How to Build a Formal Mathematical Library in which Communication is Prioritized over Certification: An Extended Abstract

William M. Farmer

McMaster University, Hamilton, Ontario, Canada
wmfarmer@mcmaster.ca
https://imps.mcmaster.ca/wmfarmer

9 June 2026

Abstract. *Formal mathematics* is mathematics done within the framework of a formal logic. The *standard approach to formal mathematics*, in which mathematics is done with a proof assistant and all details are formally proved and mechanically checked, focuses on *certification*. In contrast, the *free approach to formal mathematics*, an alternative to the standard approach that is *free* of the obligation to formally prove and mechanically check all details using a proof assistant, focuses on *communication* and *accessibility*. A *formal mathematical library (FML)* is a repository of mathematical knowledge expressed within a formal logic and organized as a network of interconnected axiomatic theories. This talk presents the free approach, describes how a communication-oriented FML can be built using the free approach, and ends by encouraging the mathematics software community to develop the logics and software needed to build communication-oriented FMLs.

1 Introduction

If we are willing to oversimplify a bit, we can say that mathematics is performed by assuming a knowledge base of mathematical concepts and facts and then adding new concepts and facts to it. The knowledge base is often not explicitly presented and the new concepts and facts are often not precisely expressed. The situation is much better in *formal mathematics*, that is, mathematics done within the framework of a formal logic.

For this discussion, a *formal logic* (*logic* for short) is a family of languages such that:

1. The languages of the logic have a *precise common syntax*.
2. The languages of the logic have a *precise common semantics with a notion of logical consequence*.
3. There is a sound *formal proof system* for the logic in which proofs can be syntactically constructed.

Although the languages of the logic share a common syntax, the expressions of a language can be presented, if desired, using multiple notations including symbol-oriented notations like those found in logic textbooks and controlled natural languages like the input language for the Naproche proof assistant [21].

In formal mathematics, the knowledge base is a *formal mathematical library (FML)* — a repository of mathematical knowledge expressed within a logic and organized as a network of interconnected axiomatic theories. The new concepts and facts are axiomatic theories, definitions, and theorems added to the FML. An *axiomatic theory* (*theory* for short) is a pair $T = (L, \Gamma)$ where L is a language of the logic and Γ is a set of sentences of L called the *axioms* of T that serve as the background assumptions about the ideas that can be expressed in L. Two theories T_1 and T_2 in the FML are connected if there is a *theory extension* from T_1 to T_2 or, more generally, a *theory morphism* from T_1 to T_2, which is a structure-preserving translation from the expressions of T_1 to the expressions of T_2 that is also meaning preserving in the sense that each theorem of T_1 is translated to a theorem of T_2.

Formal mathematics offers the following major benefits to mathematics practitioners, i.e., mathematicians, computing professionals, engineers, and scientists who use mathematics in their work:

1. The mathematical statements needed or created by a mathematics practitioner can be expressed precisely as sentences of a language in the logic since the languages of the logic have a precise syntax and semantics.
2. Conceptual errors can be systematically discovered during the process of expressing mathematical ideas in a language of the logic since the logic offers a precise conceptual framework. This is similar to how type errors are caught in a modern programming language by type checking.
3. Software can be used to study, discover, communicate, and certify the mathematical knowledge produced since the languages of the logic have a precise common syntax. Moreover, there is a precise basis for verifying the correctness of this software since the languages also have a precise common semantics.
4. The correctness of the mathematical knowledge can be verified with a very high level of assurance by constructing and mechanically checking formal proofs in the logic's formal proof system.
5. Mathematical knowledge can be regarded as a formal structure that can be studied, developed, searched, and presented using software.

(See [10] for a more detailed description of these five benefits of formal mathematics.) All of these benefits can be realized in an FML. The FML for a mathematics development effectively serves as a precise form of documentation that grows as the development progresses.

