Formal Mathematics for the Masses

William M. Farmer*

10 May 2022

Abstract

The campaign to transform traditional mathematical practice into a formal discipline appears to have been a great success. Several sophisticated proof assistants have been developed, a great deal of mathematical knowledge has been formalized, a growing number of researchers in computing and mathematics are now using proof assistants to check the theorems they prove, and a new area of computing called formal methods has been established in which formal mathematics is used in the development of hardware and software. However, the campaign has largely been a failure when viewed from a broader perspective. It has had almost no impact on mathematical practice. Far less than 1% of all mathematics practitioners have ever used a formal logic or a proof assistant in their work. We propose an alternative approach to formal mathematics that is characterized by (1) proofs are written in a traditional (informal) style, (2) the underlying logic is as close to mathematical practice as possible, (3) mathematics is organized in accordance with the little theories method as a theory graph, and (4) supporting software systems are easy to build and use. We are pursuing a research program called *Formal Mathematics for the Masses* to make formal mathematics more useful, accessible, and natural to a wider range of mathematics practitioners by implementing this alternative approach.

1 The Campaign for Formal Mathematics

Formal mathematics is mathematics done within a formal logic consisting of (1) a formal language L with a precise syntax, (2) a precise semantics for L with a notion of logical consequence, and (3) a formal proof system

*Address: Department of Computing and Software, McMaster University, 1280 Main Street West, Hamilton, Ontario L8S 4K1, Canada. E-mail: wmfarmer@mcmaster.ca.

for proving statements written in L. It offers mathematics practitioners — mathematicians, computer scientists, engineers, and scientists — two major benefits over traditional mathematics [2, 21]. First, mathematics can be done with much greater rigor in a formal logic. Mathematical ideas are expressed as unambiguous statements in the formal language. Results about these ideas are obtained using the notion of logical consequence. And proofs certifying these results are constructed in the formal proof system. Second, the study, discovery, communication, and certification of mathematics can be done with the aid of software. In particular, software can be used to write proofs in the formal proof system and then mechanically check their correctness.

Over the last 40 years, a group of computer scientists and logicians (and a few mathematicians) have participated in a loosely coordinated campaign to transform traditional mathematical practice into a formal discipline in which mathematics practitioners do mathematics with the help of *proof assistants* [15], logic-based software systems for developing and certifying formal proofs. Perhaps the best expression of their vision is the 1994 *QED Manifesto* [33]. They ultimately would like to see a mathematics world in which essentially all mathematical knowledge is expressed in formal languages and all mathematical proofs are written in formal proof systems and mechanically checked by software. They are motivated by the belief that formalizing mathematics will enable more people to do more mathematics and do it with greater ease and greater assurance of being correct.

This campaign appears to have been a great success. Several highly sophisticated proof assistants have been developed, a great deal of mathematical knowledge has been formalized, and a growing number of researchers in computing and mathematics are now using proof assistants to check the theorems they prove. Moreover, the campaign has had a significant impact on hardware and software development, spawning an area of computing called *formal methods* [16, 35, 38] in which formal mathematics is used to specify and verify software and hardware systems.

However, the campaign has largely been a failure when viewed from a broader perspective. It has had almost no impact on mathematical practice [37]. Far less than 1% of all mathematics practitioners have ever used a formal logic or a proof assistant in their work. This is in sharp contrast with the fact that nearly every mathematics practitioner uses programming languages, computer algebra systems, or statistical software on a daily basis. Formal mathematics certainly has the potential to greatly improve mathematical practice, especially in computing and engineering. So why, after decades of effort, has the campaign to formalize mathematics failed?

2 Why the Campaign has Failed

To understand why the campaign has failed, we need to consider two subjects: logic education and mathematical software development.

Formal mathematics requires an understanding of how logic can be used in practice to reason about mathematics and, in particular, mathematical structures. In a typical university logic course, students are introduced to theory-oriented logics, like first-order logic and set theory, that are designed to be studied, but not actually used. These logics are exceedingly cumbersome to employ in practice, e.g., to specify and reason about functions. Students are very rarely exposed to practice-oriented logics, like simple type theory [10] and other higher-order logics, that are actually suitable for doing mathematics. Moreover, most of these higher-order logics are far removed from mathematical practice and are clumsy when dealing with notational conventions, undefined expressions, partial functions, definite descriptions, and subtypes. As a result, almost all mathematics practitioners lack the background and experience in the practical application of logic needed to read and write formal mathematics and to utilize logic-based software systems like proof assistants. They also find the logics available to them to be unnatural and burdensome to use.

