William DeMeo

Application for Employment

Contents

William DeMeo, Ph.D.
Burnett Meyer Instructor
Department of Mathematics
University of Colorado
Boulder, CO 80302
williamdemeo@gmail.com
williamdemeo.org

December 30, 2018

Dear Members of the Search Committee,

I am seeking an assistant professorship on the faculty of mathematics and computer science. I hope you will give my application serious consideration and find that I am a great fit for your academic community.

My teaching experience is extensive, and I love encouraging and mentoring budding mathematicians and computer scientists. I've held academic appointments at four institutions and in each location I started active seminars or research groups.

- At the University of Colorado, I co-organized (with K. Kearnes and P. Lessard) a reading course on *Algebraic Theories*, and founded the *CU Lean User's Group* to help graduate students learn to use the LEAN proof assistant to support their research.
- At the University of Hawaii I co-organized (with R. Freese) the *Universal Algebra Seminar*, and founded the *Mathematical Foundations of Computing Circle* which gathered weekly for lectures on the lambda calculus, type theory, and functional programming.
- At Iowa State University I co-organized (with C. Bergman) the *Universal Algebra Seminar*, and founded the *TypeFunc* research group to expose undergraduates to accessible research problems. Three of my students solved some of these problems, and one was awarded a research grant to support this work.
- At the University of South Carolina, I co-organized (with G. McNulty) the *Algebra and Logic Seminar*, and mentored a talented undergraduate student in the honors college who won a prestigious *Magellan Scholar Grant* for this work.

I am an expert in the LEAN proof assistant and the SCALA functional programming language. I'm also comfortable with AGDA and Coq. My current research program involves the development and implementation of the foundations of mathematics in LEAN (as explained in my research description), and I have successfully introduced LEAN in the classroom (as described in my teaching statement).

Thank you for taking the time to read my letter and consider my application for employment.

Sincerely,

William DeMeo

William DeMeo

Curriculum Vitæ

Contact Information

1805 Spruce St, Apt E
Boulder, CO 80302
USA

tel: 212-308-4134
url: williamdemeo.org
email: williamdemeo@gmail.com

Research Interests

Universal algebra, lattice theory, logic, computational complexity, category theory, type theory, programming languages.

Applications: Computer-aided theorem proving with Lean and Coq, Big Data analysis with Scala/Spark, Blockchain technologies and functional programming.

Education

Doctor of Philosophy in Mathematics, University of Hawai'i at Mānoa — 2012
Thesis: *Congruence lattices of finite algebras*. Advisor: Ralph Freese

Master of Science in Mathematics, New York University Courant Institute — 1998
Thesis: *Approximating eigenvalues of large stochastic matrices*. Advisor: Jonathan Goodman

Bachelor of Arts in Economics, University of Virginia — 1994

Academic Appointments

Burnett Meyer Instructor, University of Colorado, Boulder — 2017–2019
Visiting Assistant Professor, University of Hawaii, Honolulu — 2016–2017
Post-doctoral Associate, Iowa State University, Ames — 2014–2016
Visiting Assistant Professor, University of South Carolina, Columbia — 2012–2014

Professional Experience

Senior Research Scientist, Textron Systems Corporation — 2001–2006
Role: image processing and dsp research; algorithm design and complexity analysis

Grants & Awards

NSF Research Grant (grant no. 1500218) — 2015–2018
Project Title: *Algebras and algorithms, structure and complexity theory*
Role: postdoctoral fellow on a team with 6 senior scientists and 3 postdocs
Description: 3-yr collaborative research on algebraic approaches to constraint satisfaction problems

Magellan Scholar Grant — 2013–2014
Project Title: *What does a nonabelian group sound like?*
Role: faculty mentor for undergraduate research
Description: available at soundmath.github.io/GroupSound/GroupSound

ARCS Sarah Ann Martin Award for Outstanding Research in Mathematics — 2011

Best Paper Award, International Symposium on Musical Acoustics — 2004

Publications

Journal Articles

1. Universal algebraic methods for constraint satisfaction problems, with Clifford Bergman; to appear in *Logical Methods in Computer Science (LMCS);* preprint link: arXiv [cs.LO] 1611.02867
2. Polynomial-time tests for difference terms in idempotent varieties, with Freese and Valeriote; to appear in *International Journal of Algebra & Computation (IJAC);* preprint link: diffTerm-ijac-r1-draft-20180905.pdf
3. Isotopic algebras with nonisomorphic congruence lattices (sole author) *Algebra Universalis* **72**:295–298, 2014; preprint link: Isotopy-AU-2014.pdf
4. Expansions of finite algebras and their congruence lattices (sole author) *Algebra Universalis* **69**:257–278, 2013; preprint link: DeMeo-Expansions-AU-2013.pdf

Refereed Conference Proceedings

5. *Proceedings of Algebras and Lattices in Hawaii 2018*, editor with K. Adaricheva, J. Hyndman.
6. *Topics in nonabelian harmonic analysis and* DSP *applications*, Proceedings of the International Symposium on Musical Acoustics, Nara, Japan 2004 (best paper award).
7. *Characterizing musical signals with Wigner-Ville interferences*, Proceedings of the International Computer Music Conference, Göteborg, Sweden 2002.
8. *Approximating eigenvalues of large stochastic matrices*, Proceedings of the 8th Copper Mt. Conference on Iterative Methods, Colorado, USA 1998.

Papers in Progress

A new characterization of fiber products of lattices, with P. Mayr and N. Ruskuc.
Draft available at github.com/UniversalAlgebra/fg-fin-lat

Representing finite lattices as congruence lattices of finite algebras, with R. Freese and P. Jipsen.
Draft available at github.com/UniversalAlgebra/fin-lat-rep

Books in Progress

Algebras, Categories and Types: with computer-aided proofs, with Hyeyoung Shin.

A Concise Course in Category Theory, with Charlotte Aten and Venanzio Capretta.

Problems in Real and Complex Analysis.

Problems in Groups and Rings.

Summer Schools Attended

Oregon Programming Languages Summer School Topics: parallelism and concurrency	University of Oregon July 3–21, 2018
Computer-aided Mathematical Proof Topics: bringing proof technology into mainstream mathematics	Cambridge University July 10–14, 2017
Oregon Programming Languages Summer School Topics: dependent, gradual, substructural type systems	University of Oregon June 26–July 8, 2017
Midlands Graduate School in the Foundations of Computing Science Topics: type theory, denotational semantics, category theory	University of Birmingham April 11–15, 2016
Oregon Programming Languages Summer School Topics: type theory, logic, semantics, verification	University of Oregon June 16–28, 2014
Midlands Graduate School in the Foundations of Computing Science Topics: simply typed lambda calculus, domain theory, category theory	University of Nottingham April 22–26, 2014
LMS/EPSRC Short Course in Computational Group Theory Topics: permutation & finitely presented groups, constructive recognition	University of St. Andrews Jul 29–Aug 2, 2013
NATO ASI on Computational Noncommutative Algebra	Il Ciocco, Italy, 2003

Data Science Credentials

Big Data Analysis with Scala and Spark 4-week Coursera course; grade: 93.4%	École Polytechnique Fédérale de Lausanne Verified Certificate earned 24 Nov 2017
Functional Programming Principles in Scala 6-week Coursera course; grade: 100%	École Polytechnique Fédérale de Lausanne Verified Certificate earned 17 Nov 2016
Functional Program Design in Scala 4-week Coursera course; grade: 100%	École Polytechnique Fédérale de Lausanne Verified Certificate earned 6 Aug 2016
Parallel Programming in Scala 4-week Coursera course; grade: 100%	École Polytechnique Fédérale de Lausanne Verified Certificate earned 27 Jun 2016
Startup Engineering 12-week Coursera course; grade: 99.3%	Stanford University Verified Certificate earned 23 Sep 2013

Synergistic Activities

Organizer: *Algebras and Lattices in Hawai'i Conf. to honor Freese, Lampe & Nation*	Honolulu 2018
Organizer: *Workshop on Computational Universal Algebra*	Louisville 2013
Editor for *Algebra Universalis* mathematics journal	2018–present
Referee for *Algebra Universalis*, *Order*, and *J. Logic & Analysis*	2012–present
Founder/editor: universalalgebra.org	2013–present

University of Colorado, Boulder
Ph.D. Preliminary Exam Committee for Jordan DuBeau, Ali Latfi, Athena Sparks, Michael Wheeler
Ph.D. Thesis Defense Committee for Jeffrey Shriner
Honors Thesis Defense Committee for Zetong Xue

Iowa State University
REU mentor for Charlotte Aten (mathematics major, University of Rochester)
Honors thesis advisor for Joshua Thompson (mathematics major, honors program)
Putnam Exam mentor at weekly exam practice meetings
Undergraduate Tea cohost of weekly undergraduate student gatherings
Iowa 4-H Youth Conference volunteer mentor (link)

University of South Carolina
Honors thesis mentor for Matthew Corley (computer science major, honors program)
South Carolina High School Math Contest exam design committee
Faculty mentor for Pi Mu Epsilon (math honors society)

Teaching Experience

University of Colorado, Boulder (as Burnett Meyer Instructor)

Math 2001: Discrete Mathematics	Spring 2019
Math 2001: Discrete Mathematics	Fall 2018
Math 3140: Abstract Algebra	Fall 2018
Math 6000: Model Theory (graduate course)	Spring 2018
Math 2130: Linear Algebra	Spring 2018
Math 2130: Linear Algebra	Fall 2017

University of Hawaii (as Visiting Assistant Professor)

Math 215: Applied Calculus	Spring 2017
Math 480: Senior Seminar	Spring 2017
Math 244: Calculus IV	Fall 2016
Math 321: Introduction to Advanced Math	Fall 2016

Iowa State University (as Postdoctoral Associate)

Math 317: Linear Algebra	Spring 2016
Math 317: Linear Algebra	Fall 2015
Math 160: Survey of Calculus	Fall 2015
Math 207: Elementary Linear Algebra	Spring 2015
Math 165: Calculus I	Spring 2015
Math 301: Abstract Algebra	Fall 2014
Math 165: Calculus I	Fall 2014

University of South Carolina (as Visiting Assistant Professor)

Math 700: Linear Algebra (graduate course)	Spring 2014
Math 141: Calculus I	Spring 2014
Math 374: Discrete Structures	Fall 2013
Math 122: Calculus for Business and Social Sciences	Fall 2013
Math 374: Discrete Structures	Spring 2013
Math 122: Calculus for Business and Social Sciences	Spring 2013
Math 241: Vector Calculus	Fall 2012
Math 122: Calculus for Business and Social Sciences	Fall 2012

University of Hawaii (as Graduate Student Instructor)

Math 371: Probability Theory	Summer 2011
Math 215: Applied Calculus I	Summer 2009
Math 100: Mathematical Reasoning	Summer 2010

Talks

Computing Difference Term Operations in Polynomial Time — Denver, CO 2018
BLAST Conference, University of Denver

Why Universal Algebra Needs Inductive, Dependent Types — Eugene, OR 2018
Oregon Programming Languages Summer School

A Tutorial Introduction to the Lean Prover — Boulder, CO 2018
University of Colorado Logic Seminar

The Lambda Calculus and Dependent Type Theory — Boulder, CO 2018
University of Colorado Logic Seminar

Representing Finite Lattices as Congruence Lattices (slides) — Fort Collins, CO 2017
Colorado State University Algebra Seminar

Algebraic Approach to Complexity of Constraint Satisfaction Problems (slides) — Honolulu, HI 2016
University of Hawaii Logic and Analysis Seminar

Universal Algebraic Methods for Constraint Satisfaction Problems — Denver, CO 2016
AMS Fall Western Sectional Meeting: Special Session in Algebraic Logic

The Rectangularity Theorem of Barto and Kozik (slides) — Boulder, CO 2016
Algebras and Algorithms: Structure and Complexity Theory

Constraint Satisfaction Problems and Universal Algebra (slides) — Birmingham, GBR 2016
Midlands Graduate School in the Foundation of Computing Science

Permutability in Diamonds — Ames, IA 2016
Iowa State Algebra and Combinatorics Seminar

Which Commutative Idempotent Binars are Tractable? (slides) — Nashville, TN 2015
Vanderbilt Shanks workshop: Open Problems in Universal Algebra

