PROJECT SUMMARY

Formal Foundations for Informal Mathematics Research

William DeMeo

We present an overview of a recently initiated research project, *Formal Foundations for Informal Mathematics Research* (FFIMR). The full in the project proposal is available in the document FFIMR.pdf.

1. Introduction and Motivation

A significant portion of the professional mathematician's time is typically occupied by tasks other than *Deep Research*. (By Deep Research we mean such activities as discovering truly non-trivial, publishable results, inventing novel proof techniques, or conceiving new research areas or programs.) Indeed, consider how much time we spend on "mental drudgery;" that is, looking for and fixing minor flaws in an argument, handling straightforward special cases of a long proof, or performing clever little derivations which, if we're honest, reduce to mere exercises that a capable graduate student could probably solve. Add to this the time spent checking proofs when collaborating with others or reviewing journal submissions and it's safe to say that the time most of us spend on Deep Research is fairly limited.

It may come as a surprise, then, that computer-aided theorem proving technology, which is capable of managing much of the straight-forward and less interesting aspects of our work, has not found its way into mainstream mathematics. The reasons for this are simple to state, but difficult to resolve. For most mathematicians, the potential benefit of the currently available tools is outweighed by the time and patience required to learn how to put them to effective use. The high upfront cost is due to the fact that most researchers carry out their work in a very *efficient informal dialect* of mathematics—a common dialect that we share with our collaborators, and without which proving and communicating new mathematical results is difficult, if not impossible.

A mathematician's job is to discover new theorems and to present them in a language that is rigorous enough to convince colleagues, yet informal enough to be efficient for developing and communicating new mathematics. Such a language is what we refer to as the *Informal Language* of mathematics research. To verify mathematical arguments by computer, the arguments must be be written in a language that allows machines to interpret and check them. We refer to the practice of writing such proofs as *interactive theorem proving*, and to the programming languages and software systems that check such proofs as *proof assistants* or *interactive theorem provers*. While most mainstream mathematicians have not yet adopted such systems, their use in academia and industry to verify the correctness of hardware, software, and mathematical proofs is spreading rapidly. Indeed, *constructive type theory* and *higher-order logics*, on which most modern proof assistants are based, have played vital roles in the recent formalizations of landmark mathematical results, such as the *Four-Color*

Date: 2019-02-21.

Theorem [11] and the *Feit-Thompson Odd Order Theorem* [12], as well as settling major open problems, such as the *Kepler Conjecture* [14].

Further justification for the use of proof assistants is their potential for improving the referee and validation process. The main issues here are human fallibility and the high opportunity cost of human talent. Indeed, a substantial amount of time and effort of talented individuals is expended on refereeing journal submissions. Despite this the typical review process concludes without supplying any guarantee of validity of the resulting publications [10]. Moreover, the emergence of large computational proofs (e.g., Hales' proof of the Kepler Conjecture) lead to referee assignments that are impossible burdens on individual reviewers [16]. Worse than that, even when such work survives peer review, there remain disputes over correctness, completeness, and whether nontrivial gaps exist. Some recent examples include Atiyah's claim to have proved the *Riemann Hypothesis* and Zhuk's proof of the *CSP dichotomy conjecture* of Feder and Vardi [18].

We can also enlist the help of computers for discovering and checking new mathematics. Consider the space of all proofs comprehensible by the unassisted human mind, and then observe that this is but a small fraction of the collection of all potential mathematical proofs in the universe. The real consequences of this fact are becoming more apparent as mathematical discoveries reach the limits of our ability to confirm and publish them in a timely and cost effective way. Thus, it seems inevitable that proof assistants will have an increasingly important role to play in future mathematics research [15].

Finally, modern proof assistants support automated proof search and this can be used to discover long sequences of first-order deduction steps. Consequently, mathematicians can spend less time carrying out the parts of an argument that are more-or-less obvious, and more time contemplating deeper questions. Indeed, Fields Medalist Tim Gowers expects collaboration with a "semi-intelligent database" to "take a great deal of the drudgery out of research." [13].

2. The Lean Theorem Prover and its Role in the Project

Given our motivation, the choice of proof assistant was easy; we chose the *Lean Programming Language* [1]—developed by Leonardo de Moura (Microsoft Research), Jeremy Avigad (Carnegie Mellon), and their colleagues—for a number of reasons. First, Lean is designed and developed by logicians and computer scientists working together to create a language and syntax that presents things *as they should be*, so that the working in the language feels almost as natural as working in the Informal Language. Thus it is reasonable to expect mathematicians, even those lacking special training in computer science, to adopt the system and use the libraries we develop.