Several large FMLs has been constructed using proof assistants including those associated with the proof assistants ACL2 [17], Agda [3], HOL [13], HOL Light [15], Isabelle/HOL [24], Lean [5], Methmath/ZFC [20], Mizar [22], PVS [23], and Roq [25]. These libraries serve the user communities of their corresponding

proof assistants as well as the formal mathematics community as a whole. However, there is a place for smaller specialized FMLs that serve projects involved with mathematics research, software development, engineering, etc. FMLs, both large and small, are also very useful in mathematics education for organizing and presenting mathematical knowledge. In fact, building a small FML is one of the best ways for students to understand how mathematics is a process of creating, exploring, and interconnecting mathematical models.

The standard way of building an FML is to employ the *standard approach to formal mathematics* in which mathematics is done with a proof assistant and all details are formally proved and mechanically checked. The primary strength of the standard approach is *certification*. By verifying mathematical results using machine-checked formal proofs, the standard approach produces mathematics that has a very high level of assurance of being correct. The approach is thus crucial in situations where correctness is a paramount concern as with mathematics that is highly complex, poorly understood (often due to its novelty), or underlying a critical real-world application.

On the other hand, the primary weakness of the standard approach is *communication*. Generally speaking, proof assistants sacrifice communication for the sake of certification. Formal proofs are great for certifying that something is true but often do a poor job of communicating why something is true. Most mathematics practitioners — including mathematicians — are much more interested in communicating mathematical ideas than in formally certifying their correctness. This is particularly true in mathematics education or in applications of well-understood mathematics. Another weakness of the standard approach is that proof assistants are, as a rule, difficult to implement and to learn how to use.

The purpose of this talk is to describe how to build an FML in which communication is prioritized over certification using the *free approach to formal mathematics* [10], an alternative to the standard approach to formal mathematics that is *free* of the obligation to formally prove and mechanically check all details using a proof assistant. The talk presents the free approach, describes how an communication-oriented FML can be built using the free approach, and ends by encouraging the mathematics software community to develop the logics and software needed to build communication-oriented FMLs for applications in which the high assurance of correctness achieved by fully certified formal mathematics is unnecessary.

2 The Free Approach to Formal Mathematics

The free approach to formal mathematics focuses on two goals: *communication* and *accessibility*. To achieve these goals, an implementation of the free approach needs to satisfy four requirements which are stated and explained below. These four requirements characterize the free approach.

The first requirement (R1) is that *the underlying logic is fully formal and supports standard mathematical practice*. Supporting mathematical practice makes the logic easier to learn and use and makes formalization a more natural process.

It also makes formal mathematical statements easier to read, write, and reason about — which helps the developer to identify mistakes and see connections.

The second requirement (R2) is that *proofs can be traditional, formal, or a combination of the two.* This flexibility in how proofs are written enables proofs to be a vehicle for communication as well as certification. Traditional proofs are usually easier to read and write and better suited for communicating the ideas behind proofs than pure formal proofs.

The third requirement (R3) is that *there is support for organizing mathematical knowledge using the little theories method* [11]. This method organizes mathematical knowledge as a network of interconnected theories called a *theory graph* [19]. The nodes of the graph are *theories* of a logic and the directed edges are *theory morphisms* between the theories. To maximize clarity, each mathematical topic is developed in the "little theory" in the theory graph that has the most convenient vocabulary and level of abstraction. To minimize redundancy, the definitions and theorems produced in this little theory are transported, as needed, to other theories via the theory morphisms in the theory graph instead of directly producing them in the theories where they are needed.

And, finally, the fourth requirement (R4) is that *there are several levels of supporting software.* The levels can range from just LaTeX support to a full proof assistant. An important intermediate example is a software system that is the same as a proof assistant except it only supports the development of traditional proofs. A system of this kind, a *traditional-proof proof assistant*, would be much simpler and thus easier to both implement and learn how to use than a typical proof assistant. Having several levels of software support enables the user to choose the level of support they want to have and the level of investment in learning the software they want to make.

An implementation of the free approach is likely to serve the needs of the typical mathematics practitioner — who is likely more interested in communicating mathematical ideas than in formally certifying their correctness — much better than an implementation of the standard approach. This is especially true when the mathematical knowledge involved is well understood and certification via traditional proof is adequate for the purpose at hand. In summary, the free approach is not a replacement for the standard approach, but we believe it would be more useful, accessible, and natural than the standard approach for the great majority of mathematics practitioners.

Three other alternatives to the standard approach to formal mathematics have been proposed. The first is Tom Hales' *formal abstracts in mathematics* project [14] in which proof assistants are used to create *formal abstracts*, which are formal presentations of mathematical theorems without formal proofs. The second is Michael Kohlhase's *flexiformal mathematics* [16, 18] initiative in which mathematics is a mixture of traditional mathematics with traditional proofs and formal mathematics with formal proofs. The third is the *natural language approach* in which mathematics is performed using a proof assistant like Naproche [21] or Natty [6] that accepts a controlled form of mathematical English (or another natural language) as input.

The free approach differs from each of these approaches in fundamental ways. The formal abstracts approach seeks to formalize collections of theorems without proofs using proof assistants and hold open the possibility of adding formal proofs later. The free approach is not committed to using proof assistants and is not required to facilitate the addition of formal proofs. The flexiformal approach gives the user the flexibility to produce mathematics that is partly formal with formal proofs and partly nonformal with traditional proofs. The free approach in contrast produces mathematics that is fully formal except that proofs may be traditional. The natural language approach produces fully certified formal mathematics written in natural language. The use of natural language can improve communication, but the natural language approach is really just an instance of the standard approach in which natural language notation is used to present formal expressions.

Finally, the reader may be wondering whether generative AI will make the free approach obsolete. Autoformalization using an LLM can replace a traditional proof in a body of mathematics with a formal proof when the mathematics is built on a solid logical foundation. Translating a formal certification-oriented proof to a traditional communication-oriented proof using an LLM is likely to be a much bigger challenge, requiring a deep understanding of the mathematical context in which the proof resides and the target audience for the proof. We thus expect that autoformalization will be much more successful at certifying a traditional proof than "autoexplanation" will be at communicating the key ideas embodied in a formal proof. For the foreseeable future, effective communication of mathematical ideas will need human intelligence more than artificial intelligence.

3 How to Build a Communication-Oriented FML

Suppose that a group of mathematics practitioners want to build a communication-oriented FML using the free approach to formal mathematics. This can be done as follows.

First, the group chooses a logic that is appropriate for developing the FML. The logic should be close to mathematical practice. It should include a module system that supports the little theories method. It should provide appropriate levels of both theoretical and practical expressibility. (*Theoretical expressivity* is the measure of what ideas can be expressed in the logic without regard to how the ideas are expressed, and *practical expressivity* is the measure of how readily ideas can be expressed in the logic.) And it should be equipped with the level of software support desired by the group.

Second, the group constructs the FML using the little theories method, the logic's module system, and the logic's software support. The construction involves the following four activities:

1. Adding theories to the FML.
2. Developing the theories in the FML by producing definitions and theorems in them.

3. Constructing theory morphisms between the theories in the FML.
4. Transporting definitions and theorems from one theory to another via the theory morphisms in the FML.

Third, the group writes proofs in a flexible manner to maximize communication. The proofs are needed to validate the definitions, theorems, and theory morphisms in the FML. The proofs can be traditional proofs, formal proofs in the logic's proof system, or semiformal proofs that demonstrate the existence of a formal proof.

We have illustrated in [12] how a modest-size FML associated with monoid theory can be constructed using the free approach. The FML includes 12 theories, 18 theory morphisms, 15 definitions, and 36 theorems.

The underlying logic of the FML is Alonzo [9], a practice-oriented version of Church's type theory [2, 4], Alonzo Church's formulation of simple type theory [8]. Named in honor of Church, Alonzo is a classical higher-order predicate logic that has a very simple type system and is tailored for reasoning about mathematical structures. Its formal proof system is derived from Peter Andrews' elegant proof system for $\mathcal{Q}_0$ [1]. Unlike traditional predicate logics, it admits partial functions and undefined expressions in accordance with the approach employed in mathematical practice that we call the *traditional approach to undefinedness* [7, 9]. It has a *formal notation* for machines and a *compact notation* for humans, based on a rich set of notational definitions and conventions, that closely resembles the notation found in mathematical practice. And it has two semantics, one for mathematics based on *standard models* and one for logic based on Henkin-style *general models*. Taken as a whole, Alonzo offers a very good balance between theoretical and practical expressivity.

The FML for monoid theory is constructed using Alonzo's module system [9] that is designed specifically to support the little theories method. It includes modules for constructing theories and theory morphisms, developing theories by defining new concepts and posing and proving conjectures, and transporting definitions and theorems from one theory to another. The FML is constructed with just the simplest level of software support: LaTeX macros for presenting Alonzo types and expressions and LaTeX environments for presenting Alonzo modules. All the proofs are written in a traditional style, but several of them make use of the axioms, rules of inference, and metatheorems of Alonzo's formal proof system. Thus [12] demonstrates how a communication-oriented FML can be constructed using a practice-oriented logic like Alonzo, minimal software support, and entirely traditional proofs.

4 Conclusion

A formal mathematical library (FML) offers a mathematics practitioner a precise way of organizing and documenting mathematical knowledge. It is a mathematical structure that can be studied, developed, searched, and presented using software. FMLs are usually constructed today using the standard approach to formal mathematics in which mathematics is done with a proof assistant and all details

are formally proved and mechanically checked. As a result, most FMLs prioritize certification over communication — which does not serve the mathematics practitioner for whom communication is more important than formal certification.

Using the free approach to formal mathematics, FMLs can be developed that prioritize communication over certification. Building FMLs in this way is likely to be attractive to mathematics practitioners who are foremost interested in communicating mathematical ideas and who do not want to pay the high cost of using the standard approach to build an FML. It is also likely to be attractive to mathematics educators and their students who want to explore the mathematical structure of bodies of mathematical knowledge.

The mathematics community would greatly benefit from having more mathematics practitioners construct and employ FMLs. To achieve this goal, we need more work by logic engineers to design practice-oriented logics and more work by mathematical software developers to build easy-to-use software tools for constructing communication-oriented FMLs.

References

1. P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof*, volume 27 of *Applied Logic Series*. Springer, second edition, 2002.
2. C. Benzmüller and P. Andrews. Church's type theory. In E. N. Zalta and U. Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2025 edition, 2025. Available at `http://plato.stanford.edu/archives/spr2024/entries/type-theory-church/`.
3. A. Bove, P. Dybjer, and U. Norell. A brief overview of Agda — A functional language with dependent types. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2009)*, volume 5674 of *Lecture Notes in Computer Science*, pages 73–78. Springer, 2009.
4. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
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. A. Dingle. A natural-language proof assistant for higher-order logic (work in progress). In C. W. Brown, D. Kaufmann, C. Nalon, A. Steen, and M. Suda, editors, *Proceedings of the 9th Workshop on Practical Aspects of Automated Reasoning (PAAR 2024, co-located with the IJCAR 2024, Nancy, France)*, volume 3717 of *CEUR Workshop Proceedings*, pages 57–73. CEUR-WS.org, 2024.
7. W. M. Farmer. Formalizing undefinedness arising in calculus. In D. A. Basin and M. Rusinowitch, editors, *Automated Reasoning — IJCAR 2004*, volume 3097 of *Lecture Notes in Computer Science*, pages 475–489. Springer, 2004.
8. W. M. Farmer. The seven virtues of simple type theory. *Journal of Applied Logic*, 6:267–286, 2008.
9. W. M. Farmer. *Simple Type Theory: A Practical Logic for Expressing and Reasoning About Mathematical Ideas*. Computer Science Foundations and Applied Logic. