$ the operation is the constant operation, which is a model of any law whose sides both have positive order. Equivalently, these laws are consequences of the constant law E41.
- For $(a, b) = (1, 0)$ the operation is a projection, which is a model of any law whose sides start with the same first variable, equivalently the consequences of E4. (The choice $(a, b) = (-1, 0)$ is a model of fewer laws hence is not useful.) Likewise $(a, b) = (0, 1)$ shows that laws whose sides end with the same last variable (consequences of E5) have full spectrum.
- For $(a, b) = (1, -1)$ the operation is abelian group subtraction, characterized by Tarski's axiom E543, which shows that any law implied by E543 has full spectrum. Likewise, backwards subtraction $(a, b) = (-1, 1)$ provides models for E1090 (equivalent to the dual of E543) and its consequences.
- The operation with $(a, b) = (1, 1)$ cannot satisfy a law for all n.
- Finally, the operation $(a, b) = (-1, -1)$ is a model of some more laws, such as the semi-symmetric quasigroup law E14 and totally symmetric quasigroup law E492.

These considerations account for 3068 laws, and there remains three dual pairs of laws to treat. This is done through ad-hoc models: a piecewise linear model for E1682 and its dual, and models whose operation table is mostly constant for the remaining laws E1482, E1523, and their duals.

with κ generators is such a model. Indeed, the generators are distinct in this magma (otherwise En would imply E2) so its cardinal μ is at least κ. Conversely, $\mu \leq \sum_T \kappa^{|T|}$ where the range sums over finite binary trees, and this sum is bounded above by $\aleph_0 \cdot \kappa = \kappa$.

[25]Further investigations show that the lowest-numbered law with models of sizes 2 and 3 but not full spectrum is E80887, namely $x \simeq y \diamond (y \diamond (y \diamond (((y \diamond y) \diamond x) \diamond y)))$, of order 6.

Since a law implied by a full spectrum law has full spectrum itself, the implication graph reduces significantly the number of laws for which it is useful to formalize the full spectrum property. Accounting for duality and implications, we found it sufficient to formalize the proof that 32 laws have no magma of size 2 or none of size 3, and the explicit construction of magmas of all finite sizes for the 7 laws E4, E41, E492, E543, E1482, E1523, and E1682. In conclusion, we prove that 3074 laws (65%) have full spectrum $\mathrm{Spec}(\mathrm{E}n) = \mathbb{Z}_{>0}$ and 1620 (35%) do not (including 1496 laws equivalent to E2). These percentages remain roughly stable at higher orders, with 60% of laws of order up to 9 having full spectrum, as will be reported elsewhere.

10. Higman–Neumann laws

10.1. **Describing groups as magmas.** The ETP is focused exclusively on magmas, which only feature a single (binary) operation. Many mathematical structures traditionally defined using several operations can nevertheless be fully described as magmas with a well-chosen combined operation, from which the whole structure can be reconstructed. The first example is how Boolean algebras defined in terms of three operations $(\wedge, \vee, \neg)$ were equivalently described in 1913 in terms of the Sheffer stroke $x \diamond y := \neg(x \wedge y)$ [58]. Once such a single operation is found, a separate endeavor is to determine which laws it must satisfy to get the desired structure, and, in favorable cases find a single law that encapsulates the whole structure, or even find all equivalent laws of minimum order. The earliest such example is Tarski's description of abelian groups in terms of subtraction $x \diamond y := x + (-y)$, subject to a single axiom $\mathrm{x} \simeq \mathrm{y} \diamond (\mathrm{z} \diamond (\mathrm{x} \diamond (\mathrm{y} \diamond \mathrm{z})))$ (E543), found in 1938 [63]. It then took three decades [28, 59, 54] to sort out the full equivalence class of E543 among laws of order 4. For Boolean algebras, a minimum-order single-law description was only found in [44], nine decades after Sheffer's work.

We plan to report elsewhere on other examples such as modules over Eisenstein integers $\mathbb{Z}[\omega_3]$ or Gaussian integers $\mathbb{Z}[\omega_4]$, with ω_k a primitive kth root of unity, which can be described by the operation $x \diamond y := x + \omega_k y$ subject to the order-6 laws E85914 and E86082, respectively.

Here, we describe the case of groups. The binary operation $*$, unary operation $(\cdot)^{-1}$, and nullary operation e (identity element) can be repackaged into a single division operation $x \diamond y := x * y^{-1}$, from which the original operations are easily reconstructed: for instance $x * y = x \diamond ((y \diamond y) \diamond y)$. A group equipped with division, called a Ward quasigroup, is a magma $(G, \diamond)$ satisfying the unipotence law $\mathrm{x} \diamond \mathrm{x} \simeq \mathrm{y} \diamond \mathrm{y}$ (E40), the right-unit squares law $\mathrm{x} \simeq \mathrm{x} \diamond (\mathrm{y} \diamond \mathrm{y})$ (E11), and a version of the associativity law dubbed the half-group law, $\mathrm{x} \diamond \mathrm{y} \simeq (\mathrm{x} \diamond \mathrm{z}) \diamond (\mathrm{y} \diamond \mathrm{z})$ (E3737), from which group axioms are easily derived. These three laws are equivalent to a single law $\mathrm{E}_{\mathrm{HN}} := \mathrm{E}42323216$ of order 8, found by Higman and Neumann [28],

$$\mathrm{E}_{\mathrm{HN}} \colon \mathrm{x} \simeq \mathrm{y} \diamond \Big((((\mathrm{y} \diamond \mathrm{y}) \diamond \mathrm{x}) \diamond \mathrm{z}) \diamond (((\mathrm{y} \diamond \mathrm{y}) \diamond \mathrm{y}) \diamond \mathrm{z}) \Big).$$

McCune found two more laws equivalent to this one and of the same order [45], E42302852 and E147976245. A natural question is to find all characterizations of Ward quasigroups (groups equipped with division) with minimum order. Throughout our exploration, we used

two criteria: the law must be satisfied by group division, and must fail for magmas that are not Ward quasigroups.

10.2. **Basic constraints.** There are 298 012 537 laws of order up to 8, and running an ATP on all of them is too slow, so one needs efficient ways to filter them beforehand. Let us begin with restrictions on the shape of any law equivalent to $\mathrm{E_{HN}}$. The law must take the form $\mathrm{x} \simeq \ldots$ as otherwise it would be satisfied by the constant operation on any set. The law must be satisfied when evaluated with all variables set to the same element (say, 1) in the Ward quasigroup $\mathbb{Z}$ equipped with subtraction. In particular the law must have even order. This reduces from 3470 shapes of order up to 8, down to just 548 shapes.

Next come some restrictions on the variables. The right-hand side must not start nor end with the variable x, as otherwise the projection operations $x \diamond y := x$, $x \diamond y := y$ respectively would satisfy the law. The law must have at least three variables: otherwise it is satisfied by division in any diassociative loop (such as a Moufang loop), namely a quasigroup with identity element in which every 2-generated submagma is a group. Each variable must appear an even number of times, so that the law holds in Boolean groups (abelian groups of exponent 2). These basic constraints leave 54, 9000, and 1 841 910 candidate laws of orders 4, 6, and 8, respectively, which can be efficiently enumerated since the conditions so far constrain separately the shape and list of variables (refer to Appendix A for the relevant definitions).

Imposing further that the law is satisfied by division in a free non-abelian group (with one generator per variable) reduces these numbers of laws to 0, 59, and 5692 at these same orders. All of the laws coming out of these filters are consequences of $\mathrm{E_{HN}}$; accordingly, one must determine which of these candidates imply said law.