Proof assistants are the dominant kind of software system for doing formal mathematics. They are heavily focused on *formally certifying mathematical results*. Tremendous progress has been made in this direction, and proof assistants have been successfully used to check complex proofs of deep mathematical theorems [17, 20]. However, the great majority of mathematics practitioners — including mathematicians — are much more interested in *communicating mathematical ideas* than certifying mathematical results formally. For example, an engineer who wants to carefully specify a system is likely to be much more interested in communicating what the system is intended to do than in proving that all the details are correct.

Formal certification of results is a considerably more difficult task than communication of ideas. As a consequence, proofs assistants are very complex and notoriously difficult to learn how to use. For someone who just wants to express ideas in a formal language with a precise syntax and semantics, a proof assistant impedes more that it facilitates. This is especially true for proof assistants, like Agda [29, 30], Coq [4], and Lean [5], based on dependent type theories. With complex type systems and deductive systems based on constructive reasoning, dependent type theories are very far afield from what mathematics practitioners are familiar with and what they need. Thus proof assistants, in which certification is first and communi-

cation is second, do not adequately serve the needs of most mathematics practitioners.

In summary, there are four reasons for the failure of the campaign to formalize mathematics. First, very few mathematics practitioners have the background in the practical application of logic needed to read and write formal mathematics and to utilize proof assistants. Second, the formal logics and proof assistants used for formal mathematics are, as a rule, very far removed from mathematical practice. Third, proof assistants are very complex and notoriously difficult to learn how to use. And fourth, and most importantly, proof assistants have been designed primarily for formally certifying mathematical results and only secondarily for communicating mathematical ideas, but mathematics practitioners are usually much more interested in the latter than the former.

3 An Alternative Approach

We propose an alternative approach to formal mathematics that is characterized by:

1. Proofs are written in a traditional (informal) style and optimized for communication.

2. The underlying formal logic is as close to mathematical practice as possible.

3. Mathematical knowledge is organized in accordance with the *little theories method* [12] as a *theory graph* [25, 26] consisting of axiomatic theories as nodes and theory morphisms (meaning-preserving mappings from the formulas of one theory to the formulas of another) as directed edges.

4. Supporting software systems are easy to build and use.

The first characteristic, that proofs are traditional instead of formal, is the key. Traditional proofs are easier to read and write than formal proofs and better suited for communicating the ideas behind proofs. Supporting software systems do not need a facility for developing and checking formal proofs. As a result, they can be much simpler and thus easier to both implement and learn how to use. Also, there is no harm in having the underlying logic include features that make formal proof more difficult since there are no formal proofs. This approach should not be construed as "lightweight"

formal mathematics [23, 39]: Everything is done in a formal logic except for the proofs which are written in a traditional style that facilitates communication. Certification is still important, but it is done, as in standard mathematical practice, via traditional proof.

This approach is certainly not meant to be a replacement for the standard approach to formal mathematics in which everything is proved formally using a proof assistant. It is intended to be a complementary approach for those mathematics practitioners who lack either the background or the need to do formal proofs. It is also quite possible that someone doing mathematics with this approach might want to certify some results — for example, results that require a high assurance of correctness or are tricky to verify in a traditional manner — by producing formal proofs of them using a proof assistant.

4 Formal Mathematics for the Masses

We are pursuing a long-term research program called *Formal Mathematics for the Masses* to make formal mathematics more useful, accessible, and natural to a wider range of mathematics practitioners by implementing this alternative approach to formal mathematics. The research program has two phases. The first is the development of a practical logic that is very close to mathematical practice and educational material for learning the logic, and the second is the development of software to support the use of the logic.