Some Small Finite Algebras Yielding Tractable CSP Templates — Ames, IA 2015
Iowa State Algebra and Combinatorics Seminar

Algebraic CSP and Tractability of Commutative Idempotent Binars (slides) — Denton, TX 2015
BLAST Conference, University of North Texas

Isotopic Algebras — Ames, IA 2015
Iowa State Algebra and Combinatorics Seminar

What Does a Nonabelian Group Sound Like? (slides) — Baltimore, MD 2014
MAA Special Session: At the Intersection of Mathematics and the Arts

Interval Enforceable Properties of Finite Groups (slides) — Louisville, KY 2013
AMS Special Session on Finite Universal Algebra

Tutorial: UACalc at the command line and in the cloud — Louisville, KY 2013
Workshop on Computational Universal Algebra

Approximating Eigenvalues of Large Stochastic Matrices — Columbia, SC 2013
University of South Carolina Combinatorics Seminar

Congruence Lattices of Finite Algebras (plenary lecture) (slides) — Orange, CA 2013
BLAST Conference, Chapman University

Transposition Principles for Subgroups and Equivalence Relations (slides) — Asheville, NC 2013
Zassenhaus Group Theory Conference

Isotopic Algebras with Nonisomorphic Congruence Lattices (slides) — Boulder, CO 2013
AMS Special Session on Algebras, Lattices, and Varieties

Synchronizing Automata and the Černý Conjecture (slides) — Boulder, CO 2013
Graduate Algebra Seminar, University of Colorado

Talks (continued)

The Finite Lattice Representation Problem in Four Parts — Columbia, SC 2012
University of South Carolina Algebra and Logic Seminar

Interval Sublattice Enforceable Properties of Finite Groups (slides) — Columbus, OH 2012
The 31st Ohio State-Denison Mathematics Conference

Expansions of Finite Algebras and their Congruence Lattices (slides) — Honolulu, HI 2012
American Mathematical Society sectional meeting

Intervals in Subgroup Lattices and Permutation Representations — Cullowhee, NC 2012
Western Carolina University Group Theory Seminar

Recent Progress on the Finite Lattice Representation Problem — Honolulu, HI 2011
Achievement Rewards for College Scientists: Scholar Presentations

The Finite Lattice Representation Problem — Seoul, KOR 2009
First Joint Meeting of the Korean and American Mathematical Societies

References

Ralph Freese
Professor of Mathematics
University of Hawaii
2565 McCarthy Mall
Honolulu, HI 96822
phone: 808-956-4680
email: ralph@math.hawaii.edu

Clifford Bergman†
Professor of Mathematics
Iowa State University
396 Carver Hall
Ames, Iowa 50011
phone: 515-294-1752
email: cbergman@iastate.edu

George McNulty
Professor of Mathematics
University of South Carolina
1523 Greene Street
Columbia, SC 29208
phone: 803-777-7469
email: mcnulty@math.sc.edu

Peter Mayr†
Assistant Professor of Mathematics
University of Colorado, Boulder
2300 Colorado Avenue
Boulder, CO 80309
phone: 303-492-7754
email: peter.mayr@colorado.edu

Peter Jipsen
Professor of Mathematics
Chapman University
545 W. Palm Ave
Orange, CA 92866
phone: 714-744-7918
email: jipsen@chapman.edu

Agnes Szendrei†
Professor of Mathematics
University of Colorado, Boulder
2300 Colorado Avenue
Boulder, CO 80309
phone: 303-492-7683
email: szendrei@colorado.edu

J.B. Nation
Emeritus Professor of Mathematics
University of Hawaii
2565 McCarthy Mall
Honolulu, HI 96822
phone: 808-956-4680
email: jb@math.hawaii.edu

Bill Lampe†
Emeritus Professor of Mathematics
University of Hawaii
2565 McCarthy Mall
Honolulu, HI 96822
phone: 808-956-4680
email: bill@math.hawaii.edu

† teaching reference

Teaching Philosophy, Mentoring, and Teaching Experience

William DeMeo

1. Introduction

After five years of teaching math and computer science in college classrooms, my motivation to teach continues to grow stronger. I have had the good fortune of working in math departments that place a heavy emphasis on teaching, and this experience has given me a deeper appreciation and respect for our responsibility to pass on knowledge to future generations. It has also taught me many lessons about how to teach well.

Since 2012 I have taught two math courses every semester, and have taken every opportunity available to attend teacher training sessions, read student evaluations, and solicit feedback from both students and peers. This has given me many occasions on which to reflect on my beliefs about education in general and effective teaching in particular. I have come to realize that my mission as a teacher revolves around three core principles, each of which has the main goal of engaging the student:

(1) **leadership,** to engage students by demonstrating a deep appreciation and passion for the subject;
(2) **communication,** to engage students by connecting with them at an appropriate level;
(3) **pedagogy,** to engage students by developing a set of best practices that stimulate them mentally and facilitate understanding.

The sections that follow consider each of these core principles in more detail and describe how they have come to form the core of my teaching philosophy.

2. Core Principles

2.1. **Leadership.** My passion for the subject being taught must be apparent to the students from the first day of class. This is essential for establishing credibility and, more importantly, for convincing the students that the subject is truly worthy of their attention. Such enthusiasm is contagious and helps motivate students to engage in active listening, participation, and learning. I have found that convincing students of my passion for whatever subject I teach is actually the easiest part of my job as an educator, and each semester my teaching evaluations reveal that my love of the subject is both obvious and contagious.

2.2. **Communication.** I strive to have an open, two-way dialog of friendly yet respectful communication in the classroom. It is important that all students—regardless of background, race, nationality, or gender—feel welcomed and encouraged to participate in a safe environment. Furthermore, it is important to let students know my door is always open for them to come to me with any issues they might encounter in class, about math or their academic and professional lives.

To accomplish these goals, I have developed a variety of strategies based on essential educational principles of active learning, diversity and sensitivity training, planning, presentation and assessment. Some of these techniques depend vitally on our awareness of student backgrounds, interests, and abilities, as we discuss in the next section.

2.3. **Learner diversity.** As we enjoy larger numbers of intelligent and eager learners from more and more diverse backgrounds, from minority groups, both domestic and international, we must be cognizant of both the learning style of individuals and the cultural diversity of the whole class. Using this as our guide, we can customize our instruction and assessment accordingly. Since a variety of learning styles are likely to be represented in any modern classroom, it is important to alternate frequently between instruction and assessment. Without frequent learner feedback and participation in every class, it is impossible to foster a comfortable atmosphere of communication and mutual understanding, and without that, we cannot assess the level of student engagement and we cannot know the degree to which students have processed the material covered.

3. Pedagogy

3.1. **Cycles and Iterations.** One effective method of teaching is to cycle through various modes of interaction with the students: presentation, participation, feedback and assessment. Each of these modes should itself vary throughout the class and the semester. Presentation mode, for example, can vary across a number of dimensions. First, the medium of presentation should alternate between speech/audible and written/graphical/visual to accommodate differences of learners.

Second, the level of detail at which material is presented should proceed along a spectrum from general to specific. This may depend on the subject matter, but I find it helpful to outline and motivate the entire course in the first lecture. Thereafter, roughly the first quarter of the course may be explained and motivated. Finally, each subsection of the first quarter is described and motivated, still using fairly general terms. Finally each section may be covered in detail.

Proceeding along such a path, from the general to the specific, reinforces material by reiterating the same idea at various levels of abstraction. While watching Bob Harper's lectures at CMU, I learned that this strategy has a name: "iterative deepening." Besides the obvious reinforcement effect that repetition has, another positive outcome of iterative deepening is that learners leave the course with a better understanding of how each topic fits into the greater scheme of the course. It is this broad view of the course, rather than the details covered in a particular section, that is likely to remain with the students long after the course has ended.

3.2. **The Learner's Role.** Another effective way to teach a classroom of diverse learners, is to help the students stay engaged by "making the subject their own." One can accomplish this by giving them some role in determining what they will learn and how they will learn it. If students are given the chance to arrive at an understanding of some problem or topic by relying on their own ingenuity and deriving their own methods for solving the problem, then, even if the problem remains unsolved, the student's appreciation of the problem or topic is typically much deeper than if they had listened passively to a lecture. This sort of learning can be facilitated in the classroom by providing just the right amount of guidance so that learners can reach correct conclusions. However, determining the appropriate level of guidance depends crucially on knowing the backgrounds and abilities of the learners.

Over the past few years I have experimented with this approach to great effect. When introducing a new topic, I often try to provide just an outline of the basic ideas, and a precise description of our objective and then ask students to think, to grow accustomed to

seeking answers themselves, to not wait for more guidance or answers to come from someone else. Often I leave out basic terminology and help students develop what seems like the most natural definitions and notation that might help to reach the goal. We then embark upon a journey of discovering the theory together, by considering the class's collective contributions, and making the necessary adjustments. In some cases, the students do not need very much help to arrive at sensible and useful definitions. Before long our discussion can be honed into a more formal development of the important principles, theorems, and methods.

To give a simple example, I was recently impressed by a classroom of freshman and sophomore computer science majors who, with little guidance, had no trouble formulating set-theoretic definitions of the natural numbers, the integers, ordered pairs, and n-tuples, despite having had no prior exposure to set theory. The notions of equivalence relation and partial ordering also seemed quite natural to most of them. Thereafter, when the class participated in the lecture and as we solved example problems, employing the definitions that *they* "invented," the students seemed not only more energized than usual, but also more comfortable with the material—not surprisingly, since they had a hand in its development. Again, the goal is for students to *make the subject their own.*

This experience has given me a new perspective on effective teaching, and I have learned that with only minor adjustments in presentation style, some topics can be treated in a manner that helps students gain a deeper understanding than if they passively listen to a lecture. Furthermore, with very careful planning and time management, this approach doesn't necessarily require significantly more class time.

Providing students with the main ideas and objectives and encouraging them to develop their own methods and come to their own conclusions is not always appropriate for every topic at every level. However, there is strong evidence suggesting that, at least at the elementary school level, a hands-off approach can have quite miraculous outcomes.[1] At the college and graduate levels there is a standard curriculum that must be covered in a limited amount of time, and an entirely hands-off approach is infeasible and inappropriate. However, the benefits of class participation can be tremendous and must not be ignored at any level.

[1] See, for example, http://www.wired.com/business/2013/10/free-thinkers/

4. Mentoring Experience

Among my most gratifying accomplishments as an educator has been successfully engaging students in math and computer science research. The following subsections highlight some of these activities.

4.1. **Research Experiences for Undergraduates (REU).** In the summer of 2016 at Iowa State, Cliff Bergman and I co-mentored a bright and precocious math major from University of Rochester named Charlotte Aten. It was quickly apparent that Charlotte was not satisfied with anything less than most general and elegant solution to a given problem. It was this taste for generality that attracted her to universal algebra and led her to Ames. It turned out, however, that even universal algebra was not general enough for Charlotte, so I suggested we take up category theory. This was a good fit and we spent much of the summer working through the basics of category theory up to the Yoneda Lemma. Consequently, Charlotte managed to accomplish one of her goals, which was to work out a generalized version of Cayley's Theorem for algebraic and relational structures.

4.2. **TypeFunc Research Group.** At Iowa State I founded a research group called TypeFunc (github.com/TypeFunc) which convened weekly during the academic year. In its first year, the group consisted of four undergraduate students—Paul Lubberstedt (math major), Hunter Praska (cs major), Hyeyoung Shin (cs major), and Joshua Thompson (math major)—and the goal was to generate interest in some approachable open research problems in areas related to algebra, logic, complexity and theoretical computer science, and to guide the students' reading and research activities in these areas.

There were some student presentations in spring 2015. Thompson presented proofs of some known results; he showed that the Law of Excluded Middle (LEM) is not refutable in Intuitionistic Propositional Logic, and he proved Diaconescu's Theorem (Choice implies LEM). Lubberstedt gave a presentation on the meaning of "judgment" in intuitionistic logic. I gave demonstrations of the Coq proof assistant, which motivated two of the students (Praska and Shin) to learn Coq by working through "Software Foundations" by Pierce, et al [PdAC$^+$16]. Shin has since become an expert in Coq after working her way through most of the Software Foundations book, and then using the proof assistant to model a simple imperative programming language.