10.3. **Using automated theorem provers.** We repeatedly whittled down the list of candidates by accumulating a collection of finite countermodels, namely magmas that satisfy a candidate law while violating one of the laws E11, E40 and E3737 characterizing Ward quasigroups. Automated searches of small magmas (of size up to 8) with *Mace4* or *Vampire* gave many countermodels (of which 12 are enough). A second source was that of linear models $x \diamond y := ax + by$ on $\mathbb{Z}/n\mathbb{Z}$ with $(a, b) \neq (1, n-1)$: the largest one we used is $x \diamond y := 261x + 33y \bmod 307$ to rule out the candidate law E68185620, $\mathrm{x} \simeq (\mathrm{y} \diamond \mathrm{y}) \diamond (\mathrm{y} \diamond ((\mathrm{x} \diamond (\mathrm{z} \diamond \mathrm{y})) \diamond ((\mathrm{x} \diamond \mathrm{x}) \diamond \mathrm{z})))$. Finally, we introduced some models that are "almost" Ward quasigroups: the 7-element smallest non-associative inverse loop (equipped with division), the 10-element smallest non-associative Steiner loop (commutative loop in which divisions coincide with multiplication), and the 16-element Moufang loop of unit octonions over $\mathbb{Z}$.

These steps eliminated all candidates of order less than 8,[26] and left only 213 laws of order 8 that could be equivalent to $\mathrm{E_{HN}}$. These laws come in 31 families consisting of a "parent"

[26] In particular we recover the inexistence of laws of order 6 characterizing Ward quasigroups, already announced by McCune and Kinyon at `https://www.cs.unm.edu/~mccune/projects/gtsax/#division`.

5-variable law and some specializations with pairs of variables being identified. The lowest-numbered law in this list is McCune's law

$$\mathrm{x} \simeq \mathrm{y} \diamond \Big((((\mathrm{x} \diamond \mathrm{x}) \diamond \mathrm{x}) \diamond \mathrm{z}) \diamond (((\mathrm{x} \diamond \mathrm{x}) \diamond \mathrm{y}) \diamond \mathrm{z}) \Big) \quad \text{(E42302852)},$$

which is in the same family as the Higman–Neumann law. Another common feature is that all 213 candidate laws include at least one subexpression of the form $v \diamond v$ for some variable v.

For 179 candidate laws E, we showed the implication $\mathrm{E} \models \mathrm{E}_{\mathrm{HN}}$ using the ATP *Prover9*. For equations of this order, the ATP computation times increase significantly compared to order-4 laws, with some proofs taking 20 times longer than checking with *Prover9* all 8 178 279 positive implications of the main project. The choices of parameters bounding the ATP search (such as the parameter `max_weight` limiting clause complexity in *Prover9*) were particularly crucial, with different values being optimal in different proofs. Another important speed-up was obtained by seeking proofs of a simple property such as E11, E40, E3737, or injectivity/surjectivity of left or right multiplications, then seeking proofs that the candidate law together with that property implies some other property, and so on, until proving all three laws characterizing Ward quasigroups. The reverse approach also proved useful, namely finding which property would allow the proof to succeed, then seeking a proof of that property from the candidate law. The law E102744082 was a particularly difficult instance: together with injectivity of right multiplications it easily implies the Higman–Neumann law, but the proof that E102744082 does imply injectivity took around 10 CPU hours to obtain in a sweeping search with general parameters; an optimized choice of *Prover9* options trims this time down to 8 minutes. The two characterizations E42302946 and E89176740 of division in groups deserve particular mention for being nicely expressed in terms of the right-cubing map $C(x) := (x \diamond x) \diamond x$:

$$\mathrm{x} \simeq \mathrm{y} \diamond ((C(\mathrm{x}) \diamond \mathrm{z}) \diamond (C(\mathrm{y}) \diamond \mathrm{z})), \qquad \mathrm{x} \simeq C(\mathrm{y}) \diamond ((C(\mathrm{x}) \diamond \mathrm{z}) \diamond (\mathrm{y} \diamond \mathrm{z})).$$

Among the 34 remaining candidates, we showed the finite implication $\mathrm{E} \models_{\mathrm{fin}} \mathrm{E}_{\mathrm{HN}}$ for 21 laws E, which means that the law E characterizes Ward quasigroups among finite magmas. Let us illustrate the proof technique for $\mathrm{x} \simeq (\mathrm{y} \diamond \mathrm{y}) \diamond (\mathrm{y} \diamond ((\mathrm{x} \diamond \mathrm{z}) \diamond (((\mathrm{x} \diamond \mathrm{x}) \diamond \mathrm{y}) \diamond \mathrm{z})))$ (E67953597). In a finite magma, one gets $x = L_{y \diamond y} \circ L_y \circ f_{y,z}(x)$ in terms of the function $f_{y,z} \colon x \mapsto (x \diamond z) \diamond (((x \diamond x) \diamond y) \diamond z)$. Finiteness implies that the composition of several functions can only be a bijection if all of them are bijections, thus left multiplications are bijective. By selecting $y = L^{-1}_{x \diamond x}(w)$ one gets that $(x \diamond z) \diamond (w \diamond z)$ equals the z-independent expression $L_y^{-1} \circ L^{-1}_{y \diamond y}(x)$. Taking $w = x$ yields that the square of $L_x(z)$ is z-independent, hence (by surjectivity of L_x) all squares are equal. A routine ATP run then concludes. While the resulting proofs of finite implications are relatively short and have been successfully ported to *Lean*, our automated search involved thousands of *Vampire* runs. Indeed, rather than the condition that a bijective composition implies bijectivity of its constituents, we had to use the more concrete property that injectivity is equivalent to surjectivity for various collections of specific functions $f \colon M \to M$ such as left or right multiplications, cubing, etc., with a brute-force search over which functions to include in a given run.

The remaining 13 candidates have proven to be quite resistant to both proof and counter-model attacks with a wide range of parameter options, target clauses, additional clauses, and given time (up to 10 hours for each candidate and given experiment). We have shown

with *Prover9* that a model of any of the remaining candidates E that has a right-identity element, or that satisfies the right-unit squares law E11 or unipotence law E40, is a Ward quasigroup. As such, any putative countermodel to the implication $E \models E_{HN}$ must be far from being a Ward quasigroup, in the sense that it must violate these laws.[27] In particular it cannot be a loop; this discards, e.g., the 32-element loop of unit sedenions over $\mathbb{Z}$ as a potential countermodel.

In summary, out of the 298 012 537 laws of order up to 8, we found 179 laws characterizing Ward quasigroups, 21 characterizing them among finite magmas but perhaps not infinite ones, and 13 candidates for which we have neither a counterexample nor a proof even for finite magmas. No efforts have been made yet to find infinite countermodels with the techniques developed in the main project. The results of this section[28] have not been formalized in *Lean* yet. More details and further progress on this side project will be published elsewhere.

11. AI and Machine Learning contributions

As discussed in Section 7, the ETP made extensive use of automated theorem provers in completing the primary goal of determining and then formalizing all the implications between the specified equational laws. In contrast, we were only able to utilize modern large language models (LLMs) in a fairly limited fashion. Such models were useful in writing initial code for graphical user interfaces that we discuss further in Section 12, as well as performing some code autocompletion (using tools such as *Github Copilot*) when formalizing an informal proof in *Lean*. In one instance, *ChatGPT* was used[29] to guess a complete rewriting system for the law $x \diamond ((y \diamond y) \diamond z) \simeq x \diamond y$ (E1659) which could then be formally verified, thus resolving all implications from this equation. However, in most of the difficult implications that resisted automated approaches, we found that LLMs did not provide useful suggestions beyond what the human participants could already propose.

On the other hand, we found that machine learning (ML) methods showed some promise of being able to heuristically predict the truth value of portions of the implication graph; we shall now discuss a convolutional neural network approach.[30]

11.1. **Convolutional neural network model for the implication graph.**

To model the implication graph, we used a convolutional neural network (CNN). For each pair of equations

[27] In fact, we find that countermodels must violate many laws satisfied by group division: E11, E40, E823, E835, E842, E846, E1049, E1637, E1673, E1718, E1835, E1876, E3282, E3323, E3662, E3665, E3677, E3684, E3721, E3729, E3737, E3761, E3823, E3870, E3891, E3943, E4270, E4590 of order up to 4, and many more of higher order, including the associative right division law E912704, $x \diamond (y \diamond (z \diamond w)) \simeq (x \diamond (w \diamond z)) \diamond y$.