2.1. **Domain specific automation.**

To support the formalization of theorems, we will develop libraries that contain formal statements and proofs of all of the core definitions and theorems of universal algebra. We will automate proof search in the specific domain of universal algebra, and develop tools to help find and formalize theorems emerging from our own mathematics research. We are currently involved in four research projects in universal algebra [4, 6, 7, 8], and an invaluable tool for our work would be a proof assistant with rich libraries for general algebra, equipped with *domain-specific proof tactics for automatically invoking the standard mathematical idioms from our field.* The latter is called *domain-specific automation* (DSA), and one of our primary goals is to demonstrate the utility of DSA for proving new theorems.

3. Foundations of mathematics and computing science

The overriding goal of our project is to re-examine and formalize the foundations of mathematics, with a particular focus on our primary areas of expertise, universal algebra, to do so in a *practical* and *computable* way, and to codify these foundations and advance the state-of-the-art in computer-aided theorem proving technology. The goal will be achieved when the software becomes a natural, if not necessary, part of the working mathematician's toolbox. We envision a future in which we can hardly imagine proving new theorems, completing referee assignments, or communicating and disseminating new mathematics without the support of a proof assistant.

Functional programming languages that support *dependent* and *(co)inductive types* have brought about new opportunities to apply abstract concepts from universal algebra and category theory to the practice of programming, to yield code that is more modular, reusable, and safer, and to express ideas that would be difficult or impossible to express in *imperative* or *procedural programming languages* [3, 17, 5, Chs. 5 & 10]. The *Lean Programming Language* [1] is one example of a functional language that supports dependent and (co)inductive types, and it is the language in which we will carry out our practical foundations program.

Let us summarize in broad terms, using nontechnical language, the main objectives of the project. We intend to

(1) present the core of universal algebra using *practical logical foundations*; in particular, definitions, theorems and proofs shall be constructive and have computational meaning, whenever possible;
(2) develop software that extends the *Lean Mathematical Components Library* [2] to include the output of (1), implementing the core results of our field as *types* and their proofs as *programs* (or *proof objects*) in the *Lean Programming Language* [1, 9].
(3) develop *domain-specific automation (DSA)* tools that make it easier for working mathematicians harness the power of modern proof assistant technology;
(4) teach mathematicians how to use the assets developed in items (1)–(3) to do the following:
    a. translate existing or proposed Informal Language proofs (typeset in LaTeX, say) into Lean so they can be formally verified and tagged with a certificate of correctness;
    b. construct and formally verify proofs of new theorems using Lean;
    c. import (into Lean) software packages and algorithms used by algebraists (e.g. UACalc or GAP) so that these tools can be certified and subsequently invoked when constructing formal proofs of new results.

A primary motivation for this project was our observation that, on the one hand, many important constructs in universal algebra can be defined inductively, and on the other hand, type theory in general, and Lean in particular, offers excellent support for defining inductive types and powerful tactics for proving their properties. These two facts suggest that there is much to gain from implementing universal algebra in an expressive type system that offers powerful tools for proving theorems about inductively defined types.

4. Conclusion

This project presents us with the opportunity to formalize theorems emerging from our research in universal algebra and lattice theory. The PI is involved in three different research

projects in universal algebra. A proof assistant equipped with special libraries and tactics for formalizing universal algebra would be an invaluable tool for these research problems.

Furthermore, a number of results and ideas on which our work depends appear in journals and conference proceedings with many important details missing. Using Lean allows us to not only verify the correctness of theorems and proofs, but also determine the precise foundational assumptions required to confirm the validity of the result. Thus, when doing mathematics with the help of modern proof assistant technology, we are presented with the possibility of automating the process of generalizing results.

The idea is that implementing theorems as types and proofs as "proof objects" would produce the following:

(1) computer verified, and possibly simplified, proofs of known results
(2) better understanding of existing theory and algorithms
(3) new and/or improved theorems and algorithms

There is substantial evidence that Lean is the best platform for formalizing universal algebra. The type theory on which Lean is based provides a foundation for computation that is more powerful than first-order logic and is ideally suited to the specification of the basic objects and most common proof strategies found in of our field.

The full project proposal demonstrates the utility of dependent and inductive types by expressing some fundamental concepts of universal algebra in Lean. Please see the document FFIMR.pdf for more details.

As scientists, we should take seriously any theory that may exposes weaknesses in our assumptions or our research habits. We should not be threatened by these disruptive forces; we must embrace them, deconstruct them, and take from them whatever they can offer our mission of advancing pure mathematics.

Appendix A. Metadata

Title	Formal Foundations for Informal Mathematics Research
Primary Field	03C05 Equational classes, universal algebra
Secondary Fields	03B35 Mechanization of proofs and logical operations 08B05 Equational logic, Malcev conditions 68N18 Functional programming and lambda calculus

Principal Investigator.