The first phase of the research is largely complete. We have developed a practice-oriented logic called *Alonzo* that is based on Alonzo Church's formulation of simple type theory [1, 3] and that admits undefined expressions in accordance with what we call the *traditional approach to undefinedness* [9]. The development includes the syntax and semantics of Alonzo, notational conventions and definitions used for constructing Alonzo types and expressions, a proof system for Alonzo that is sound and complete, and mathematical knowledge modules (such as theories and theory morphisms) needed to build theory graphs in Alonzo. The logic is exceptionally well suited for specifying and reasoning about mathematical structures. With its support for undefined expressions (resulting from partial functions and definite descriptions) and an extensive set of notational conventions and definitions, Alonzo is very close to mathematical practice. We have also developed a set of LaTeX macros and environments for constructing Alonzo objects such as types, expressions, theories, and theories morphisms.

We have completed about 90% of a textbook entitled *Simple Type The-*

ory: A Practical Logic for Expressing and Reasoning about Mathematical Ideas [11] that presents simple type theory using Alonzo. *Simple Type Theory* is intended to give students the background in logic they need for doing formal mathematics plus a practical logic in which to work. The target audience for the textbook is graduate and advanced undergraduate students as well as mathematics practitioners who need a logic they can employ in practice to specify and reason about complex systems. Unlike the theory-oriented logics presented in traditional logic textbooks, Alonzo is offered as a logic for actually doing mathematics in practice.

The second phase of program is to develop of the following three software systems:

1. An *expression assistant* for producing internal representations of Alonzo expressions, providing type checking, presenting the expressions in various formats including LaTeX, and enabling notational definitions to be deployed and unfolded. In short, the expression assistant will be an interactive development environment (IDE) for the Alonzo language.

2. A *theory assistant* for constructing theory graphs by defining theories in Alonzo, developing them by making definitions and stating and proving theorems, and then interconnecting them with theory morphisms. In addition, the system will provide various kinds of mechanical support such as expression simplification, symbolic computation, meta-level reasoning, and transportation of definitions and theorems from one theory to another via theory morphisms. Unlike a proof assistant, the theory assistant will not include a facility for producing formal proofs. As a result, it will be much easier to use — as well as design and implement — than a proof assistant.

3. A *theory exporter* for exporting theories written in Alonzo to proof assistants based on simple type theory like HOL [18], HOL Light [22], Isabelle [32], and PVS [31] or to other proof assistants that support classical reasoning such as Lean [5], Metamath [27] and Mizar [28]. This will enable the traditional proofs in Alonzo theories, if desired, to be reformulated as formal, machine-checked proofs when correctness is required at the highest level of assurance.

The software produced by the research program will support three styles of doing mathematics. The first style is to write all the expressions in a mathematical document in Alonzo using the expression assistant. The second style is to use the theory assistant to produce theory graphs in Alonzo

that are fully formal except that the proofs are traditional. And the third style is to produce a theory graph using the theory assistant and then formally certify selected proofs by exporting their theories to a proof assistant. The long-term goal of the research program is to develop an integrated, well-documented software system for doing mathematics that is fully formal except for proofs.

5 Related Work

In recognition that the formalization campaign has failed to reach the great majority of the mathematics community, alternative approaches to formalize mathematics have been proposed. Two of the most notable lie in the space between traditional mathematics and fully certified formal mathematics. The first is Tom Hales' *formal abstracts in mathematics* project [14, 19] in which proof assistants are used to the create *formal abstracts*, which are formal presentations of mathematical theorems without formal proofs. The second is Michael Kohlhase's *flexiformal mathematics* [24] initiative in which mathematics is a mixture of traditional and formal mathematics and proofs can be either traditional or formal. These approaches, unfortunately, have not made much of an inroad into the 99% of mathematics practitioners who have never employed formal mathematics.

Our approach is similar to both of these approaches, but there are important differences. The formal abstracts approach seeks to formalize *collections of theorems* without proofs, either traditional or formal, using proof assistants, while we seek to formalize *theory graphs* with traditional proofs using a theory assistant, a much simpler kind of system. The objective of the flexiformal mathematics approach is to give the user the flexibility to produce mathematics with varying degrees of formality. In contrast, our approach is to produce mathematics that is fully formal except for proofs. And, if desired, some proofs can be made formal in a flexiformal fashion by exporting the relevant part of the work to a proof assistant.

Several proofs assistants in use today, including HOL4, HOL Light, Isabelle, and PVS, are based on simple type theory like Alonzo. However, the logics for these proof assistants, as well as most other proof assistants, do not admit undefined expressions even though undefined expressions are commonplace in mathematical practice. The most notable exception is the IMPS proof assistant [13] whose logic LUTINS [6, 7, 8] is a version of Church's type theory with undefined expressions, partial functions, definite description, and subtypes. Alonzo is closely related to LUTINS, but its syntax and

semantics are simpler and it employs many more notational definitions than LUTINS does. IMPS has demonstrated that a logic like LUTINS or Alonzo can be effectively implemented and that formal mathematics can stay closer to mathematical practice by handling undefinedness directly.

IMPS introduced the idea of organizing mathematical knowledge in a proof assistant as a theory graph [8, 12]. Now most proof assistants utilize theory morphisms or theory graphs in some fashion. The Specware software development system [36] and the MMT knowledge representation framework [34] are two notable systems in which theory graphs play a central role.

6 Conclusion

There are millions of mathematics practitioners — and billions if mathematics students are included — yet only a minuscule portion of these do mathematics with the help of formal logics and proof assistants. By developing and disseminating a practice-oriented logic with software to support its use, the Formal Mathematics for the Masses research program will make formal mathematics useful and accessible to a much larger portion of mathematics practitioners than previous approaches. The program will also provide a stepping stone for helping mathematics practitioners cross the void between traditional mathematics and fully formal mathematics. The development of an integrated system for doing formal mathematics will accelerate the campaign to formalize mathematics and will strengthen the research being pursued in both mechanized mathematics and formal methods.

References

[1] P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof, Second Edition.* Kluwer, 2002.

[2] J. Avigad. The mechanization of mathematics. *Notices of the American Mathematical Society*, 65:681–690, 2018.

[3] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[4] *Reference Manual for the Coq Proof Assistant, Version 8.15.1*, 2022. Accessed on 2022-May-10.

[5] L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer. The Lean theorem prover (system description). In A. P. Felty and A. Middeldorp, editors, *Automated Deduction — CADE-25*, volume 9195, pages 378–388, 2015.

[6] W. M. Farmer. A partial functions version of Church's simple theory of types. *Journal of Symbolic Logic*, 55:1269–91, 1990.

[7] W. M. Farmer. A simple type theory with partial functions and subtypes. *Annals of Pure and Applied Logic*, 64:211–240, 1993.

[8] W. M. Farmer. Theory interpretation in simple type theory. In J. Heering, K. Meinke, B. Möller, and T. Nipkow, editors, *Higher-Order Algebra, Logic, and Term Rewriting*, volume 816 of *Lecture Notes in Computer Science*, pages 96–123. Springer, 1994.

[9] W. M. Farmer. Formalizing undefinedness arising in calculus. In D. Basin and M. Rusinowitch, editors, *Automated Reasoning—IJCAR 2004*, volume 3097 of *Lecture Notes in Computer Science*, pages 475–489. Springer, 2004.

[10] W. M. Farmer. The seven virtues of simple type theory. *Journal of Applied Logic*, 6:267–286, 2008.

[11] W. M. Farmer. *Simple Type Theory: A Practical Logic for Expressing and Reasoning about Mathematical Ideas*. Unpublished textbook, 90% completed, 255 pp., 2022.

[12] W. M. Farmer, J. D. Guttman, and F. J. Thayer. Little theories. In D. Kapur, editor, *Automated Deduction—CADE-11*, volume 607 of *Lecture Notes in Computer Science*, pages 567–581. Springer, 1992.

[13] W. M. Farmer, J. D. Guttman, and F. J. Thayer. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11:213–248, 1993.

[14] Formal Abstracts. https://formalabstracts.github.io, 2022. Accessed on 2022-May-10.

[15] H. Geuvers. Proof assistants: History, idea and future. *Sadhana*, 34:3–25, 2009.

[16] S. Gnesi and T. Margaria. *Formal Methods for Industrial Critical Systems: A Survey of Applications*. Wiley, 2013.

[17] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Le Roux, A. Mahboubi, R. O'Connor, S. Ould Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry. A machine-checked proof of the odd order theorem. In *Interactive Theorem Proving (ITP 2013)*, volume 7998 of *Lecture Notes in Computer Science*, pages 163–179. Springer, 2013.

[18] M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[19] T. Hales. Formal abstracts in mathematics. In F. Rabe, W. M. Farmer, G. O. Passmore, and Y. Abdou, editors, *Intelligent Computer Mathematics*, volume 11006 of *Lecture Notes in Computer Science*, page xiii. Springer, 2018.

[20] T. Hales et al. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 5, 2017.

[21] J. Harrison. Formalized mathematics. Technical Report 36, Turku Centre for Computer Science (TUCS), 1996.

[22] J. Harrison. HOL Light: An overview. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics*, volume 5674 of *Lecture Notes in Computer Science*, pages 60–66. Springer, 2009.

[23] D. Jackson. Lightweight formal methods. In J. N. Oliveira and P. Zave, editors, *FME 2001: Formal Methods for Increasing Software Productivity*, volume 2021 of *Lecture Notes in Computer Science*, page 1. Springer, 2017.

[24] M. Kohlhase. The Flexiformalist Manifesto. In A. Voronkov, V. Negru, T. Ida, T. Jebelean, D. Petcu, S. M. Watt, and D. Zaharie, editors, *14th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2012)*, pages 30–35. IEEE Press, 2012.

[25] M. Kohlhase. Mathematical knowledge management: Transcending the one-brain-barrier with theory graphs. *European Mathematical Society (EMS) Newsletter*, 92:22—-27, June 2014.

[26] M. Kohlhase, F. Rabe, and V. Zholudev. Towards MKM in the large: Modular representation and scalable software architecture. In S. Autexier, J. Calmet, D. Delahaye, P. D. F. Ion, L. Rideau, R. Rioboo, and

A. P. Sexton, editors, *Intelligent Computer Mathematics*, volume 6167 of *Lecture Notes in Computer Science*, pages 370–384. Springer, 2010.

[27] N. D. Megill and D. A. Wheeler. *Metamath: A Computer Language for Mathematical Proofs*. Lulu Press, 2019. `http://us.metamath.org/downloads/metamath.pdf`.

[28] A. Naumowicz and A. Korniłowicz. A brief overview of Mizar. In *Theorem Proving in Higher Order Logics*, volume 5674 of *Lecture Notes in Computer Science*, pages 67–72. Springer, 2009.

[29] U. Norell. *Towards a Practical Programming Language based on Dependent Type Theory*. PhD thesis, Chalmers University of Technology, 2007.

[30] U. Norell. Dependently typed programming in Agda. In A. Kennedy and A. Ahmed, editors, *Proceedings of TLDI'09*, pages 1–2. ACM, 2009.

[31] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T. A. Henzinger, editors, *Computer Aided Verification: 8th International Conference, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414. Springer, 1996.

[32] L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.

[33] The QED Manifesto. In A. Bundy, editor, *Automated Deduction—CADE-12*, volume 814 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 1994.

[34] F. Rabe and M. Kohlhase. A scalable module system. *Information and Computation*, 230:1–54, 2013.

[35] T. Ringer et al. *QED at Large: A Survey of Engineering for Formally Verified Software*. Now Publishers, 2019.

[36] D. R. Smith. Mechanizing the development of software. In M. Broy and R. Steinbrueggen, editors, *Calculational System Design*, pages 251–292. IOS Press, 1999.

[37] F. Wiedijk. The QED manifesto revisited. In R. Matuszwski and A. Zalewska, editors, *From Insight to Proof, Festschrift in Honour of Andrzej Trybulec*, pages 121–133. University of Białystok, 2007.

[38] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald. Formal methods: Practice and experience. *ACM Computing Surveys*, 41:1–36, 2009.

[39] A. Zamansky, M. Spichkova, G. Rodríguez-Navas, P. Herrmann, and J. Blech. Towards classification of lightweight formal methods. In E. Damiani, G. Spanoudakis, and L. Maciaszek, editors, *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 305–313. SciTePress, 2018.