In late spring 2016 I taught the TypeFunc group about the Constraint Satisfaction Problem (csp) research I had been engaged in with Cliff Bergman. I explained the algebraic approach to the so called "csp dichotomy conjecture." The students seemed fascinated by this topic and, by the end of the semester, were very eager to attend a related workshop at Vanderbilt called "Open Problems in Universal Algebra" [Sha15].

4.3. **Mentoring at the University of Colorado, Boulder.** During the Spring 2018 semester I co-lead (with Keith Kearnes) a reading course based on the book *Algebraic Theories*, by Jiri Adamek [ARV11]. This course was attended by a very talented graduate student named Paul Lassard. I also had the opportunity to serve on a number of Ph.D. prelim exam committees (for Jordan DuBeau, Ali Latfi, Athena Sparks, and Michael Wheeler), one Ph.D. thesis defense committee (for Jeffrey Shriner), and one honors thesis defense committee (for Zetong Xue).

4.4. **Undergraduate Thesis Advising at Iowa State University.** At Iowa State, I advised and collaborated with Joshua Thompson, one of our brightest math majors, and a student in the Honors College. Thompson worked with me on the project of determining which algebraic structures have "absorbing subuniverses," and he (along with Praska and Shin, of the TypeFunc group) accompanied Cliff Bergman and me to the Shanks Workshop on "Open Problems in Universal Algebra" at Vanderbilt in 2015 [Sha15]. Thompson has said this was a "life-changing experience."

Josh Thompson's senior thesis research was a great success and he solved the open problem that I had presented him with. Specifically, Thompson identified all proper absorbing subuniverses of the 4-element algebras that Bergman and I had been studying as part of our algebraic CSP project.

4.5. **Undergraduate Thesis Advising at University of South Carolina.** I advised and collaborated with Matthew Corley, a talented undergraduate student in the Honors College, who worked with me on a project involving nonabelian harmonic analysis of musical signals. Professor Reginald Bain (Director of the Experimental Music Studio in USC's Music Department) joined this project as a co-mentor. I guided Corley though the process of developing a formal project proposal, research schedule and budget, which we submitted through the university's grant office. The proposal earned Corley a prestigious *Magellan Scholarship Grant* to support his work on the project. I gave a preliminary report on this collaboration in a talk entitled "What does a nonabelian group sound like?" at the MAA session *At the Intersection of Mathematics and the Arts*, which took place at the Joint Math Meetings in Baltimore, January, 2014.[2]

4.6. **Pi Mu Epsilon and High School Math Contests.** For two years at University of South Carolina I served as a faculty mentor for Pi Mu Epsilon and the Gamecock Math Club, and served on the High School Math Contest Committee charged with creating the written portion of the South Carolina High School Math Contest exam (2013, 2014).

4.7. **Mentoring at the University of Hawaii.** As a graduate student, I had the opportunity to serve as mathematics mentor to the undergraduates of the Bio-Math Program—a joint program with students from the Biology, Mathematics, and Zoology Departments. These students worked on the project of cataloging sounds of marine life, and especially fish species, with the goal of creating a comprehensive library of audio samples that could be used by zoologists for their research. Since I have a great deal of experience in digital signal processing and dsp software, my primary roles were teaching the students some signal processing theory and helping them get up to speed with the software tools required to achieve their goals.

[2]An abstract for the MAA talk is available at:
http://www.ams.org/amsmtgs/2160_abstracts/1096-c5-2578.pdf
The project web page resides at http://soundmath.github.io/GroupSound.

5. Curriculum Development

5.1. **Lean in the Classroom.** I was lucky enough to teach a graduate course in Model Theory during the Spring 2018 semester at University of Colorado, Boulder. This year I am teaching Abstract Algebra in the Fall of 2018, and Discrete Math (which serves as an "introduction to proofs" class) in both the Fall and Spring semesters of the 2018/19 academic year. My experience with these courses has convinced me that the *Lean Theorem Proving Language* has an important role to play in the teaching logic and proof to undergraduates.

In the Fall 2018, I presented logic to my first- and second-year undergraduate students using Gentzen style natural deduction almost exclusively. (Of course, the students were also exposed to classical logic and truth tables, and I taught them the not-so-subtle differences between classical and constructive reasoning principles.) Because natural deduction involves only a very small set of concrete derivation rules, and essentially no axioms, it is ideal for getting the students up and proving with first-order logic. The students seem more confident and satisfied with their proofs than the students in previous courses, when the treatment was more traditional, based entirely on set theory, truth tables, and proof sequences.

Another benefit of using Gentzen style natural deduction was that it enabled the students to better appreciate *soundness* and *completeness*. It seems easier to explain these concepts to students who have some experience building proof derivation trees (that are purely syntactic) on the one hand, while on the other hand having some experience with truth values and what it means for a formula to be true in a particular model.

Evidently, Paul Taylor has witnessed a similar phenomenon, as he descirbes in [Tay99], where he refers to "proof boxes" (essentially natural deduction diagrams).

> "I have seen logic introduced to first year undergraduates both in the form of truth-assignments and using proof boxes, and firmly believe that the box method is preferable... it is a formal version of the way mathematicians actually reason, even if they claim to use Boolean algebra when asked."

About half of the students in my undergraduate class were computer science majors, and most of them have had some prior programming experience. So, I decided to introduce Lean in the classroom in order to keep the students more engaged and to further develop their logical agility and reinforce their understanding of the reciprocal roles played by the introduction and elimination rules of natural deduction. At the same time, I did not want to alienate any students who had no experience with, and possibly some aversion to, computer programming. As a compromise I introduced Lean in the classroom via team homework assignments. As it turns out, their Lean proofs were among their best proofs! They became quite skilled at constructing formal proofs in Lean, and their confidence and understanding of the analogous paper-and-pencil proofs seemed much stronger after having worked with the proof assistant.

Other educators have reported similar successful outcomes with proof assistants in a classroom setting [Nip12, Pie09]. Kevin Buzzard at Imperial College London has the ambitious goal of formalizing the entire undergraduate mathematics curriculum in Lean [Buz18]. These circumstances point to a future where computer-aided theorem proving tools will be routinely used, resulting in better teaching, deeper student understanding, and ultimately future generations of scientists who are more productive and reliable.

5.2. **Colorado Lean User's Group.** During my first year as a postdoc at CU Boulder, I started the *Colorado Lean User's Group* with the goal of encouraging graduate students in math and computer science to learn how to conduct their work in the Lean proof assistant. This effort centered around weekly hour-long meetings in the CU *Programming Languages and Verification Lab* conference room with about five graduate students. We used the hour to learn about and discuss Lean. By the end of the year, some of us had gained a fair amount of expertise in Lean and we are now using the language to support our research projects.

5.3. **Short Course on Mathematical Foundations of Computing (MFC).** During my time as a visiting assistant professor at University of Hawaii, I started the "Hawaii MFC Circle," the Hawaii Chapter of TypeFunc (see Section 4.2). With help from some colleagues—Dusko Pavlovic in Information and Computer Science and Bjørn Kjos-Hanssen in Math—I developed a short course on the Mathematical Foundations of Computing Science that took place in April of 2017. Modelled on the successful European summer schools in theoretical computer science, the course covered a subset of the following topics: Lambda Calculus, Type Theory, Functional Programming and Dependent Types, Automated Theorem Proving in Coq.

The enthusiastic response to the initial proposal from graduate students in the Math and Computer Science Departments—and even one student at the Institute for Astronomy—surpassed my expectations. Within the first 24 hours, over a dozen students expressed a desire to take the course, and some even said they had been waiting for this kind of offering since entering the phd program. In the end we taught about 10 students for three weeks in April. Our Github team repository, including course materials, can be found at github.com/TypeFunc.

References

[ARV11] J. Adámek, J. Rosický, and E. M. Vitale. *Algebraic theories*, volume 184 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 2011. A categorical introduction to general algebra, With a foreword by F. W. Lawvere.

[Buz18] Kevin Buzzard. Xena project. weblog, 2018. URL: https://xenaproject.wordpress.com/.

[Nip12] T. Nipkow. Teaching semantics with a proof assistant: No more lsd trip proofs. pages 24–38. Springer, 2012.

[PdAC+16] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, and Brent Yorgey. *Software Foundations*. Electronic textbook, 2016. Version 4.0. http://www.cis.upenn.edu/~bcpierce/sf.

[Pie09] B. C. Pierce. Lambda, the ultimate TA: Using a proof assistant to teach programming language foundations. In G. Hutton and A. P. Tolmach, editors, *International Conference on Functional Programming*, pages 121–122. ACM, :2009.

[Sha15] Open problems in universal algebra, a shanks workshop at vanderbilt university, May 2015. http://www.math.vanderbilt.edu/~moorm10/shanks/.

[Tay99] Paul Taylor. *Practical foundations of mathematics*, volume 59 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1999.

University of Colorado, Boulder (as Burnett Meyer Instructor)

Math 2001: Discrete Mathematics	Spring 2019
Math 2001: Discrete Mathematics	Fall 2018
Math 3140: Abstract Algebra	Fall 2018
Math 6000: Model Theory (graduate course)	Spring 2018
Math 2130: Linear Algebra	Spring 2018
Math 2130: Linear Algebra	Fall 2017

University of Hawaii (as Visiting Assistant Professor)

Math 215: Applied Calculus	Spring 2017
Math 480: Senior Seminar	Spring 2017
Math 244: Calculus IV	Fall 2016
Math 321: Introduction to Advanced Math	Fall 2016

Iowa State University (as Postdoctoral Associate)

Math 317: Linear Algebra	Spring 2016
Math 317: Linear Algebra	Fall 2015
Math 160: Survey of Calculus	Fall 2015
Math 207: Elementary Linear Algebra	Spring 2015
Math 165: Calculus I	Spring 2015
Math 301: Abstract Algebra	Fall 2014
Math 165: Calculus I	Fall 2014

University of South Carolina (as Visiting Assistant Professor)

Math 700: Linear Algebra (graduate course)	Spring 2014
Math 141: Calculus I	Spring 2014
Math 374: Discrete Structures	Fall 2013
Math 122: Calculus for Business and Social Sciences	Fall 2013
Math 374: Discrete Structures	Spring 2013
Math 122: Calculus for Business and Social Sciences	Spring 2013
Math 241: Vector Calculus	Fall 2012
Math 122: Calculus for Business and Social Sciences	Fall 2012

University of Hawaii (as Graduate Student Instructor)

Math 371: Probability Theory	Summer 2011
Math 215: Applied Calculus I	Summer 2009
Math 100: Mathematical Reasoning	Summer 2010

Publications of William DeMeo (as of January 2019)

Journal Articles

1. Universal algebraic methods for constraint satisfaction problems with Clifford Bergman; to appear in *Logical Methods in Computer Science (LMCS);* preprint link: arXiv [cs.LO] 1611.02867

2. Polynomial-time tests for difference terms in idempotent varieties with Freese and Valeriote; to appear in *International Journal of Algebra & Computation (IJAC);* preprint link: diffTerm-ijac-r1-draft-20180905.pdf

3. Isotopic algebras with nonisomorphic congruence lattices *Algebra Universalis* **72**:295–298, 2014; preprint link: Isotopy-AU-2014.pdf

4. Expansions of finite algebras and their congruence lattices *Algebra Universalis* **69**:257–278, 2013; preprint link: DeMeo-Expansions-AU-2013.pdf

Refereed Conference Proceedings

5. Proceedings of Algebras and Lattices in Hawaii 2018 editor with K. Adaricheva, J. Hyndman; preprint link: lulu.com

6. Topics in nonabelian harmonic analysis and DSP applications *Proceedings of the International Symposium on Musical Acoustics* Nara, Japan 2004 (best paper award); preprint link: DeMeo-ISMA2004-FinalPaper.pdf

7. Characterizing musical signals with Wigner-Ville interferences *Proceedings of the International Computer Music Conference (ICMC);* Göteborg, Sweden 2002; preprint link: DeMeo-ICMC2002.pdf

8. Approximating eigenvalues of large stochastic matrices *Proceedings of the 8th Copper Mt. Conference on Iterative Methods* Colorado, USA 1998 preprint link: williamdemeo.github.io/MSThesis

Papers in Progress

- A new characterization of fiber products of lattices with P. Mayr and N. Ruskuc; preprint link: fg-free-lat-draft-20181018.pdf

- Representing finite lattices as congruence lattices of finite algebras with R. Freese and P. Jipsen; preprint link: article-v20160610.pdf

Books in Progress

- Algebras, Categories and Types: with computer-aided proofs with Hyeyoung Shin.