William DeMeo (Burnett Meyer Instructor, University of Colorado, Boulder)

Collaborators.

Clifford Bergman (Professor, Iowa State University)

Ralph Freese (Professor, University of Hawaii)

Peter Jipsen (Professor, Chapman University)

Connor Meredith (Graduate Student, University of Colorado, Boulder)

Hyeyoung Shin (Graduate Student, Northeastern University)

Siva Somayyajula (Graduate Student, Carnegie Mellon University)

External Contributors and Advisors.

Jeremy Avigad (Professor, Carnegie Mellon University)

Andrej Bauer (Professor, University of Ljubljana)

Jasmin Blanchette (Assistant Professor, Vrije Universiteit Amsterdam)

Venanzio Capretta (Professor, University of Nottingham)

References

[1] Jeremy Avigad, Mario Carneiro, Leonardo de Moura, Johannes Hölzl, and Sebastian Ullrich. The Lean theorem proving language, 2018. Carnegie Mellon University and Microsoft Research. URL: `https://leanprover.github.io/`.

[2] Jeremy Avigad, Mario Carneiro, and Johannes Hölzl. The lean mathematical components library (mathlib). GitHub.com, 2018. URL: `https://github.com/leanprover/mathlib`.

[3] Andrej Bauer. Algebraic effects and handlers. OPLSS 2018, June 2018. Lecture notes available at: `https://github.com/OPLSS/introduction-to-algebraic-effects-and-handlers`; Recorded lecture available at: `https://youtu.be/atYp386EGo8`.

[4] Clifford Bergman and William DeMeo. Universal algebraic methods for constraint satisfaction problems. *CoRR: arXiv preprint*, abs/1611.02867, 2016. URL: `https://arxiv.org/abs/1611.02867`, `arXiv:1611.02867`.

[5] P. Chiusano and R. Bjarnason. *Functional Programming in Scala*. Manning Publications, 2014. URL: `https://books.google.com/books?id=bmTRlwEACAAJ`.

[6] William DeMeo, Ralph Freese, and Peter Jipsen. Representing finite lattices as congruence lattices of finite algebras, 2018. URL: `https://github.com/UniversalAlgebra/fin-lat-rep`.

[7] William DeMeo, Ralph Freese, and Matthew Valeriote. Polynomial-time tests for difference terms in idempotent varieties. *Intl. J. Algebra and Computation*, 2018. (to appear) preprint available at github.com/UniversalAlgebra/term-conditions/.

[8] William DeMeo, Peter Mayr, and Nik Ruskuc. A new characterization of fiber products of lattices, 2018. URL: `https://github.com/UniversalAlgebra/fg-free-lat`.

[9] William DeMeo and Siva Somayyajula. The lean universal algebra library. GitHub.com, 2018. URL: `https://github.com/UniversalAlgebra/lean-ualib`.

[10] J. Fasel. Erratum on "on the number of generators of ideals in polynomial rings". *Annals of Mathematics*, 186(2):647–648, 2017.

[11] Georges Gonthier. Formal proof—the four-color theorem. *Notices Amer. Math. Soc.*, 55(11):1382–1393, 2008.

[12] Georges Gonthier, Andrea Asperti, Jeremy Avigad, and et al. A machine-checked proof of the odd order theorem. In *Interactive theorem proving*, volume 7998 of *Lecture Notes in Comput. Sci.*, pages 163–179. Springer, Heidelberg, 2013. `doi:10.1007/978-3-642-39634-2_14`.

[13] William Timothy Gowers, Natarajan Shankar, and Patrick Ion. Panel on future directions for big proof, July 2017. Panel discussion; recorded video available online. URL: `https://www.newton.ac.uk/seminar/20170714143015301`.

[14] Thomas Hales, et al. A formal proof of the Kepler conjecture. *Forum Math. Pi*, 5:e2, 29, 2017. `doi:10.1017/fmp.2017.1`.

[15] Michael Harris. Mathematician's of the future? *Slate*, 2015. URL: `https://goo.gl/RjM8yd`.

[16] M. J. H. Heule and O. Kullmann. The science of brute force. *Communications of the ACM*, 60(8):70–79, 2017.

[17] J. Hughes. Why functional programming matters. *Comput. J.*, 32(2):98–107, April 1989. URL: `http://dx.doi.org/10.1093/comjnl/32.2.98`, `doi:10.1093/comjnl/32.2.98`.

[18] Dmitriy Zhuk. The proof of CSP dichotomy conjecture. *CoRR arXiv*, abs/1704.01914, 2017. URL: `http://arxiv.org/abs/1704.01914`, `arXiv:1704.01914`.