[28] `https://github.com/teorth/equational_theories/blob/main/data/Higman-Neumann.json`

[29] `https://chatgpt.com/share/670ce7db-8a44-800d-a5dc-8462c12eca3b`

[30] For some discussion of other ML experiments performed during the project, see `https://leanprover.zulipchat.com/#narrow/channel/458659-Equational/topic/Machine.20learning.2C.20first.20results` for a (vectorized) transformer neural network approach, and `https://leanprover.zulipchat.com/#narrow/channel/458659-Equational/topic/Graph.20ML.3A.20Directed.20link.20prediction.20on.20the.20implication.20graph` for directed link prediction on the implication graph using Graph Neural Network (GNN) autoencoders.

(p, q), the input of the CNN consisted of a character-level tokenization (no vectorization) of the two equations, and the output of the CNN was a yes/no label depending on whether p implies q. The CNN processed the input data using a 5-layer architecture, each layer composed of a 1-D convolution, followed by batch normalization and rectified linear unit activation functions [26]. After the last convolutional layer, a flattening layer and a softmax activation function were used to obtain the output of the network, i.e., the prediction of the implication for the input pair of equations. Note that we trained our models with the 16/10/2024 version of the (infinite) implication graph, for which 362 of the hardest implications (less than 0.002% of the total) were still unknown.

Prior to training the CNN, we divided the data into training (60%), validation (20%), and test (20%) subsets. The CNN was implemented in TensorFlow 2.9.0 [2] and trained on an NVIDIA 3080 Ti GPU with the following configuration: the binary cross-entropy as the loss function to minimize, the Adam method with an initial learning rate of 10^{-3} for the adjustment of network weights, a batch size of 1024 with random shuffling, a learning rate reduction by a factor of 2 after 15 epochs without improvement in the validation loss, and early stopping if no improvement occurred for 40 epochs, with the model with the lowest validation loss being retained as the final CNN. The final CNN was evaluated on the test set, reaching a prediction accuracy of 99.7%, which means that the model misclassified around 66k of the 22 million implications. Since this accuracy was somewhat surprising for such a small and "simple" model, as a control we generated a random label (yes/no) for each pair of equations, then trained the CNN on this data with 60%/20%/20% training/validation/test percentages, resulting in a 49.99% accuracy, as expected from an unbiased model.

It could be the case that the high accuracy of our CNN model was mostly due to it learning the transitivity of the implication relation, as opposed to it discovering patterns in the identities. To clarify this point, it was proposed to train our CNN model either on a random poset or on the equational poset with vertices and labels permuted, and check whether a similar accuracy was achieved. This experiment has not been performed yet.

In any case, since there are around 600k explicit implications from which the rest can be derived by transitivity (2.7% of the total), if the CNN was learning transitivity it should perform well with a very small training dataset. Accordingly, we trained and assessed a CNN model with a 5%/5%/90% training/validation/test proportion, with a resulting 99.6% accuracy on test (the process finishing in under 20 minutes). But the high accuracy is maintained with even smaller training datasets, as evidenced in the following table:

TABLE 2. Prediction accuracy as function of size of training set

Training/validation/test proportion (%)	**Prediction accuracy (%)**
60/20/20	99.7
5/5/90	99.6
1/1/98	99.3
0.5/0.5/99	98.9
0.1/0.1/99.8	92.2

Since these training datasets were significantly smaller than the subset of explicit implications, and were not carefully chosen from the poset extremes but taken randomly, we can conclude that even if the CNN were learning transitivity, that by itself is probably insufficient to explain the high accuracy achieved by the CNN model.

Since sometimes machine learning is announced as a form of data compression, let us now comment on the level of data compression achieved by our CNN model. In the table below we compare the sizes of three different encodings of the implication graph: a) As the simplest approach, we can encode the full implication graph in one file as labelled pairs of equations of the form (p, q, yes/no). b) On the other extreme, we can encode it as a bit table containing neither the explicit equation expressions nor their numbers, but just a 1/0 label for each point with coordinates (p,q), together with a table mapping each number to its corresponding equation expression and a small script to recover the file in a). c) Finally, we can encode it in the complete files of the CNN model produced by TensorFlow 2.9.0. In addition, we can either consider these files in their raw form, or we can highly (but losslessly) compress them to achieve a rough comparison of their actual information content; accordingly, in the table below we also include the sizes of the encoding models when compressed with 7-zip LZMA2 ultra compression with a 1536MB dictionary size and a 273 word size.

TABLE 3. Sizes of different encodings for the implication graph

Encoding model	**Uncompressed size**	**Compressed size**
Labelled pairs of equations	1.5GB	9MB
Bit table	42MB	40KB
CNN (99.7% accuracy)	1.34MB	700KB

As we see, the information in the CNN model is more than 13 times less than in the labelled pairs model, while it is more than 18 times that of the bit table model. We also note that the CNN model in its raw form is already quite incompressible, and more than 30 times smaller than the raw bit table.

Lastly, also take into consideration that our CNN model does not only encode the implication graph up to order 4 (with 0.03% of noise), but a priori may also be able to predict it for higher orders with a significant accuracy. Thus it could be used to guide and speed up the determination of the implication graph up to order 5, by letting ATPs focus first on the CNN's predicted status of the studied implication.

12. User Interfaces

A number of custom web applications were developed as part of the ETP. While many past Lean formalization projects have primarily relied on the Lean blueprint tool to organize tasks and track progress, the large volume of (transitive) implications tracked by the ETP, along with the research-oriented nature of the project, necessitated the development of custom tools to complement the blueprint tool. These web applications also made information more accessible to project participants and other interested parties, including those unfamiliar with

Lean or the custom software developed for the project. The project features four primary interfaces:

(1) The **ETP dashboard**[31] displays the high-level overview of the project: the total number of resolved, conjectured, and unknown implications for the general and finite implication graphs. The dashboard also includes links to other tools, data, and visualizations about the implication graphs.
(2) The **Equation Explorer**[32] is the primary tool to navigate the implication graph. For a given equation, it display its inbound and outbound implications, as well as other members of its equivalence class. The explorer allows navigating either the general or finite implication graphs. The explorer also features custom commentary for a given equation (when available), serving as a repository for information and links. It also links to Graphiti visualizations and an example of its smallest satisfying magma, if one exists. Figure 19 shows an example view of the explorer.
(3) **Graphiti**[33] visualizes the implication graph as a Hasse diagram, where downward edges represent subset relationships, and upward edges represent implications. Equivalence classes are collapsed into single nodes for clarity. Graphiti supports search parameters to visualize specific subsets of the graph. It can also display the entire implication graph, though the complete graph is large and challenging to navigate. Figure 9 is an example of a Graphiti visualization.
(4) The **Finite Magma Explorer**[34] tests which equations a given finite magma satisfies or fails to satisfy. Users input finite magmas as Cayley tables. The tool is aware of the finite implication graph, so if an input magma witnesses an unknown refutation, it notifies the user and provides instructions for contributing the result to the GitHub repository.

The data for these tools is extracted directly from the Lean-formalized proofs in the project's GitHub repository, ensuring it always faithfully reflects the current state of progress. Additionally, the data is automatically updated with each code change using continuous integration (CI), eliminating the need for manual updates.

13. Data management

All the data and formalizations generated by the project are placed in the Github repository[35], while the discussion is almost entirely contained in a dedicated channel on the Lean Zulip[36]. The implication graph can be downloaded from Equation Explorer[37], and can also indicate the individual *Lean* theorems required to establish or refute any given implication, although currently we have only formalized a generating set of implications and refutations in *Lean*, rather than the entirety of the implication graph.