- A Concise Course in Category Theory with Charlotte Aten and Venanzio Capretta.

- Problems in Real and Complex Analysis link: PSRCA.html

- Problems in Groups and Rings link: PSRCA.html

Brief Summary of Research Activities

William DeMeo

1. Introduction

The last two decades have witnessed explosive growth in the number of applications of abstract mathematics, especially in computer science, and this trend continues on a steep upward trajectory. Mathematical fields like *universal algebra* and *category theory* have long had a substantial influence on the development of theoretical computer science, particularly in *domain theory*, *denotational semantics*, and *programming languages research* [13, 20]. Dually, progress in theoretical computer science has informed and inspired a substantial amount of pure mathematics in the last half-century [2, 3, 5, 6, 19, 20], just as physics and physical intuition motivated so many of the mathematical discoveries of the last two centuries.

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* [4, 12, 8, 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 ideally suited for expressing the key concepts and result of universal algebra. As such, Lean is the primary language we have chosen for our most recently launched project, *Formal Foundations for Informal Mathematics Research*. This project is briefly mentioned in Section 2.1 below and described in detail in the document demeo_informal_foundations.pdf.

Universal algebra has also been invigorated by a recently discovered connection to complexity theory, and this connection was the primary focus of my work from 2015 to 2017. The theory of finite algebras has turned out to be broadly applicable in obtaining deep and definitive results about the complexity of algorithmic problems in the broad class of *constraint satisfaction problems* (CSPs). We call this area *algebraic CSP theory*. The tools of universal algebra, combined with combinatorial reasoning about polymorphisms (multi-variable endomorphisms) acting on finite graphs and other relational structures, has produced deep new results concerning the complexity of CSPs, and has explained and united older results. Furthermore, almost all of the new results have been turned around and used to produce startling new algebraic results of a kind never seen before in universal algebra.

There is now an extensive and growing research literature on algebraic CSP theory. Consequently, the field of universal algebra is more active now than at any other time in its brief eighty-five year history. My work in algebraic CSP is summarized in Section 2.4 below. Details can be found in the manuscript entitled "Universal Algebraic Methods for Constraint Satisfaction Problems," [7] that I authored with Cliff Bergman, available on the arXiv.

2. Research Projects

2.1. **Lean and Formal Foundations.** I am fascinated by the connections between programming languages and mathematics, and my most recently initiated research program aims to develop a library of all core definitions and theorems of universal algebra in the Lean proof assistant and programming language [1]. The title of this project is *Formal Foundations for Informal Mathematics Research* and a detailed project description is available in the document demeo_informal_foundations.pdf.

2.2. **New Characterizations of Bounded Lattices and Fiber Products.** About a year ago I began collaborating with Peter Mayr (CU Boulder) and Nik Ruskuc (University of St. Andrews) who were interested in knowing when a homomorphism $\varphi\colon \mathbf{F} \to \mathbf{L}$ from a finitely generated free lattice $\mathbf{F}$ onto a finite lattice $\mathbf{L}$ has a kernel $\ker\varphi$ that is a finitely generated sublattice of $\mathbf{F}^2$. We conjectured that this could be characterized by whether or not the homomorphism is *bounded*.[1] and I presented a proof of one direction of this conjecture at the Algebras and Lattices in Hawaii conference earlier this year. Last month we proved the converse and thus confirmed our conjecture. All along Mayr and Ruskuc have had in mind an application for the fact that our new result is equivalent to a characterization of *fiber products* of lattices. With the proof of our new characterization theorem complete, we expect to have a manuscript ready for submission by January 2019.

2.3. **Tractability of Deciding Existence of Special Terms.** Among my most recent research accomplishments was the discovery that a certain decision problem about algebraic structures that previously seemed out of reach is actually tractable. In collaborative project with Ralph Freese (University of Hawaii) and Matthew Valeriote (McMaster University), we considered the following practical question: Given a finite algebra $\mathbf{A}$ in a finite language, can we efficiently decide whether the variety generated by $\mathbf{A}$ has a so called *difference term*? In a paper that will soon appear in *IJAC*, we answer this question (positively) in the idempotent case and then describe algorithms for constructing difference term operations [9].

A *difference term* for a variety $\mathcal{V}$ is a ternary term d in the language of $\mathcal{V}$ that satisfies the following: if $\mathbf{A} = \langle A, \ldots\rangle \in \mathcal{V}$, then for all $a, b \in A$ we have

$$d^{\mathbf{A}}(a,a,b) = b \quad \text{and} \quad d^{\mathbf{A}}(a,b,b)\ [\theta,\theta]\ a, \tag{1}$$

where θ is any congruence containing (a,b) and $[\cdot,\cdot]$ denotes the *commutator*. When the relations in (1) hold for a single algebra $\mathbf{A}$ and term d we call $d^{\mathbf{A}}$ a *difference term operation* for $\mathbf{A}$.

Difference terms are studied extensively in the general algebra literature. (See, for example, [14, 15, 16, 17, 18].) There are many reasons to study difference terms, but one obvious reason is the following: if we know that a variety has a difference term, this fact allows us to deduce that the algebras inhabiting this variety must

[1] See [10] for the definition of a bounded lattice homomorphism.

satisfy certain interesting properties. Perhaps the most important property can be summarized in the following heuristic slogan: *varieties with a difference term have a commutator that behaves nicely.*

The class of varieties that have a difference term is fairly broad and includes those varieties that are congruence modular or congruence meet-semidistributive. Since the commutator of two congruences of an algebra in a congruence meet-semidistributive variety is just their intersection [17], it follows that the term $d(x, y, z) := z$ is a difference term for such varieties. A special type of difference term $d(x, y, z)$ is one that satisfies the equations $d(x, x, y) = y$ and $d(x, y, y) = x$. Such terms are called *Maltsev terms*. So if $\mathbf{A}$ lies in a variety that has a difference term $d(x, y, z)$ and if $\mathbf{A}$ is *abelian* (i.e., $[1_A, 1_A] = 0_A$), then d will be a Maltsev term for $\mathbf{A}$.

Difference terms also play a role in recent work of Keith Kearnes, Agnes Szendrei, and Ross Willard. In [14] these authors give a positive answer Jónsson's famous question—whether a variety of finite residual bound must be finitely axiomatizable—for the special case in which the variety has a difference term.[2]

Computers have become invaluable as a research tool and have helped to broaden and deepen our understanding of algebraic structures and the varieties they inhabit. This is largely due to the efforts of researchers who, over the last three decades, have found ingenious ways to coax computers into solving challenging abstract algebraic decision problems, and to do so very quickly. To give a couple of examples related to our own work, it is proved in [21] (respectively, [11]) that deciding whether a finite idempotent algebra generates a variety that is congruence-n-permutable (respectively, congruence-modular) is *tractable*. Our work continues this effort by presenting an efficient algorithm for deciding whether a finitely generated idempotent variety has a difference term.

2.4. **Algebras and Algorithms, Structure and Complexity.** In 2015, I joined a group of 8 other scientists to form a universal algebra research group and secure a 3-year NSF grant for the project, "Algebras and Algorithms, Structure and Complexity Theory." Our focus is on fundamental problems at the confluence of logic, algebra, and computer science, and our main goal is to deepen understanding of how to determine the complexity of certain types of computational problems. We focus primarily on classes of algebraic problems whose solutions yield new information about the complexity of CSPs. These include scheduling problems, resource allocation problems, and problems reducible to solving systems of linear equations. CSPs are theoretically solvable, but some are not solvable efficiently. Our work provides procedures for deciding whether a given instance of a CSP is tractable or intractable, and we develop efficient algorithms for finding solutions in the tractable cases. My work on this project culminated in a 50-page manuscript, co-authored with Cliff Bergman, entitled "Universal Algebraic Methods for Constraint Satisfaction Problems" [7]. This was recently accepted for publication in the journal *Logical Methods in Computer Science* (LMCS); a draft is at arXiv [cs.LO] 1611.02867.

[2] To say a variety has *finite residual bound* is to say there is a finite bound on the size of the subdirectly irreducible members of the variety.

References

[1] Jeremy Avigad, Mario Carneiro, Leonardo de Moura, Johannes Hölzl, and Sebastian Ullrich. The lean theorem proving language. GitHub.com, 2018. URL: https://leanprover.github.io/.

[2] Andrej Bauer. Five stages of accepting constructive mathematics. *Bull. Amer. Math. Soc. (N.S.)*, 54(3):481–498, 2017. doi:10.1090/bull/1556.

[3] Andrej Bauer. On fixed-point theorems in synthetic computability. *Tbilisi Math. J.*, 10(3):167–181, 2017. doi:10.1515/tmj-2017-0107.

[4] 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.

[5] Andrej Bauer and Jens Blanck. Canonical effective subalgebras of classical algebras as constructive metric completions. *J.UCS*, 16(18):2496–2522, 2010.

[6] Andrej Bauer, Gordon D. Plotkin, and Dana S. Scott. Cartesian closed categories of separable Scott domains. *Theoret. Comput. Sci.*, 546:17–29, 2014. doi:10.1016/j.tcs.2014.02.042.

[7] 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.

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

[9] 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/.

[10] Ralph Freese, Jaroslav Ježek, and J. B. Nation. *Free lattices*, volume 42 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 1995. URL: http://dx.doi.org/10.1090/surv/042, doi:10.1090/surv/042.

[11] Ralph Freese and Matthew A. Valeriote. On the complexity of some Maltsev conditions. *Internat. J. Algebra Comput.*, 19(1):41–77, 2009. URL: http://dx.doi.org/10.1142/S0218196709004956, doi:10.1142/S0218196709004956.

[12] 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.

[13] Martin Hyland and John Power. The category theoretic understanding of universal algebra: Lawvere theories and monads. In *Computation, meaning, and logic: articles dedicated to Gordon Plotkin*, volume 172 of *Electron. Notes Theor. Comput. Sci.*, pages 437–458. Elsevier Sci. B. V., Amsterdam, 2007. URL: https://doi-org.colorado.idm.oclc.org/10.1016/j.entcs.2007.02.019, doi:10.1016/j.entcs.2007.02.019.

[14] Keith Kearnes, Ágnes Szendrei, and Ross Willard. A finite basis theorem for difference-term varieties with a finite residual bound. *Trans. Amer. Math. Soc.*, 368(3):2115–2143, 2016. URL: http://dx.doi.org/10.1090/tran/6509, doi:10.1090/tran/6509.

[15] Keith A. Kearnes. Varieties with a difference term. *J. Algebra*, 177(3):926–960, 1995. URL: http://dx.doi.org/10.1006/jabr.1995.1334, doi:10.1006/jabr.1995.1334.

[16] Keith A. Kearnes and Emil W. Kiss. The shape of congruence lattices. *Mem. Amer. Math. Soc.*, 222(1046):viii+169, 2013. URL: http://dx.doi.org/10.1090/S0065-9266-2012-00667-8, doi:10.1090/S0065-9266-2012-00667-8.

[17] Keith A. Kearnes and Ágnes Szendrei. The relationship between two commutators. *Internat. J. Algebra Comput.*, 8(4):497–531, 1998. URL: http://dx.doi.org/10.1142/S0218196798000247, doi:10.1142/S0218196798000247.

[18] Keith A. Kearnes, Ágnes Szendrei, Ross Willard, and Keith A. Kearnes. Simpler Maltsev conditions for (weak) difference terms in locally finite varieties. *Algebra Universalis*, 78(4):555–561, 2017. doi:10.1007/s00012-017-0475-7.

[19] Don Pigozzi and Antonino Salibra. Lambda abstraction algebras: representation theorems. *Theoret. Comput. Sci.*, 140(1):5–52, 1995. Selected papers of AMAST '93 (Enschede, 1993). URL: http://dx.doi.org/10.1016/0304-3975(94)00203-U, doi:10.1016/0304-3975(94)00203-U.

[20] Dana S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoret. Comput. Sci.*, 121(1-2):411–440, 1993. A collection of contributions in honour of Corrado Böhm on the occasion of his 70th birthday. URL: http://dx.doi.org/10.1016/0304-3975(93)90095-B, doi:10.1016/0304-3975(93)90095-B.

[21] M. Valeriote and R. Willard. Idempotent n-permutable varieties. *Bull. Lond. Math. Soc.*, 46(4):870–880, 2014. URL: http://dx.doi.org/10.1112/blms/bdu044, doi:10.1112/blms/bdu044.

Formal Foundations for Informal Mathematics Research

William DeMeo

Abstract. This document describes a research program, the primary goal of which is to develop and implement new theory and software libraries to support computer-aided mathematical proof in universal algebra and related fields. A distinguishing feature of this effort is the high priority placed on *usability* of the formal libraries produced. We aim to codify the core definitions and theorems of our area of expertise in a language that feels natural to working mathematicians with no special training in computer science. Thus our goal is a formal mathematical library that has the look and feel of the informal language in which most mathematicians are accustomed to working.

This research is part of a broader effort currently underway in a number of countries, carried out by disparate research groups with a common goal—to develop the next generation of **practical formal foundations for informal mathematics**, and to codify these foundations in a formal language that feels natural to mathematicians. In short, our goal is to present *mathematics as it should be*.

"Systems of axioms acquire a certain sanctity with age, and in the *how* of churning out theorems we forget *why* we were studying these conditions in the first place... Even when the mathematical context and formal language are clear, we should not perpetuate old proofs but instead look for new and more perspicuous ones."

–Paul Taylor
in *Practical Foundations of Mathematics*

Date: 2019-01-09.

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 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.

One hopes that published mathematical results *could* be translated into the rigorous language of some system of logic and formally verified. Nonetheless, few would relish spending the substantial time and energy that such an exercise would require. 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. A relatively recent trend is challenging this status quo, however, and the number of mathematicians engaging in computer-aided mathematics research is on the rise [28, 30, 37]. Indeed, at the *Computer-aided Mathematical Proof* workshop of the *Big Proof Programme*, held in 2017 at the Isaac Newton Institute of Cambridge University, we witnessed the serious interest that leading mathematicians (including two Fields Medalists) showed in computer-aided theorem proving technology [34].

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 hardware, software, and mathematical proofs is on the rise. In fact, *constructive type theory* and *higher-order logics*, on which most modern proof assistants are based, have been vital in formalizing and confirming the proofs of landmark mathematical results, such as the *Four-Color Theorem* [26] and the *Feit-Thompson Odd Order Theorem* [27], as well as settling major open problems, such as the *Kepler Conjecture* [29].

Another kind of computer-based theorem proving tool is called an *automated theorem prover*, which is quite different from a proof assistant in that the former is designed to independently search for a proof of a given statement with little or no help from the user. (Contrast this with the *interactive* nature of a proof assistant.) Unfortunately, when a

proof is found by an automated theorem prover, it tends to be very difficult to read and understand, and it often seems impossible to translate an automatically generated proof into an Informal Language proof.

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 [21]. 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 [31]. 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 [45].

As further justification for enlisting 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 [30].

Beyond their importance as a means of establishing trust in mathematical results, formal proofs can also expose and clarify difficult steps in an argument. Even before one develops a formal proof, the mere act of expressing a theorem statement (including the foundational axioms, definitions, and hypotheses on which it depends) in a precise and (when possible) computable way almost always leads to a deeper understanding of the result. Moreover, as Blanchette notes in [13], proof assistants can help us keep track of changes across a collection of results (axioms, hypotheses, etc), which facilitates experimenting with variations and generalizations. When changing a definition, a mathematician equipped with a proof assistant is alerted to proofs that need repair, unnecessary or missing hypotheses, etc.

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." [28].

1.1. **The usability gap.** Despite the many advantages and the noteworthy success stories mentioned above, proof assistants remain relatively obscure. There are a number of obvious reasons for this. First and foremost, proof assistant software tends to be tedious to use. Most mathematicians experience a significant slow down in progress when they must not only formalize every aspect of their arguments, but also express such formalizations in a language that the software is able to parse and comprehend.

One question that leads to insight into this "usability gap" that plagues most modern proof assistants is why *computer algebra systems* do not suffer from the same problem. Simply put, computer algebra systems are more popular than interactive theorem provers. One reason is that the up-front cost to end users seems substantially higher for interactive

theorem provers than for computer algebra systems, and this is because the latter are typically conceived of by mathematicians whose primary aim is to build a system that presents things "as they should be," that is, as a mathematically educated user would expect.

In many proof assistants and automated theorem provers, the mathematics are often not presented as we would expect or like them to be. In [39], Pollet and Kerber argue that this is not just a deficiency of the user interface. The problem with theorem provers goes much deeper; it goes to the core of these systems, namely to *the formal representation of mathematical concepts and knowledge*. How easy or hard it is to codify theorems and translate informal mathematical arguments into formal proofs in a particular system depends crucially on the formal foundations of that system, and the way in which these foundations are represented in the system.

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.

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* [2]—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.

Other considerations that make Lean ideally suited to this project are the following:

- **Clean and efficient design.** Lean's design and engineering is unusually clean and efficient, as the Lean developers have combined the best features from existing proof assistants (e.g., Coq, Isabelle/HOL, and Z3).
- **Powerful and extensible proof automation support.** Lean's logic is very expressive, with emphasis placed on *powerful proof automation*. In fact, the proof automation system is easy to extend via *metaprograms that one can implement right in the Lean language itself!* In this way, Lean aims to *bridge the gap between interactive and automated theorem proving*.
- **Easy-to-read proofs.** Lean is unique among computer-based theorem proving tools in that its *proofs tend to be easy to read and understood*, without any special training. In fact, working in Lean often leads to formal proofs that are cleaner, more concise, and shorter than the corresponding proofs in the Informal Language. (We provide examples in Section 4 below.) This is a crucial feature if we expect the system to be adopted by mathematicians.
- **A rich type system supporting type classes, dependent types, and quotient types.** Lean's logical foundation is a variant of Coq's—a dependent type

theory called the *Calculus of Inductive Constructions* (CiC) [16]. But Lean has a number of advantages over Coq, especially for pure mathematicians. Most notably Lean's support of *quotient types* allows reasoning about quotients without relying mainly on setoids [4], and Lean's support for dependent types is smoother than Coq's, thanks to flexible pattern matching and a generalized congruence closure algorithm [41].

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 [12, 17, 18, 19], 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.

As Lean is a very young language, its domain-specific libraries are relatively small, but they are growing rapidly. *It is vital for mathematicians to get involved at this early stage and play a leading role in the development.* If we leave this entirely to our colleagues in computer science, they will base the development on their perception of our needs, history will likely repeat itself, and the resulting libraries and tools may fail meet the needs and expectations of working mathematicians.

3. Universal Algebra and its Role in the Project

Universal (or *general*) *algebra* has been invigorated in recent years by a small but growing community of researchers exploring the connections between algebra and computer science. Some of these connections were discovered only recently and were quite unexpected. Indeed, algebraic theories developed over the last 30 years have found a number of important applications in both of the two main branches of theoretical computer science—*Theory A*, comprising *algorithms* and *computational complexity*, and *Theory B*, comprising *domain theory*, *semantics*, and *type theory* (the theory of programming languages).[1] The present proposal falls within the scope of Theory B.

3.1. **Foundations of mathematics and computing science.** Universal algebra, lattice theory, and category theory have had deep and lasting impacts on theoretical computer science, particularly in the subfields of domain theory, denotational semantics, and programming languages research [33, 40]. Dually, progress in theoretical computer science has informed and inspired a substantial amount of pure mathematics in the last half-century [5, 6, 8, 9, 38, 40], just as physics and physical intuition motivated so many of the mathematical discoveries of the last two centuries.

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*

[1]The Theory A–Theory B dichotomy was established by "The Handbook of Theoretical Computer Science" [43, 44], Volume A of which includes chapters on algorithms and complexity theory; Volume B covers domain theory, semantics, and type theory.

or *procedural programming languages* [7, 32, 15, Chs. 5 & 10]. The *Lean Programming Language* [2] 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.

In the remainder of this project description, we give some background on interactive theorem proving technology, introduce dependent and inductive types, and describe the Lean language, which will be the main vehicle for this project. We will explain how these technologies can be used to advance pure mathematics in general, and universal algebra in particular. We then present the concrete goals of the project, with some discussion of how we intend to accomplish them, and some examples of the achievements we have already made in pursuit of these goals. Before proceeding, 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* [3] 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* [2, 20].
(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.

4. Proof of Concept: Lean Universal Algebra

This section demonstrates the utility of dependent and inductive types by expressing some fundamental concepts of universal algebra in Lean. In particular, we will formally represent each of the following: *operation*, *algebra*, *subuniverse*, and *term algebra*. Our formal representations of these concepts will be clear, concise, and computable, and we will develop a notation and syntax that should seem natural and self-explanatory to algebraists. Our goal here is to demonstrate the power of Lean's type system for expressing mathematical concepts precisely and constructively, and to show that if we make careful design choices at the start of our development, then our formal theorems *and their proofs* can approximate the efficiency and readability of their Informal Language analogs.

4.1. Operations and Algebras.

We use ω to denote (our semantic concept of) the natural numbers. The symbols $\mathbb{N}$ and `nat` are synonymous, both denoting the *type of natural numbers*, as implemented in Lean. A *signature* $S = (F, \rho)$ consists of a set F of *operation symbols*, along with a *similarity type* function $\rho\colon F \to N$. The value $\rho f \in N$ is called the *arity* of f. In classical universal algebra we typically assume $N = \omega$, but for most of the

basic theory this choice is inconsequential and, as we will see when implementing general operations in Lean, it is unnecessary to commit in advance to a specific *arity type*.[2]

Classical universal algebra is the study of *varieties* (or *equational classes*) of algebraic structures where an *algebraic structure* is denoted by $\mathbf{A} = \langle A, F^{\mathbf{A}} \rangle$ and consists of a set A, called the *carrier* of the algebra, along with a set $F^{\mathbf{A}}$ of operations defined on A, one for each operation symbol; that is,

$$F^{\mathbf{A}} = \{f^{\mathbf{A}} \mid f \in F \text{ and } f^{\mathbf{A}} \colon (\rho f \to A) \to A\}.$$

Some of the renewed interest in universal algebra has focused on representations of algebras in categories other than **Set**, multisorted algebras, and higher type universal algebra [1, 10, 22, 25, 36]. These are natural generalizations that we plan to integrate into future versions of our `lean-ualib` software library, once we have a working implementation of the core of classical (single-sorted, set-based) universal algebra.

Suppose A is a set and f is a ρf-ary operation on A. In this case, we often write $f \colon A^{\rho f} \to A$. If N happens to be $\mathbb{N}$, then ρf denotes the set $\{0, 1, \ldots, \rho f - 1\}$ and a function $g \colon \rho f \to A$, identified with its graph, is simply a ρf-tuple of elements from A. The domain $A^{\rho f}$ can be represented by the type $\rho f \to A$ of functions from ρf to A, and we will represent operations $f \colon A^{\rho f} \to A$ using the function type $(\rho f \to A) \to A$.

Fix $m \in \mathbb{N}$. An m-tuple, $a = (a_0, a_1, \ldots, a_{m-1}) \in A^m$ is (the graph of) the function $a \colon m \to A$, defined for each $i < m$ by $a\, i = a_i$. Therefore, if $h \colon A \to B$, then $h \circ a \colon m \to B$ is the tuple whose value at i is $(h \circ a)\, i = h\, a\, i = h\, a_i$, which has type B. On the other hand, if $g \colon A^m \to A$, then $g\, a$ has type A. If $f \colon (\rho f \to B) \to B$ is a ρf-ary operation on B, if $a \colon \rho f \to A$ is a ρf-tuple on A, and if $h \colon A \to B$, then $h \circ a \colon \rho f \to B$, so $f(h \circ a) \colon B$.

4.2. Operations and Algebras in Lean (`lean-ualib/basic.lean`).

This section presents our implementation of operations and algebras in Lean, highlighting the similarity between the formal and informal rendering of these concepts. We start with the *type of operation symbols* and the *type of signatures*.

```
import data.set
definition op (β α) := (β → α) → α
```

An example of an operation of type `op (β α)` is the projection function π, of arity β on the *carrier type* α, which we define in Lean as follows:

```
definition π {β α} (i) : op β α := λ a, a i
```

The operation `π i` maps a given tuple `a : β → α` to its value at input `i`. For instance, suppose we have types α and β of arbitrary *height*,[3] and variables `i : β` and `f : β → α`.

```
variables (α : Type*) (β : Type*) (i : β) (f : β → α)
```

Then the command `#check π i f` shows that the type of `π i f` is α, as expected, since `π i f = f i`.

We define a signature as a structure with two fields, the type `F` of *operation symbols* and an *arity function* `ρ : F → Type*`, which takes each operation symbol f to its arity ρf.

```
structure signature := mk :: (F : Type*) (ρ : F → Type*)
```

[2] An exception is the *quotient algebra type* since, unless we restrict ourselves to finitary operations, lifting a basic operation to a quotient requires some form of choice.

[3] The *height* of a type is simply type's *level*, and the syntax `Type*` indicates that we do not wish to commit in advance to a specific height.

Next we define the *type of interpretations of operations* on the carrier type α. First, let us fix a signature `S` and define some convenient notation.[4]

```
section
  parameter {S : signature}
  definition F := S.F
  definition ρ := S.ρ
  definition algebra_on (α : Type*) := Π (f : F), op (ρ f) α
  -- (section continued at * below)
```

The first definition allows us to write `f : F` (instead of `f : S.F`) to indicate that the operation symbol `f` inhabits `F`; similarly, the second definition allows us to denote the arity of `f` by `ρ f` (instead of `S.ρ f`). In these two cases, the Lean syntax matches our Informal Language notation exactly.

The definition of `algebra_on` makes sense; if we are given a signature `S` and a carrier type α, then an `S`-algebra over α is determined by its operations on α.[5] An inhabitant of the type `algebra_on` assigns an interpretation to each `op` symbol `f : F`, which yields a function of type `(ρ f → α) → α`.

Finally, we define an algebra. Since an algebra pairs a carrier with an interpretation of the operation symbols, we use the *dependent pair type*, `Σ (x : A), B x`, also known as a *Sigma type*. This is the type of ordered pairs `<a, b>`, where `a : A`, and `b` has type `B a` which may depend on `a`. Just as the *Pi type* `Π (x : A), B x` generalizes the notion of function type `A → B` by allowing the codomain `B x` to depend on the value `x` of the input argument, a Sigma type `Σ (x : A), B x` generalizes the Cartesian product `A × B` by allowing the type `B x` of the second argument of the ordered pair to depend on the value `x` of the first.[6]

Since an algebra $\langle A, F^{\mathbf{A}} \rangle$ is an ordered pair where the type of the second argument depends on the first, it is natural to encode an algebra in type theory using a Sigma type, and we do so in the `lean-ualib` library as follows:

```
  -- (section continued from * above)
  definition algebra := sigma algebra_on
  instance alg_carrier : has_coe_to_sort algebra := ⟨_, sigma.fst⟩
  instance alg_operations : has_coe_to_fun algebra := ⟨_, sigma.snd⟩
end
```

The last two lines are tagged with `has_coe_to_sort` and `has_coe_to_fun`, respectively, because here we are using a very nice feature of Lean called *coercions*. Using this feature we can write programs using syntax that looks very similar to our Informal Language. For instance, the standard notation for the interpretation of the operation symbol f in the algebra $\mathbf{A} = \langle A, F^{\mathbf{A}} \rangle$ is $f^{\mathbf{A}}$. In our implementation, the interpretation of f is denoted

[4]The `section` command allows us to open a section throughout which our signature `S` will be available. The `section` ends when the keyword `end` appears below.

[5]plus whatever equational laws it may models; our handling of *theories* and *models* in Lean is beyond the scope of this project description; for more information, see `https://github.com/UniversalAlgebra/lean-ualib/`

[6]Lean's built-in `sigma` type is defined as follows:

```
structure sigma α : Type u (β : α → Type v) := mk :: (fst : α) (snd : β fst)
```

by `A f`. While `A f` is not identical to the Informal Language's $f^{\mathbf{A}}$, it is arguably just as elegant, and we believe that adapting to it will not be a great burden on the user. To see this notation in action, let us look at how the `lean-ualib` represents the assertion that a function is an `S`-homomorphism.

```
definition homomorphic {S : signature}
{A : algebra S} {B : algebra S} (h : A → B) :=
∀ f a, h (A f a) = B f (h ∘ a)
```

Comparing this with a common Informal Language definition of a homomorphism, which is typically something similar to $\forall f\ \forall a\ h(f^{\mathbf{A}}(a)) = f^{\mathbf{B}}(h \circ a)$, we expect working algebraists to find the `lean-ualib` syntax quite satisfactory.

4.3. **Subuniverses.** In this section, we describe another important concept in universal algebra, the *subuniverse*, and use it to illustrate one of the underlying themes that motivates this research project. Indeed, subuniverses give us our first opportunity to demonstrations the power of *inductively defined types*. Such types are essential for working with infinite objects in a constructive and computable way and for proving (by induction of course!) properties of these objects.

A *subuniverse* of an algebra $\mathbf{A} = \langle A, F^{\mathbf{A}} \rangle$ is a subset $B \subseteq A$ that is closed under the operations in $F^{\mathbf{A}}$. We denote by $\mathsf{S}\mathbf{A}$ the set of all subuniverses of $\mathbf{A}$. If B is a subuniverse of $\mathbf{A}$ and $F^{\mathbf{A}\restriction B} = \{f^{\mathbf{A}} \restriction B \mid f \in F\}$ is the set of basic operations of $\mathbf{A}$ restricted to B, then $\mathbf{B} = \langle B, F^{\mathbf{A}\restriction B} \rangle$ is a *subalgebra* of $\mathbf{A}$. Conversely, all subalgebras are of this form.

If $\mathbf{A}$ is an algebra and $X \subseteq A$ a subset of the universe of $\mathbf{A}$, then the *subuniverse of* $\mathbf{A}$ *generated by* X is defined as follows:

$$\mathrm{Sg}^{\mathbf{A}}(X) = \bigcap \{U \in \mathsf{S}\mathbf{A} \mid X \subseteq U\}. \tag{4.1}$$

To give another exhibition of the efficiency and ease with which we can formalize basic but important mathematical concepts in Lean, we now present a fundamental theorem about subalgebra generation, first in the Informal Language, and then formally in Section 4.4. Notice that the added complexity of the Lean implementation of this theorem is not significant, and the proof seems quite readable (especially when compared to similar proofs in Coq).

Theorem 4.1 ([11, Thm. 1.14])**.** *Let* $\mathbf{A} = \langle A, F^{\mathbf{A}} \rangle$ *be an algebra in the signature* $S = (F, \rho)$ *and let* $X \subseteq A$*. Define, by recursion on* n*, the sets* X_n *as follows:*

$$X_0 = X; \tag{4.2}$$

$$X_{n+1} = X_n \cup \{f\, a \mid f \in F,\ a \in X_n^{\rho f}\}. \tag{4.3}$$

Then $\mathrm{Sg}^{\mathbf{A}}(X) = \bigcup X_n$.

Proof. Let $Y = \bigcup_{n<\omega} X_n$. Clearly $X_n \subseteq Y \subseteq A$, for every $n < \omega$. In particular $X = X_0 \subseteq Y$. Let us show that Y is a subuniverse of $\mathbf{A}$. Let f be a basic k-ary operation and $a \in Y^k$. From the construction of Y, there is an $n < \omega$ such that $\forall i,\ a\, i \in X_n$. From its definition, $f\, a \in X_{n+1} \subseteq Y$. Thus Y is a subuniverse of $\mathbf{A}$ containing X. By (4.1), $\mathrm{Sg}^{\mathbf{A}}(X) \subseteq Y$. For the opposite inclusion, it is enough to check, by induction on n, that $X_n \subseteq \mathrm{Sg}^{\mathbf{A}}(X)$. Well, $X_0 = X \subseteq \mathrm{Sg}^{\mathbf{A}}(X)$ from its definition. Assume that $X_n \subseteq \mathrm{Sg}^{\mathbf{A}}(X)$. If $b \in X_{n+1} - X_n$, then $b = f\, a$ for a basic k-ary operation f and $a \in X_n^k$. But $\forall i,\ a\, i \in \mathrm{Sg}^{\mathbf{A}}(X)$ and since this latter object is a subuniverse, $b \in \mathrm{Sg}^{\mathbf{A}}(X)$ as well. □

4.4. **Subuniverses in Lean (`lean-ualib/subuniverse.lean`).**
The argument in the proof of Theorem 4.1 is of a type that one encounters frequently throughout algebra. It has two parts. First that Y is a subuniverse of $\mathbf{A}$ that contains X. Second that any subuniverse containing X must also contain Y.

We now show how the subalgebra concept and the foregoing argument is formally implemented in Lean.

```
import basic
import data.set
namespace subuniverse
  section
    open set
    parameter {α : Type*}        -- the carrier type
    parameter {S : signature}
    parameter (A : algebra_on S α)
    parameter {I : Type}         -- a collection of indices
    parameter {R : I → set α}    -- an indexed set of sets of type α
    definition F := S.F          -- the type of operation symbols
    definition ρ := S.ρ          -- the operation arity function

    -- Definition of subuniverse
    definition Sub (β : set α) : Prop :=
    ∀ (f : F) (a : ρ f → α), (∀ x, a x ∈ β) → A f a ∈ β

    -- Subuniverse generated by X
    definition Sg (X : set α) : set α := ⋂₀ {U | Sub U ∧ X ⊆ U}
```

Lean syntax for the intersection operation on collections of *sets* is $\bigcap_0$.

Next we need "introduction" and "elimination" rules for arbitrary intersections, plus the useful fact that the intersection of subuniverses is a subuniverse.

```
    -- Intersection introduction rule
    theorem Inter.intro {s : I → set α} :
    ∀ x, (∀ i, x ∈ s i) → (x ∈ ⋂ i, s i) :=
    assume x h t ⟨a, (eq : t = s a)⟩, eq.symm ▸ h a

    -- Intersection elimination rule
    theorem Inter.elim {x : α} (C : I → set α) :
    (x ∈ ⋂ i, C i) →  (∀ i, x ∈ C i) :=
    assume h : x ∈ ⋂ i, C i, by simp at h; apply h

    -- Intersection of subuniverses is a subuniverse
    lemma sub_of_sub_inter_sub (C : I → set α) :
    (∀ i, Sub (C i)) → Sub ⋂i, C i :=
    assume h : ∀ i, Sub (C i), show  Sub ⋂(i, C i), from
      assume (f : F) (a : ρ f → α) (h₁ : ∀ x, a x ∈ ⋂i, C i),
```

```
  show A f a ∈ ⋂i, C i, from
    Inter.intro (A f a)
    (λ j, (h j) f a (λ x, Inter.elim C (h₁ x) j))
```

The next three lemmas show that `Sg X` is the smallest subuniverse containing `X`.

```
-- X is a subset of Sg(X)
lemma subset_X_of_SgX (X : set α) : X ⊆ Sg X :=
assume x (h : x ∈ X),
show x ∈ ⋂₀ {U | Sub U ∧ X ⊆ U}, from
  assume W (h₁ : W ∈ {U | Sub U ∧ X ⊆ U}),
  show x ∈ W, from
    have h₂ : Sub W ∧ X ⊆ W, from h₁,
    h₂.right h

-- A subuniverse that contains X also contains Sg X
lemma sInter_mem {X : set α} (x : α) :
x ∈ Sg X  →  ∀ {R : set α }, Sub R → X ⊆ R → x ∈ R :=
assume (h₁ : x ∈ Sg X) (R : set α)  (h₂ : Sub R) (h₃ : X ⊆ R),
show x ∈ R, from h₁ R (and.intro h₂ h₃)

-- Sg X is a Sub
lemma SgX_is_Sub (X : set α) : Sub (Sg X) :=
assume (f : F) (a : ρ f → α) (h₀ : ∀ i, a i ∈ Sg X),
show A f a ∈ Sg X, from
  assume W (h : Sub W ∧ X ⊆ W), show A f a ∈ W, from
    have h₁ : Sg X ⊆ W, from
      assume r (h₂ : r ∈ Sg X), show r ∈ W, from
        sInter_mem r h₂ h.left h.right,
        have h' : ∀ i, a i ∈ W, from assume i, h₁ (h₀ i),
        (h.left f a h')
```

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 could be much to gain from implementing universal algebra in an expressive type system that offers powerful tools for proving theorems about inductively defined types.

So, we are pleased to present the following inductive type that implements the *subuniverse generated by a set*; cf. the Informal Language definition (4.2), (4.3).

```
inductive Y (X : set α) : set α
| var (x : α) : x ∈ X → Y x
| app (f : F) (a : ρ f → α) : (∀ i, Y (a i)) → Y (A f a)
```

Next we prove that the type `Y X` defines a subuniverse, and that it is, in fact, equal to $\mathrm{Sg}^{\mathbf{A}}(X)$.

```
-- Y X is a subuniverse
lemma Y_is_Sub (X : set α) : Sub (Y X) :=
assume f a (h: ∀ i, Y X (a i)), show Y X (A f a), from
Y.app f a h

-- Y X is the subuniverse generated by X
theorem sg_inductive (X : set α) : Sg X = Y X :=
have h₀ : X ⊆ Y X, from
  assume x (h : x ∈ X),
  show x  ∈ Y X, from Y.var x h,
have h₁ : Sub (Y X), from
  assume f a (h : ∀ x, Y X (a x)),
  show Y X (A f a), from Y.app f a h,
have inc_l : Sg X ⊆ Y X, from
  assume u (h : u ∈ Sg X),
  show u ∈ Y X, from (sInter_mem u) h h₁ h₀,
have inc_r : Y X ⊆ Sg X, from
  assume a (h: a ∈ Y X), show a ∈ Sg X, from
    have h' : a ∈ Y X → a ∈ Sg X, from
      Y.rec
      --base: a = x ∈ X
      ( assume x (h1 : x ∈ X),
        show x ∈ Sg X, from subset_X_of_SgX X h1 )
      --inductive: a = A f b for some b with ∀ i, b i ∈ Sg X
      ( assume f b (h2 : ∀ i, b i ∈ Y X) (h3 : ∀ i, b i ∈ Sg X),
        show A f b ∈ Sg X, from SgX_is_Sub X f b h3 ),
    h' h,
subset.antisymm inc_l inc_r
```

Observe that the last proof proceeds exactly as would a typical informal proof that two sets are equal—prove two subset inclusions and then apply the `subset.antisymm` rule: `A ⊆ B → B ⊆ A → A = B`. We proved `Y X ⊆ Sg X` in this case by induction using the *recursor*, `Y.rec`, which Lean creates for us automatically whenever an inductive type is defined. The Lean keyword `assume` is syntactic sugar for λ; this and other notational conveniences, such as Lean's `have...from` and `show...from` syntax, make it possible to render formal proofs in a very clear and readable way.

4.5. **Terms and Free Algebras.** Fix a signature $S = (F, \rho)$ and let X be a set disjoint from F. The elements of X are called *variables*. For every $n < \omega$, let $F_n = \rho^{-1}\{n\}$ be the set of n-ary operation symbols. By a word on $X \cup F$ we mean a nonempty, finite sequence of members of $X \cup T$. We denote the concatenation of sequences by simple juxtaposition. We define, by recursion on n, the sets T_n of words on $X \cup F$ by

$$T_0 = X \cup F_0;$$
$$T_{n+1} = T_n \cup \{fs \mid f \in F,\ s\colon \rho f \to T_n\}.$$

Define the set of *terms in the signature S over X* by $T_S(X) = \bigcup_{n<\omega} T_n$.

The definition of $T_S(X)$ is recursive, indicating that *the set of terms in a signature can be implemented in Lean as an inductive type*. We will confirm this in the next subsection, but before doing so, we impose an algebraic structure on $T_S(X)$, and then state and prove some basic but important facts about this algebra. These will also be formalized in the next section, giving us another chance to compare Informal Language proofs to their formal Lean counterparts, and to show off inductively defined types in Lean.

If w is a term, let $|w|$ be the least n such that $w \in T_n$, called the *height* of w. The height is a useful index for recursion and induction. Notice that the set $T_S(X)$ is nonempty iff either X or F_0 is nonempty. As long as $T_S(X)$ is nonempty, we can impose upon this set an algebraic structure. For every basic operation symbol $f \in F$ let $f^{\mathbf{T}_S(X)}$ be the operation on $T_S(X)$ that maps each tuple $t : \rho f \to T_S(X)$ to the term ft. We define $\mathbf{T}_S(X)$ to be the algebra with universe $T_S(X)$ and with basic operations $f^{\mathbf{T}_S(X)}$ for each $f \in F$.

The construction of $\mathbf{T}_S(X)$ may seem to be making something out of nothing,but it plays a crucial role in the theory. Indeed, Part (2) of Theorem 4.3 below asserts that $\mathbf{T}_S(X)$ is *universal for S-algebras*. To prove this, we need the following basic lemma, which states that a homomorphism is uniquely determined by its restriction to a generating set.

Lemma 4.2. *Let f and g be homomorphisms from $\mathbf{A}$ to $\mathbf{B}$. If $X \subseteq A$ and X generates $\mathbf{A}$ and $f\big|_X = g\big|_X$, then $f = g$.*

Proof. Suppose the subset $X \subseteq A$ generates $\mathbf{A}$ and suppose $f\big|_X = g\big|_X$. Fix an arbitrary element $a \in A$. We show $f(a) = g(a)$. Since X generates $\mathbf{A}$, there exists a (say, n-ary) term t and a tuple $(x_1, \dots, x_n) \in X^n$ such that $a = t^{\mathbf{A}}(x_1, \dots, x_n)$. Therefore,

$$\begin{aligned} f(a) = f(t^{\mathbf{A}}(x_1, \dots, x_n)) &= t^{\mathbf{B}}(f(x_1), \dots, f(x_n)) \\ &= t^{\mathbf{B}}(g(x_1), \dots, g(x_n)) = g(t^{\mathbf{A}}(x_1, \dots, x_n)) = g(a). \end{aligned}$$

□

Theorem 4.3 ([11, Thm. 4.21])**.** *Let $S = (F, \rho)$ be a signature.*

(1) $\mathbf{T}_S(X)$ is generated by X.

(2) For every S-algebra $\mathbf{A}$ and every function $h\colon X \to A$ there is a unique homomorphism $g\colon \mathbf{T}_S(X) \to \mathbf{A}$ such that $g\big|_X = h$.

Proof. The definition of $\mathbf{T}_S(X)$ exactly parallels the construction in Theorem 4.1. That accounts for (1). For (2), define $g(t)$ by induction on ρt. Suppose $\rho t = 0$. Then $t \in X \cup F$. If $t \in X$ then define $g(t) = h(t)$. For $t \notin X$, $g(t) = t^{\mathbf{A}}$. Note that since $\mathbf{A}$ is an S-algebra and t is a nullary operation symbol, $t^{\mathbf{A}}$ is defined.

For the inductive step, let $|t| = n + 1$. Then $t = f(s_1, \dots, s_k)$ for some $f \in F_k$ and $s_1, \dots, s_k$ each of height at most n. We define $g(t) = f^{\mathbf{A}}(g(s_1), \dots, g(s_k))$. By its very definition, g is a homomorphism. Finally, the uniqueness of g follows from Lemma 4.2. □

4.6. Terms and Free Algebras in Lean (`lean-ualib/free.lean`).

As a second demonstration of inductive types in Lean, we define a type representing the (infinite) collection $\mathbb{T}(X)$ of all terms of a given signature.

```
import basic
section
  parameters {S : signature} (X :Type*)
```

```
  local notation `F` := S.F
  local notation `ρ` := S.ρ

  inductive term
  | var : X → term
  | app (f : F) : (ρ f → term) → term

  def Term : algebra S := ⟨term, term.app⟩
end
```

The set of terms along with the operations $F^{\mathbf{T}} := \{\mathtt{app}\, f \mid f : F\}$ forms an algebra $\mathbf{T}(X) = \langle \mathbb{T}(X), F^{\mathbf{T}} \rangle$ in the signature $S = (F, \rho)$. Suppose $\mathbf{A} = \langle A, F^{\mathbf{A}} \rangle$ is an algebra in the same signature and $h \colon X \to A$ is an arbitrary function. We will show that $h \colon X \to A$ has a unique *extension* (or *lift*) to a homomorphism from $\mathbf{T}(X)$ to $\mathbf{A}$. Since $\mathbf{A}$ and $h \colon X \to A$ are arbitrary, this unique homomorphic lifting property holds universally; accordingly we say that the term algebra $\mathbf{T}(X)$ is *universal* for S-algebras. Before implementing the formal proof of this fact in Lean, let us first define some domain specific syntactic sugar.

```
section
  open term
  parameters {S : signature} (X :Type*) {A : algebra S}
  definition F := S.F          -- operation symbols
  definition ρ := S.ρ          -- arity function
  definition 𝕋 := @Term S      -- term algebra over X
  definition 𝕏 := @var S X     -- generators of the term algebra
```

If `h : X → A` is a function defined on the generators of the term algebra, then the *lift* (or *extension*) of `h` to all of `𝕋(X)` is defined inductively as follows:

```
  definition lift_of (h : X → A) : 𝕋(X) →
  | (var x) := h x
  | (app f a) := (A f) (λ x, lift_of (a x))
```

To prove that the term algebra is absolutely free, we show that the lift of an arbitrary function `h : X → A` is a homomorphism and that this lift is unique.

```
  -- The lift is a homomorphism.
  lemma lift_is_hom (h : X → A) : homomorphic (lift_of h) :=
  λ f a, show lift_of h (app f a) = A f (lift_of h ∘ a), from rfl

  -- The lift is unique.
  lemma lift_is_unique : ∀ {h h' : 𝕋(X) → A},
  homomorphic h → homomorphic h' → h ∘ 𝕏 = h' ∘ 𝕏 → h = h' :=
  assume (h h' : 𝕋(X) → A) (h₁ : homomorphic h)
    (h₂ : homomorphic h')(h₃ : h ∘ 𝕏 = h' ∘ 𝕏),
    show h = h', from
      have h₀ : ∀ t : 𝕋(X), h t = h' t, from
        assume t : 𝕋(X),
        begin
```

```
        induction t with t f a ih₁ ,
        show h 𝕏( t) = h' 𝕏( t),
        { apply congr_fun h₃ t },

        show h (app f a) = h' (app f a),
        { have ih₂  : h ∘ a = h' ∘ a, from funext ih₁,
          calc h (app f a) = A f (h ∘ a) : h₁ f a
                       ... = A f (h' ∘ a) : congr_arg (A f) ih₂
                       ... = h' (app f a) : (h₂ f a).symm }
      end,
    funext h₀
end
```

5. Summary of Project Stages

In the introduction, we set out the goals of the project in broad strokes. Here we describe four stages of concrete activities that map out a plan for achieving our goals.

Stage 1. **lean-ualib.** Implement in Lean *formal statements and proofs* of the definitions and theorems that constitute the *core* of our field, universal algebra. We call this the *Lean Universal Algebra Library*, abbreviated `lean-ualib`.[7]

Stage 2. **domain-specific automation.** Develop *proof tactics* in Lean that carry out, in a highly automated way, the most common arguments and proof techniques of universal algebra; that is, make the *proof idioms* of our field readily available in Lean.

Stage 3. **lean-uaailib.** *Develop search and artificial intelligence tools to accelerate growth of the library either by guided user input or by allowing the library to grow itself.*

Stage 4. **Algebras, Categories and Types: with computer-aided proofs.** Create a textbook that presents (a) the theory that is formalized in `lean-ualib`, (b) a comprehensive *reference manual* for the library, and (c) *examples and instructions for working mathematicians.*

Although Stage 1 alone may seem like a massive undertaking, we will start with the basic results of the theory and formalize the lemmas and theorems that are most commonly used in our field to prove deeper results. In the process, we will

(1) figure out how to *automate proof search* in the specific domain of universal algebra;
(2) create libraries for operation clones, free algebras, homomorphisms, and congruences, and implement DSA that make these libraries relatively easy to use;
(3) explore Lean's *metaprogramming framework* and develop techniques and tools that help mathematicians access this framework so that turning their reasoning methods into automated tactics is straight-forward;
(4) integrate computer algebra systems like the UACalc [23] and GAP [24].

Then we will formalize deeper results, prioritizing definitions, lemmas, and theorems according to how widely they are used in our field. Having already formalized the *free algebra in an arbitrary signature* and implemented a formal proof that it is *absolutely free*,[8] our next target is *Birkhoff's HSP Theorem*. With these and other foundational results

[7] Work on the `lean-ualib` has already begun; the permanent residence of our open source repository is at https://github.com/UniversalAlgebra/lean-ualib.

[8] *ibid.* lean-ualib/tree/master/src/free.lean.

established, we will have a small mathematical arsenal at our disposal that we can exploit when formalizing proofs of deeper theorems. Thus, the library will expand until we have formalized the core of our subject.

With this strategy, we expect the library development will begin at a moderate pace but will quickly accelerate. To ensure that we don't get off to a slow start or risk "reinventing the wheel," besides developing domain-specific automation tools, we will we will also build upon existing Lean libraries, such as the *Lean Mathematical Components Library* [3], so we don't have to formalize everything from scratch. These strategies will combine to substantially reduce and limit the amount of work required to complete the first stage.

The ultimate goal is to advance the state of Lean and its mathematical libraries to the point at which it becomes *a proof assistant that helps working mathematicians*, by making them both more productive and more confident in their results.

6. Conclusion

This project also 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.

Indeed, algebraic structures are most naturally specified using typed predicate calculus expressions and explicitly requires variable-dependent sets, just like the specification language of Martin-Löf's Theory of Types [35]. In order to support this feature, programming constructs should be able to return set or type values, hence *dependent types*. This accommodates a stronger form of function definition than is available in most programming languages; specifically, it allows the type of a result of a function application to depend on a formal parameter, the value (not merely the type) of the input.

For formalizing long complex mathematical arguments Lean relies on *computational reflection*. Dependent types make it possible for data, functions, and *potential* computations to appear inside types. Standard mathematical practice is to interpret and expand these objects, replacing a constant by its definition, instantiating formal parameters, etc. In

contrast, Lean supports this through typing rules that lets such computation happen transparently, and arbitrary long computations can be eliminated from a proof. Consequently, entirely new ways of proving certain calculational results become available to us. With these tools at our disposal, we can quickly and efficiently formalize many results, beyond merely certifying their correctness. We expect that the proposed deliverables of this project will have a substantial positive impact the pace of mathematical discoveries in our field.

This effort requires a careful reconsideration of how to express the informal logical foundations of our subject, since this determines how smoothly we can implement the core mathematical theory in the formal language of the proof assistant, and this will in turn influence how accessible are the resulting libraries.

We are building tools that will make the common, informal *mathematical idioms* available in Lean, which will make this software more accessible and make it easier to discover new theorems, verify existing theorems, and test conjectures, all using a language that is relatively close to the Informal Language commonly used by those of us working in universal algebra and related fields.

Our goal is to formally prove theorems and do research mathematics while at the same time address the main usability roadblocks that stand in the way of widespread adoption of proof assistant technology. The theorems we decide to formalize and implement in the software are selected, together with our collaborators, to guide the development of theorem libraries and verified tools and methods (or *proof tactics*) for doing modern research in mathematics. The main objective of this project (and others like it [3, 13]) is to develop and codify the **practical formal foundations of informal mathematics** in which most modern research is carried out.

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.

Acknowledgements

I am greatly indebted to Jasmin Blanchette who shared with me the original proposal for his *Lean Forward Project* [13]. That had a significant impact on me, and inspired me to focus a significant part of my research activities on Lean development. In short, the idea and design of the present document owes a great deal to Jasmin's *Lean Forward Project Proposal* (cf. [13]).

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 03B40 Combinatory logic and lambda-calculus 03F07 Structure of proofs 03F55 Intuitionistic mathematics 08B05 Equational logic, Malcev conditions 08B20 Free algebras 68N15 Programming languages 68N18 Functional programming and lambda calculus 68T15 Theorem proving (deduction, resolution, etc.) 68W30 Symbolic computation and algebraic computation

A.1. **Project Personnel.**

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)

Appendix B. complementary projects

B.1. **Lean Forward.** The Netherlands Organization for Scientific Research recently awarded a five-year grant to Jasmin Blanchette for his *Lean Forward* project, which is similar but complementary to our project. First, Dr. Blanchette is a computer scientist enlisting the support of mathematicians, whereas we are mathematicians enlisting support of computer scientists. Moreover, Jasmin works primarily with number theorists, whereas we are focused on universal algebra. We have had fruitful contact with Jasmin at the *Big Proof* workshop last summer at Cambridge University's Isaac Newton Institute, and will attend the inaugural meeting of the Lean Forward project in January 2019 in Amsterdam. We look forward to productive future collaborations with the Lean Forward scientists.

B.2. **Existing libraries for universal algebra and lattice theory.** There has been prior work (mostly by computer scientists) on formalizing certain parts of universal algebra. The first significant example of this was initiated in Venanzio Capretta's phd thesis (see [14]) presenting the fundamentals of universal algebra using intuitionistic logic so that the resulting theorems have computational meaning (via the propositions-as-types/proofs-as-programs correspondence explained earlier). Capretta's formalizations were done in Coq.

Another more recent development in Coq is the `mathclasses` library, initiated by Bas Spitters and Eelis van der Weegen [42].

B.3. **Formal proof archives.** The Archive of Formal Proofs is a collection of proof libraries, examples, and larger scientific developments, mechanically checked in the theorem prover Isabelle. It is organized in the way of a scientific journal, is indexed by dblp and has an ISSN: 2150-914x. Submissions are refereed and companion AFP submissions to conference and journal publications are encouraged. A development version of the archive is available as well.

References

[1] J. Adámek, J. Rosický, and E. M. Vitale. *Algebraic theories*, volume 184 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 2011. A categorical introduction to general algebra, With a foreword by F. W. Lawvere.

[2] Jeremy Avigad, Mario Carneiro, Leonardo de Moura, Johannes Hölzl, and Sebastian Ullrich. The lean theorem proving language. GitHub.com, 2018. URL: `https://leanprover.github.io/`.

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

[4] Gilles Barthe, Venanzio Capretta, and Olivier Pons. Setoids in type theory. *J. Funct. Programming*, 13(2):261–293, 2003. Special issue on "Logical frameworks and metalanguages". URL: `http://dx.doi.org/10.1017/S0956796802004501`, `doi:10.1017/S0956796802004501`.

[5] Andrej Bauer. Five stages of accepting constructive mathematics. *Bull. Amer. Math. Soc. (N.S.)*, 54(3):481–498, 2017. `doi:10.1090/bull/1556`.

[6] Andrej Bauer. On fixed-point theorems in synthetic computability. *Tbilisi Math. J.*, 10(3):167–181, 2017. `doi:10.1515/tmj-2017-0107`.

[7] 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`.

[8] Andrej Bauer and Jens Blanck. Canonical effective subalgebras of classical algebras as constructive metric completions. *J.UCS*, 16(18):2496–2522, 2010.

[9] Andrej Bauer, Gordon D. Plotkin, and Dana S. Scott. Cartesian closed categories of separable Scott domains. *Theoret. Comput. Sci.*, 546:17–29, 2014. `doi:10.1016/j.tcs.2014.02.042`.

[10] Mike Behrisch, Sebastian Kerkhoff, and John Power. Category theoretic understandings of universal algebra and its dual: monads and Lawvere theories, comonads and what? In *Proceedings of the 28th*

Conference on the Mathematical Foundations of Programming Semantics (MFPS XXVIII), volume 286 of *Electron. Notes Theor. Comput. Sci.*, pages 5–16. Elsevier Sci. B. V., Amsterdam, 2012. doi:10.1016/j.entcs.2012.08.002.

[11] Clifford Bergman. *Universal algebra*, volume 301 of *Pure and Applied Mathematics (Boca Raton)*. CRC Press, Boca Raton, FL, 2012. Fundamentals and selected topics.

[12] 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.

[13] J. C. Blanchette. Lean forward: Usable computer-checked proofs and computations for number theorists. GitHub.com, 2018. URL: https://lean-forward.github.io/.

[14] Venanzio Capretta. Universal algebra in type theory. In *Theorem proving in higher order logics (Nice, 1999)*, volume 1690 of *Lecture Notes in Comput. Sci.*, pages 131–148. Springer, Berlin, 1999. URL: http://dx.doi.org/10.1007/3-540-48256-3_10, doi:10.1007/3-540-48256-3_10.

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

[16] Thierry Coquand and Gérard Huet. The calculus of constructions. *Inform. and Comput.*, 76(2-3):95–120, 1988. URL: http://dx.doi.org/10.1016/0890-5401(88)90005-3, doi:10.1016/0890-5401(88)90005-3.

[17] 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.

[18] 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/.

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

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

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

[22] Eric Finster. Higher algebra in type theory. GitHub.com, 2018. URL: https://github.com/ericfinster/higher-alg.

[23] Ralph Freese, Emil Kiss, and Matthew Valeriote. Universal Algebra Calculator, 2008. URL: http://www.uacalc.org.

[24] The GAP Group. *GAP – Groups, Algorithms, and Programming, Ver. 4.4.12*, 2008. URL: http://www.gap-system.org.

[25] David Gepner, Rune Haugseng, and Joachim Kock. ∞-operads as analytic monads. arXiv, 2017. URL: https://arxiv.org/abs/1712.06469.

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

[27] 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.

[28] 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.

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

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

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

[32] 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.

[33] Martin Hyland and John Power. The category theoretic understanding of universal algebra: Lawvere theories and monads. In *Computation, meaning, and logic: articles dedicated to Gordon Plotkin*,

volume 172 of *Electron. Notes Theor. Comput. Sci.*, pages 437–458. Elsevier Sci. B. V., Amsterdam, 2007. URL: https://doi-org.colorado.idm.oclc.org/10.1016/j.entcs.2007.02.019, doi:10.1016/j.entcs.2007.02.019.

[34] Ursula Martin, Larry Paulson, and Andrew Pitts. Computer-aided mathematical proof. In *Big Proof Workshop*, Cambridge University, 2017. Isaac Newton Institute. URL: https://www.newton.ac.uk/event/bprw01.

[35] Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory. Lecture Notes*. Bibliopolis, Naples, 1984. Notes by Giovanni Sambin.

[36] Karl Meinke. Universal algebra in higher types. *Theoret. Comput. Sci.*, 100(2):385–417, 1992. doi:10.1016/0304-3975(92)90310-C.

[37] B. C. Pierce. Lambda, the ultimate TA: Using a proof assistant to teach programming language foundations. In G. Hutton and A. P. Tolmach, editors, *International Conference on Functional Programming*, pages 121–122. ACM, :2009.

[38] Don Pigozzi and Antonino Salibra. Lambda abstraction algebras: representation theorems. *Theoret. Comput. Sci.*, 140(1):5–52, 1995. Selected papers of AMAST '93 (Enschede, 1993). URL: http://dx.doi.org/10.1016/0304-3975(94)00203-U, doi:10.1016/0304-3975(94)00203-U.

[39] M. Pollet and Manfred Kerber. Informal and formal representations in mathematics. *Studies in Logic, Grammar and Rhetoric: From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, 10(23):75–94, 1 2007.

[40] Dana S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoret. Comput. Sci.*, 121(1-2):411–440, 1993. A collection of contributions in honour of Corrado Böhm on the occasion of his 70th birthday. URL: http://dx.doi.org/10.1016/0304-3975(93)90095-B, doi:10.1016/0304-3975(93)90095-B.

[41] Daniel Selsam and Leonardo de Moura. Congruence closure in intensional type theory. In *Automated reasoning*, volume 9706 of *Lecture Notes in Comput. Sci.*, pages 99–115. Springer, [Cham], 2016. doi:10.1007/978-3-319-40229-1_8.

[42] Bas Spitters and Eelis van der Weegen. Developing the algebraic hierarchy with type classes in Coq. In *Interactive theorem proving*, volume 6172 of *Lecture Notes in Comput. Sci.*, pages 490–493. Springer, Berlin, 2010. URL: http://dx.doi.org/10.1007/978-3-642-14052-5_35, doi:10.1007/978-3-642-14052-5_35.

[43] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity*. MIT Press, Cambridge, MA, USA, 1990.

[44] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science (Vol. B): Formal Models and Semantics*. MIT Press, Cambridge, MA, USA, 1990.

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