[31] https://teorth.github.io/equational_theories/dashboard/
[32] https://teorth.github.io/equational_theories/implications/
[33] https://teorth.github.io/equational_theories/graphiti/
[34] https://teorth.github.io/equational_theories/fme/
[35] https://github.com/teorth/equational_theories
[36] https://leanprover.zulipchat.com/#narrow/channel/458659-Equational
[37] https://teorth.github.io/equational_theories/implications/

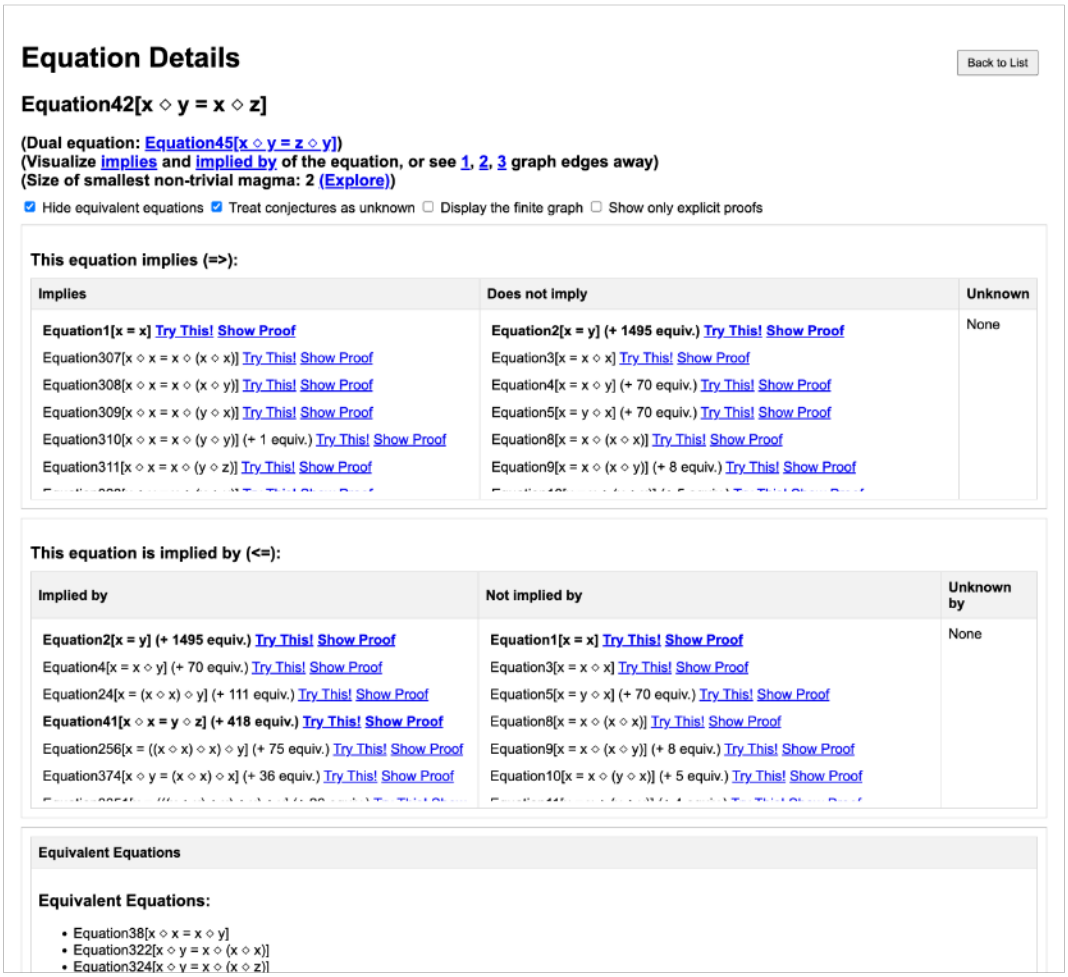


FIGURE 19. An example of the information displayed by the Equation Explorer for a specific equation.

14. Conclusions and future directions

This project successfully demonstrated that large-scale explorations of a space of mathematical statements (in this case, the implications or non-implications between selected equational laws) can be crowdsourced using modern collaboration platforms and proof assistants. No single tool or method was able to study the entirety of this space, and many informal proofs generated contained non-trivial errors; but there were multiple techniques that could treat significant portions of the space, and through a collaborative effort combined with the proof validation provided by *Lean*, one could synthesize these partial and fallible contributions into a complete and validated description of the entire implication graph. While this particular graph was a comparatively simple structure to analyze, we believe that this paradigm could also serve as a model for future projects devoted to exploring more sophisticated large-scale mathematical structures.

Several factors appeared to be helpful in ensuring the success of the project, including the following:

- **A clearly stated primary goal, with an end condition and precise numerical metrics to measure partial completion.** From the outset, there was a specific goal to attain, namely to completely determine and then formalize the implication graph on the original set of 4694 laws. Progress towards that goal could be measured by a number of metrics, such as the number of implications that were conjectured but unformalized, or not conjectured at all. Such metrics allowed participants to see how partial contributions, such as formalizing a certain subset of implications, advanced the project directly towards its primary goal. This is not to say that all activity was devoted solely towards this primary goal, but it did provide a coherent focus to help guide and motivate other secondary activities.
- **A highly modular project**. It was possible for any given coauthor to work on a small subset of implications and focus on a single proof technique, without needing to understand or rely upon other contributions to the project. This allowed the work to be both parallelized and decentralized; many contributors launched their own investigations broadly within the framework of the project, without needing centralized approval or coordination.
- **Low levels of required mathematical and formal prerequisites**. The problems considered in the project did not require advanced mathematical knowledge (beyond a general familiarity with abstract algebra), nor a sophisticated understanding of formal proof assistants. This permitted contributions from a broad spectrum of participants, including those without a graduate mathematical training, as well as mathematicians with no experience in proof formalization. At a technical level, it also meant that formalization of proofs into *Lean* could be done immediately once certain base definitions (such as `Magma`) were constructed. This can be compared for instance with the recent formalization of the Polynomial Freiman–Ruzsa conjecture[38], in which significant effort was expended in the first few days to settle on a suitable framework to formalize the mathematics of Shannon entropy. While some more sophisticated formal structures (such as the syntactic description of laws as pairs of words in a `FreeMagma`) were later introduced in the project, it was relatively straightforward to refactor previously written code to be compatible with these structures as they were incorporated into the project.
- **Variable levels of difficulty, and the amenability to partial progress.** Traditional mathematics projects generally involve a small number of extremely hard problems, with incomplete progress on these problems being difficult to convert into clean partial results. In contrast, the ETP studied a large number of problems with a very broad range of difficulty, so that even if a given proof strategy did not work for a given implication, it could be the case that there was some class of easier implications for which the strategy was successful. This allowed for a means to validate such ideas, and allowed the project to build up a useful and diverse toolbox of proof techniques which became increasingly necessary to handle the final and most difficult implications in the project. It also created a dynamic in which the project initially focused on easy techniques to resolve a significant fraction of the implications, gradually transitioning into more sophisticated methods that focused on a much smaller number of outstanding implications that had proven resistant (or even "immune") to all easier approaches.

[38] `https://github.com/teorth/pfr`

- **Centralized and standardized platforms for discussion, project management, and validation.** While the project was decentralized at the level of the participant, there was a centralized location (a channel[39] on the Lean Zulip) to discuss all aspects of the project, as well as a centralized repository[40] to track all contributions and outstanding issues, a centralized blueprint[41] to describe technical details of proofs to be formalized, and a single formal language (*Lean*) to validate all contributions. A significant portion of the activity in the early stages of the project was devoted to setting out the standards and workflows for handling both the discussion and the contributions, in particular setting up a contributions page[42] and adopting a code of conduct[43]. This gave some structure and predictability to what might otherwise be a chaotic effort.
- **Development of custom visualization tools.** As discussed in Section 12, several tools were developed (in part with AI assistance) to help visualize and navigate the implication graph while it was in a partial stage of development, allowing for participants to independently identify problems to work on, and to validate and use the contributions of other participants even before they were fully formalized. For instance, a participant could propose a finite counterexample to an implication by posting a link to the magma in *Finite Magma Explorer*, allowing for immediate validation of the counterexample, or use *Equation Explorer* or *Graphiti* to observe some interesting phenomenon in the implication graph that other participants could reproduce and study.
- **Applicability of existing software tools.** As described in Section 7, many of the implications in the ETP were amenable to application of "off-the-shelf" automated theorem provers (ATPs); while some trial and error was needed to determine good choices of parameters, these tools could largely be applied directly to the project without extensive customization. (However, the later transcription of ATP output into Lean was sometimes non-trivial.)
- **Receptiveness to new techniques and tools.** Crucially, the methods used to make progress on the project were not specified in advance, and contributions from participants with new ideas, techniques, or software tools that were not initially anticipated were welcomed. For instance, the theory of canonizers (Section 6.3) was not initially known to the first project participants, but was brought to the attention of the project by a later contributor. Conversely, while there were hopes expressed early in the project that modern large language models (LLMs) could automatically generate many of the proofs required, it turned out in practice that other forms of automation, particularly ATPs, were significantly more effective at this task (at least if one restricted to publicly available LLMs), and the project largely moved away from the use of such LLMs (other than to help create the code for the visualization tools).

[39] https://leanprover.zulipchat.com/#narrow/channel/458659-Equational
[40] https://github.com/teorth/equational_theories
[41] https://teorth.github.io/equational_theories/blueprint/
[42] https://github.com/teorth/equational_theories/blob/main/CONTRIBUTING.md
[43] https://github.com/teorth/equational_theories/blob/main/CODE_OF_CONDUCT.md

There are several mathematical and computational questions that could potentially be addressed in future work building upon the outcomes of ETP. Here is a list of some possible such future directions.

(1) Does the law $x \simeq y \diamond (x \diamond ((y \diamond x) \diamond y))$ (E677) imply $x \simeq ((x \diamond x) \diamond x) \diamond x$ (E255) for finite magmas, i.e., E677 $\models_{\text{fin}}$ E255? This is the last remaining implication (up to duality) for finite magmas to be resolved. A number of partial results on this problem may be found at `https://teorth.github.io/equational_theories/blueprint/677-chapter.html`.

(2) The ETP focused on determining relations $E \models E'$ between one law and another. Could the same methods also systematically determine more complex logical relations, such as $E_1 \wedge E_2 \models E_3$, for all laws E_1, E_2, E_3 in a specified set? This includes the question of implications between equational laws in semigroups (associative magmas). One could also consider implications involving magma properties that are not equational laws, such as cancellability or existence of a unit element.

(3) Call an implication $E_1 \models E_2$ "irreducible" if there is no equational law E with $E_1 \models E \models E_2$, other than those laws equivalent to either E_1 or E_2. For instance, E2 $\models$ E4 is irreducible, since E4 implies any law of the form $w \simeq w'$ where the left-most variable of w matches the left-most variable of w'. On the other hand, E4 in conjunction with any law not of that form yields E2. Similar *ad hoc* arguments can produce other irreducible implications, e.g., E2 $\models$ En for $n = 5, 895, 26302$. Could one replicate the ETP to classify all stable implications among the same 4694 equations studied in this project?

(4) For a given finite non-implication $E_1 \not\models_{\text{fin}} E_2$, are there bounds on the proportion of variable assignments for which E_2 holds, similarly to how in a finite group either all elements square to the neutral element, or at most 3/4 of them do?

Some other directions do not concern implications between laws, but may benefit from data generated by the ETP.

(5) Does the law $x \simeq y \diamond (y \diamond (y \diamond (x \diamond (z \diamond y))))$ (E5093) have any infinite models? In [32] it was shown that it has no non-trivial finite models, but the infinite model case was left as an open question. A partial classification of laws of order 5 with infinite models but no finite models is given at `https://teorth.github.io/equational_theories/blueprint/order-5-austin-laws.html`.

(6) A key feature of finite magmas $\mathcal{M}$ is that they are surjunctive, in the sense that any definable map from M to itself that is injective, is also surjective (or vice versa), where "definable" is with respect to the language of magmas. Are there equational theories that admit surjunctive models, but yet do not have any non-trivial finite models?

(7) Are all finite weak central groupoids, namely magmas obeying $x \simeq (y \diamond x) \diamond (x \diamond (z \diamond y))$ (E1485), necessarily of size n^2 or $2n^2$? More generally, what is the spectrum of each law or conjunction of laws, and what are the possible asymptotics for the fine spectrum in terms of model size?

(8) How "stable" is a given law E? For instance, if a finite magma satisfies a law E some proportion $1 - \varepsilon$ of the time, with ε small, can the magma be perturbed into

one that satisfies E exactly? Related to this is the question of whether a law E is "rigid" or "mutable": is it possible to add an element or to make a small number of modifications to a magma satisfying E, in a way that still preserves E? Such properties helped suggest whether certain magma construction techniques, such as modifying a base magma, were likely to be successful.

(9) For each law, can its free magma with one or more generators be described explicitly?

(10) Which laws admit an interesting theory of smooth magmas, analogous to Lie groups?

14.1. **Miscellaneous remarks.** It is possible that the timing in which certain proof methods were introduced into the project created some opportunity costs. For instance, by deploying automated theorem provers at an early stage, we might have settled some implications that had more interesting human-readable proofs that we missed. Similarly, we developed some sophisticated theory for the equation E854, such as Corollary 6.12, that is now superseded by finite counterexamples; but had the finite counterexamples been discovered first, we would not have found the theoretical arguments. It may be productive for future work to revisit some portions of the implication graph and locate alternate proofs and methods.

Acknowledgments

We are grateful to the many additional participants to the Equational Theories Project for their numerous comments and encouragement, with particular thanks to Stanley Burris, Edward van de Meent and David Roberts. Additionally, we note that Shreyas Srinivas is a doctoral student at the Saarbrücken Graduate School for Computer Science.

Appendix A. Numbering system

In this section we record the numbering conventions we use for equational laws.

For this formal definition we use the natural numbers $0, 1, 2, \dots$ to represent and order indeterminate variables; however, in the main text, we use the symbols x, y, z, w, u, v, r, s, t instead (and do not consider any laws with more than eight variables).

To define the ordering we use on equational laws, we first consider the case where there is a single indeterminate $*$. We place a well-ordering on words w, w' with a single indeterminate $*$ by declaring $w > w'$ if one of the following holds:

- w has a larger order than w'.
- $w = w_1 \diamond w_2$ and $w' = w'_1 \diamond w'_2$ have the same order $n \geq 1$ with $w_1 > w'_1$.
- $w = w_1 \diamond w_2$ and $w' = w'_1 \diamond w'_2$ have the same order $n \geq 1$ with $w_1 = w'_1$ and $w_2 > w'_2$.

Thus

$$\begin{aligned} * < * \diamond * < * \diamond (* \diamond *) < (* \diamond *) \diamond * \\ < * \diamond (* \diamond (* \diamond *)) < * \diamond ((* \diamond *) \diamond *) < \dots \end{aligned}$$

We similarly place a well-ordering on equational laws $w_1 \simeq w_2$ with a single indeterminate $*$ by declaring $w_1 \simeq w_2 > w_1' \simeq w_2'$ if one of the following holds: as follows:

- $w_1 \simeq w_2$ has a larger order than $w_1' \simeq w_2'$.
- If $w_1 \simeq w_2$ has the same order as $w_1' \simeq w_2'$, and $w_1 > w_1'$.
- If $w_1 \simeq w_2$ has the same order as $w_1' \simeq w_2'$, $w_1 = w_1'$, and $w_2 > w_2'$.

Thus for instance

$$(* \diamond * \simeq * \diamond (* \diamond *)) < (* \diamond * \simeq (* \diamond *) \diamond *).$$

Finally for equational laws with alphabet x, y, z, w, u, v, r, s, t, define the *shape* of that law to be the law formed by replacing all indeterminates with $*$; for instance, the shape of $x \diamond (y \diamond z) = (x \diamond y) \diamond z$ (E4512), is $* \diamond (* \diamond *) \simeq (* \diamond *) \diamond *$. We then place a well-ordering on $w_1 \simeq w_2$ with indeterminates x, y, z, w, u, v, r, s, t by declaring $w_1 \simeq w_2 > w_1' \simeq w_2'$ if one of the following holds:

- The shape of $w_1 \simeq w_2$ is greater than the shape of $w_1' \simeq w_2'$.
- $w_1 \simeq w_2$ and $w_1' \simeq w_2'$ have the same shape, and the string of variables appearing in $w_1 \simeq w_2$ is lower in the lexicographical ordering (using $x < y < z < w < u < v < r < s < t$) than the corresponding string for $w_1' \simeq w_2'$.

Thus for instance any law of shape $* \diamond * \simeq * \diamond (* \diamond *)$ is lower than any law of shape $* \diamond * \simeq (* \diamond *) \diamond *$. Among the laws of shape $* \diamond * \simeq * \diamond (* \diamond *)$, the lowest is $x \diamond x \simeq x \diamond (x \diamond x)$, which is less than (say) $x \diamond x \simeq y \diamond (y \diamond y)$, which is in turn less than $x \diamond y \simeq x \diamond (x \diamond x)$.

We say that two equational laws are *definitionally equivalent*[44] if one can be obtained from another by some combination of relabeling the variables and applying the symmetric law $w_1 \simeq w_2 \iff w_2 \simeq w_1$. For instance, $(0 \diamond 1) \diamond 2 \simeq 1$ is definitionally equivalent to $0 \simeq (1 \diamond 0) \diamond 2$. We then replace every equational law with their minimal element in their definitional equivalence class, which can be viewed as the *normal form* for that law; for instance, the normal form of $(0 \diamond 1) \diamond 2 \simeq 1$ would be $0 \simeq (1 \diamond 0) \diamond 2$. Finally, we eliminate any law of the form $w \simeq w$ other than $0 \simeq 0$. We then number the remaining equations $E1, E2, \dots$. For instance, E1 is the trivial law $0 \simeq 0$, E2 is the constant law $0 \simeq 1$, E3 is the idempotent law $0 \simeq 0 \diamond 0$, and so forth. Lists and code for generating these equations, or the equation number attached to a given equation, can be found in the ETP repository.

The number of equations in this list of order $n = 0, 1, 2, \dots$ is given by

$$2, 5, 39, 364, 4284, 57882, 888365, \dots$$

[44] This can be distinguished from the weaker notion of *propositional equivalence* (mutual entailment) used in the rest of the paper.

(https://oeis.org/A376640). The number can be computed to be

$$C_{n+1}B_{n+2}/2$$

if n is odd, 2 if $n = 0$, and

$$(C_{n+1}B_{n+2} + C_{n/2}(2D_{n+2} - B_{n+2}))/2 - C_{n/2}B_{n/2+1}$$

if $n > 2$ is even, where C_n, B_n are the Catalan and Bell numbers, and D_n is the number of partitions of $[n]$ up to reflection, which for $n = 0, 1, 2, \dots$ is

$$1, 1, 2, 4, 11, 32, 117, \dots$$

(https://oeis.org/A103293). A proof of this claim can be found in the ETP blueprint. In particular, there are 4694 equations of order at most 4.

Below we record some specific equations appearing in this paper, using the alphabet x, y, z, w in place of 0, 1, 2, 3 for readability.

(E1)	$x \simeq x$	(Trivial law)
(E2)	$x \simeq y$	(Singleton law)
(E3)	$x \simeq x \diamond x$	(Idempotent law)
(E4)	$x \simeq x \diamond y$	(Left-absorptive law)
(E5)	$x \simeq y \diamond x$	(Right-absorptive law)
(E10)	$x \simeq x \diamond (y \diamond x)$	
(E11)	$x \simeq x \diamond (y \diamond y)$	(Right-unit squares law)
(E23)	$x \simeq (x \diamond x) \diamond x$	
(E40)	$x \diamond x \simeq y \diamond y$	(Unipotence law)
(E41)	$x \diamond x \simeq y \diamond z$	(Constant law)
(E43)	$x \diamond y \simeq y \diamond x$	(Commutative law)
(E46)	$x \diamond y \simeq z \diamond w$	(Constant law)
(E47)	$x \simeq x \diamond (x \diamond (x \diamond x))$	
(E73)	$x \simeq y \diamond (y \diamond (x \diamond y))$	
(E151)	$x \simeq (x \diamond x) \diamond (x \diamond x)$	
(E168)	$x \simeq (y \diamond x) \diamond (x \diamond z)$	(Central groupoid law)
(E206)	$x \simeq (x \diamond (x \diamond y)) \diamond y$	
(E255)	$x \simeq ((x \diamond x) \diamond x) \diamond x$	
(E327)	$x \diamond y \simeq x \diamond (y \diamond z)$	(Right reduction law)
(E378)	$x \diamond y \simeq (x \diamond y) \diamond y$	(Right idempotence law)
(E395)	$x \diamond y \simeq (z \diamond x) \diamond y$	(Left reduction law)
(E413)	$x \simeq x \diamond (x \diamond (x \diamond (y \diamond x)))$	
(E543)	$x \simeq y \diamond (z \diamond (x \diamond (y \diamond z)))$	(Tarski's axiom)
(E677)	$x \simeq y \diamond (x \diamond ((y \diamond x) \diamond y))$	(Last open implication)
(E817)	$x \simeq x \diamond ((x \diamond x) \diamond (x \diamond x))$	

	(E854)	$x \simeq x \diamond ((y \diamond z) \diamond (x \diamond z))$	
	(E1045)	$x \simeq x \diamond ((y \diamond (y \diamond x)) \diamond x)$	
	(E1055)	$x \simeq x \diamond ((y \diamond (z \diamond x)) \diamond x)$	
	(E1110)	$x \simeq y \diamond ((y \diamond (x \diamond x)) \diamond y)$	
	(E1117)	$x \simeq y \diamond ((y \diamond (x \diamond z)) \diamond z)$	
	(E1286)	$x \simeq y \diamond (((x \diamond y) \diamond x) \diamond y)$	
	(E1485)	$x \simeq (y \diamond x) \diamond (x \diamond (z \diamond y))$	(Weak central groupoids)
10	(E1571)	$x \simeq (y \diamond z) \diamond (y \diamond (x \diamond z))$	(Boolean groups)
	(E1629)	$x \simeq (x \diamond x) \diamond ((x \diamond x) \diamond x)$	
	(E1648)	$x \simeq (x \diamond y) \diamond ((x \diamond y) \diamond y)$	
	(E1659)	$x \simeq (x \diamond y) \diamond ((y \diamond y) \diamond z)$	
	(E1689)	$x \simeq (y \diamond x) \diamond ((x \diamond z) \diamond z)$	(Equivalent to (E2))
1	(E1729)	$x \simeq (y \diamond y) \diamond ((y \diamond x) \diamond y)$	
	(E2301)	$x \simeq (y \diamond (x \diamond (y \diamond x))) \diamond y$	
	(E2441)	$x \simeq (x \diamond ((x \diamond x) \diamond x)) \diamond x$	
	(E2910)	$x \simeq ((y \diamond (x \diamond y)) \diamond x) \diamond y$	(Dual of (E677))
19	(E3316)	$x \diamond y \simeq x \diamond (y \diamond (x \diamond y))$	
	(E3737)	$x \diamond y \simeq (x \diamond z) \diamond (y \diamond z)$	
	(E3925)	$x \diamond y \simeq (x \diamond (y \diamond x)) \diamond y$	
	(E4315)	$x \diamond (y \diamond x) \simeq x \diamond (y \diamond z)$	
	(E4380)	$x \diamond (x \diamond x) \simeq (x \diamond x) \diamond x$	(Cube-associativity law)
3	(E4482)	$x \diamond (y \diamond y) \simeq (y \diamond y) \diamond x$	(Central squares law)
5	(E4512)	$x \diamond (y \diamond z) \simeq (x \diamond y) \diamond z$	(Associative law)
	(E4531)	$x \diamond (y \diamond z) \simeq (y \diamond z) \diamond x$	(Central products law)
16	(E5093)	$x \simeq y \diamond (y \diamond (y \diamond (x \diamond (z \diamond y))))$	
	(E85914)	$x \simeq y \diamond (z \diamond ((y \diamond x) \diamond (z \diamond (y \diamond z))))$	(Eisenstein modules)
	(E86082)	$x \simeq y \diamond (z \diamond ((y \diamond z) \diamond (w \diamond (x \diamond w)))))$	(Gaussian modules)
1	(E345169)	$x \simeq (y \diamond ((x \diamond y) \diamond y)) \diamond (x \diamond (z \diamond y))$	(Sheffer stroke)

We also list some order-8 characterizations of group division relevant for Section 10.

27	(E42302852)	$x \simeq y \diamond ((((x \diamond x) \diamond x) \diamond z) \diamond (((x \diamond x) \diamond y) \diamond z))$	(McCune law)
	(E42302946)	$x \simeq y \diamond ((((x \diamond x) \diamond x) \diamond z) \diamond (((y \diamond y) \diamond y) \diamond z))$	
	(E42323216)	$x \simeq y \diamond ((((y \diamond y) \diamond x) \diamond z) \diamond (((y \diamond y) \diamond y) \diamond z))$	(Higman–Neumann law)
1	(E67953597)	$x \simeq (y \diamond y) \diamond (y \diamond ((x \diamond z) \diamond (((x \diamond x) \diamond y) \diamond z)))$	(in finite magmas)
7	(E89176740)	$x \simeq ((y \diamond y) \diamond y) \diamond ((((x \diamond x) \diamond x) \diamond z) \diamond (y \diamond z))$	
	(E102744082)	$x \simeq ((y \diamond y) \diamond ((x \diamond z) \diamond x)) \diamond ((z \diamond w) \diamond (x \diamond w))$	
	(E147976245)	$x \simeq ((y \diamond y) \diamond (y \diamond (x \diamond (((y \diamond y) \diamond y) \diamond z)))) \diamond z$	(McCune law)

Appendix B. Author contributions

In a companion document to this paper, the contributions of each author of this paper to the ETP are described, following the standard CRediT categories[45]. Below are the affiliations and grant acknowledgments of individual participants.

- Matthew Bolan: University of Toronto, matthew.bolan@mail.utoronto.ca. Supported by an Ontario Graduate Scholarship.
- Joachim Breitner: Lean FRO, mail@joachim-breitner.de, ORCID 0000-0003-3753-6821
- Jose Brox: IMUVA-Mathematics Research Institute, Universidad de Valladolid, josebrox@uva.es. Supported by a postdoctoral fellowship "Convocatoria 2021" funded by Universidad de Valladolid, and partially supported by grant PID2022-137283NB-C22 funded by MCIN/AEI/10.13039/501100011033 and ERDF "A way of making Europe"
- Nicholas Carlini: Unaffiliated, nicholas@carlini.com
- Mario Carneiro: Chalmers University of Technology & Gothenburg University, Sweden, marioc@chalmers.se
- Floris van Doorn: University of Bonn, vdoorn@math.uni-bonn.de
- Martin Dvorak: Institute of Science and Technology Austria, martin.dvorak@matfyz.cz
- Andrés Goens: TU Darmstadt, andres.goens@tu-darmstadt.de
- Aaron Hill: Unaffiliated, aa1ronham@gmail.com
- Harald Husum: Unaffiliated, harald.husum@gmail.com
- Hernán Ibarra Mejia: Unaffiliated, hernan@ibarramejia.com
- Zoltan A. Kocsis: University of New South Wales, z.kocsis@unsw.edu.au
- Bruno Le Floch: CNRS and Laboratoire de Physique Théorique et Hautes Énergies, Sorbonne Université, blefloch@lpthe.jussieu.fr, ORCID 0000-0002-3965-9705
- Amir Livne Bar-on: Unaffiliated, amir.livne.baron@gmail.com
- Lorenzo Luccioli: University of Bologna, lorenzo.luccioli2@unibo.it
- Douglas McNeil: Unaffiliated, dsm054@gmail.com
- Alex Meiburg: Perimeter Institute for Theoretical Physics / University of Waterloo Institute for Quantum Computing, teqtp@ohaithe.re
- Pietro Monticone: University of Trento, pietro.monticone@studenti.unitn.it
- Pace P. Nielsen: Department of Mathematics, Brigham Young University, pace@math.byu.edu
- Emmanuel Osalotioman Osazuwa: University of Benin, emmanuel.osazuwa@physci.uniben.edu, ORCID 0009-0003-1415-8263
- Giovanni Paolini: University of Bologna, g.paolini@unibo.it
- Marco Petracci: University of Bologna, marco.petracci@studio.unibo.it
- Bernhard Reinke: Aix-Marseille Université, bernhard.reinke@univ-amu.fr

[45] https://credit.niso.org/

- David Renshaw: Institute for Computer-Aided Reasoning in Mathematics, renshaw@icarm.io
- Marcus Rossel: Barkhausen Institut, marcus.rossel@barkhauseninstitut.org
- Cody Roux: Amazon Web Services, cody.roux@gmail.com
- Jérémy Scanvic, Laboratoire de Physique, École Normale Supérieure de Lyon, jeremy.scanvic@ens-lyon.fr
- Shreyas Srinivas: CISPA Helmholtz Center for Information Security, Saarbrücken, Germany. shreyas.srinivas@cispa.de
- Anand Rao Tadipatri: University of Cambridge, art71@cam.ac.uk
- Terence Tao: Department of Mathematics, UCLA, tao@math.ucla.edu. Supported by the James and Carol Collins Chair, the Mathematical Analysis & Application Research Fund, and by NSF grants DMS-2347850, and is particularly grateful to recent donors to the Research Fund, ORCID 0000-0002-0140-7641
- Vlad Tsyrklevich: Unaffiliated, vlad@tsyrklevi.ch, ORCID 0009-0009-9511-5460
- Fernando Vaquerizo-Villar: Biomedical Engineering Group, University of Valladolid, and CIBER de Bioingeniería, Biomateriales y Nanomedicina, Instituto de Salud Carlos III, fernando.vaquerizo@uva.es
- Daniel Weber: Ben-Gurion University of the Negev, weberdan@post.bgu.ac.il
- Fan Zheng: Unaffiliated, fanzheng1729@outlook.com

References

[1] GitHub - PatrickMassot/leanblueprint: plasTeX plugin to build formalization blueprints. — github.com. `https://github.com/PatrickMassot/leanblueprint`. [Accessed 22-09-2025].

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Kristina Aleksandrova, Jan Jakubuv, and Cezary Kaliszyk. Prover9 unleashed: Automated configuration for enhanced proof discovery. In Nikolaj Bjørner, Marijn Heule, and Andrei Voronkov, editors, *Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 100 of *EPiC Series in Computing*, pages 360–369. EasyChair, 2024.

[4] Gilles Audemard and Laurent Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, pages 399–404, 2009.

[5] Gilles Audemard and Laurent Simon. Refining Restarts Strategies for SAT and UNSAT. In Michela Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 118–126. Springer, 2012.

[6] A. K. Austin. A note on models of identities. *Proc. Amer. Math. Soc.*, 16:522–523, 1965.

[7] A. K. Austin. Finite models for laws in two variables. *Proc. Amer. Math. Soc.*, 17:1410–1412, 1966.

[8] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, Cambridge, 1998.

[9] Alexander Bentkamp, Jasmin Blanchette, Visa Nummelin, Sophie Tourret, Petar Vukmirovic, and Uwe Waldmann. Mechanical Mathematicians. *Commun. ACM*, 66(4):80–90, 2023.

[10] George M Bergman. The diamond lemma for ring theory. *Advances in Mathematics*, 29(2):178–218, 1978.

[11] Joel Berman and Stanley Burris. A computer study of 3-element groupoids. In *Logic and algebra*, pages 379–429. Routledge, 2017.

[12] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

[13] Matthew Bolan, Joachim Breitner, Jose Brox, Mario Carneiro, Martin Dvorak, Andres Goens, Aaron Hill, Harald Husum, Hernán Ibarra Mejia, Kocsis, Zoltan, Bruno Le Floch, Lorenzo Luccioli, Douglas McNeil, Alex Meiburg, Pietro Monticone, Giovanni Paolini, Marco Petracci, Bernhard Reinke, David Renshaw, Marcus Rossel, Cody Roux, Jeremy Scanvic, Shreyas Srinivas, Anand Rao Tadipatri, Terence Tao, Vlad Tsyrklevich, Daniel Weber, and Fan Zheng. The Equational Theories Project, September 2024.

[14] Stanley Burris and H. P. Sankappanavar. *A course in universal algebra*, volume 78 of *Grad. Texts Math.* Springer, Cham, 1981.

[15] Kevin Buzzard and Richard Taylor. FLT, 2025.

[16] Mario Carneiro. Lean4Lean: Verifying a Typechecker for Lean, in Lean, 2025.

[17] Joshua Clune, Yicheng Qian, Alexander Bentkamp, and Jeremy Avigad. Duper: A Proof-Producing Superposition Theorem Prover for Dependent Type Theory. In Yves Bertot, Temur Kutsia, and Michael Norrish, editors, *15th International Conference on Interactive Theorem Proving, ITP 2024, September 9-14, 2024, Tbilisi, Georgia*, volume 309 of *LIPIcs*, pages 10:1–10:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[18] Leonardo de Moura and Nikolaj Bjørner. Satisfiability Modulo Theories: An Appetizer. In Marcel Vinícius Medeiros Oliveira and Jim Woodcock, editors, *Formal Methods: Foundations and Applications*, pages 23–36, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[19] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Efficient E-Matching for SMT solvers. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 183–198. Springer, 2007.

[20] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[21] Yael Dillies and Terence Tao. Formalization of the Polynomial Freiman-Ruzsa Conjecture of Marton, November 2023.

[22] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[23] Gabriel Ebner, Sebastian Ullrich, Jared Roesch, Jeremy Avigad, and Leonardo de Moura. A metaprogramming framework for formal verification. *Proc. ACM Program. Lang.*, 1(ICFP):34:1–34:29, 2017.

[24] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[25] Trevor Evans. Products of points—some simple algebras and their identities. *Amer. Math. Monthly*, 74:362–372, 1967.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[27] Timothy Gowers and Michael Nielsen. Massively collaborative mathematics. *Nature*, 461(7266):879–881, October 2009.

[28] Graham Higman and B. H. Neumann. Groups as groupoids with one law. *Publ. Math. Debrecen*, 2:215–221, 1952.

[29] Alexey Ignatiev, Antonio Morgado, and João Marques-Silva. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*, pages 428–437, 2018.

[30] Alexey Ignatiev, Zi Li Tan, and Christos Karamanos. Towards Universally Accessible SAT Technology. In *SAT*, pages 4:1–4:11, 2024.

[31] Mikoláš Janota. Experimental results for vampire on the equational theories project, 2025.

[32] A. Kisielewicz. Austin identities. *Algebra Universalis*, 38(3):324–328, 1997.

[33] Andrzej Kisielewicz. Varieties of algebras with no nontrivial finite members. In *Lattices, semigroups, and universal algebra (Lisbon, 1988)*, pages 129–136. Plenum, New York, 1990.

[34] Donald E. Knuth. Notes on central groupoids. *J. Combinatorial Theory*, 8:376–390, 1970.

[35] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967)*, pages 263–297. Pergamon, Oxford-New York-Toronto, Ont., 1970.

[36] Thomas Koehler, Andrés Goens, Siddharth Bhat, Tobias Grosser, Phil Trinder, and Michel Steuwer. Guided equality saturation. *Proc. ACM Program. Lang.*, 8(POPL):1727–1758, 2024.

[37] Laura Kovács and Andrei Voronkov. First-Order Theorem Proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013.

[38] Daniel Kroening and Ofer Strichman. *Decision Procedures - An Algorithmic Point of View, Second Edition*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.

[39] André Kündgen, Gregor Leander, and Carsten Thomassen. Switchings, extensions, and reductions in central digraphs. *J. Combin. Theory Ser. A*, 118(7):2025–2034, 2011.

[40] Jannis Limperg and Asta Halkjær From. Aesop: White-Box Best-First Proof Search for Lean. In Robbert Krebbers, Dmitriy Traytel, Brigitte Pientka, and Steve Zdancewic, editors, *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023*, pages 253–266. ACM, 2023.

[41] W. McCune. Prover9 and Mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

[42] William McCune. Solution of the Robbins problem. *J. Automat. Reason.*, 19(3):263–276, 1997.

[43] William McCune. Single axioms: with and without computers. In *Computer mathematics (Chiang Mai, 2000)*, volume 8 of *Lecture Notes Ser. Comput.*, pages 83–89. World Sci. Publ., River Edge, NJ, 2000.

[44] William McCune, Robert Veroff, Branden Fitelson, Kenneth Harris, Andrew Feist, and Larry Wos. Short single axioms for Boolean algebra. *J. Automat. Reason.*, 29(1):1–16, 2002.

[45] William W McCune. Single axioms for groups and abelian groups with various operations. *Journal of Automated Reasoning*, 10(1):1–13, 1993.

[46] Ralph McKenzie. On spectra, and the negative solution of the decision problem for identities having a finite nontrivial model. *J. Symbolic Logic*, 40:186–196, 1975.

[47] N. S. Mendelsohn and R. Padmanabhan. Minimal identities for Boolean groups. *J. Algebra*, 34:451–457, 1975.

[48] C. A. Meredith and A. N. Prior. Equational logic. *Notre Dame J. Formal Logic*, 9:212–226, 1968.

[49] Pietro Monticone. LeanProject, 2025.

[50] Leonardo de Moura and Sebastian Ullrich. The Lean 4 Theorem Prover and Programming Language. In *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*, page 625–635, Berlin, Heidelberg, 2021. Springer-Verlag.

[51] V. L. Murskiĭ. The existence in the three-valued logic of a closed class with a finite basis having no finite complete system of identities. *Dokl. Akad. Nauk SSSR*, 163:815–818, 1965.

[52] V. L. Murskiĭ. The existence of a finite basis of identities, and other properties of "almost all" finite algebras. *Problemy Kibernet.*, (30):43–56, 1975.

[53] Robert Nieuwenhuis and Albert Oliveras. Proof-producing congruence closure. In *International Conference on Rewriting Techniques and Applications*, pages 453–468. Springer, 2005.

[54] R Padmanabhan. On single equational-axiom systems for abelian groups. *Journal of the Australian Mathematical Society*, 9(1–2):143–152, 1969.

[55] R. Padmanabhan and R. W. Quackenbush. Equational theories of algebras with distributive congruences. *Proc. Amer. Math. Soc.*, 41:373–377, 1973.

[56] J. D. Phillips and Petr Vojtěchovský. The varieties of loops of Bol-Moufang type. *Algebra Universalis*, 54(3):259–271, 2005.

[57] Marcus Rossel. An Equality Saturation Tactic for Lean. 2024.

[58] Henry Maurice Sheffer. A set of five independent postulates for Boolean algebras, with application to logical constants. *Trans. Amer. Math. Soc.*, 14(4):481–488, 1913.

[59] Marlow Sholander. Postulates for commutative groups. *The American Mathematical Monthly*, 66(2):93–95, 1959.

[60] Th. Skolem. Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theoreme über dichte Mengen. Krist. Vid. Selsk. Skr. I, 1920, Nr. 4, 36 S. (1922)., 1922.

[61] European Mathematical Society. EMS Code of Practice for Mathematical Publication, 19 March 2025.

[62] Terence Tao. A pilot project in universal algebra to explore new ways to collaborate and use machine assistance?, Sep 2024.

[63] Alfred Tarski. Ein Beitrag zur Axiomatik der Abelschen Gruppen. *Fundamenta Mathematicae*, 30(1):253–256, 1938.

[64] Walter Taylor. The fine spectrum of a variety. *Algebra Univers.*, 5:263–303, 1975.

[65] The Coq Development Team. The Coq Proof Assistant 8.20.0, December 2024. `https://doi.org/10.5281/zenodo.14542673`.

[66] The bbchallenge Collaboration, Justin Blanchard, Dan Briggs, Konrad Deka, Nathan Fenner, Yannick Forster, Georgi Georgiev (Skelet), Matthew L. House, Rachel Hunter, Iijil, Maja Kądziołka, Pavel Kropitz, Shawn Ligocki, mxdys, Mateusz Naściszewski, savask, Tristan Stérin, Chris Xu, Jason Yuen, and Théo Zimmermann. Determination of the fifth Busy Beaver value, 2025. In preparation, `https://github.com/bbchallenge/bbchallenge-paper`.

[67] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. egg: Fast and extensible equality saturation. *Proc. ACM Program. Lang.*, 5(POPL):1–29, 2021.

[68] Stephen Wolfram. The physicalization of metamathematics and its implications for the foundations of mathematics, Mar 2022.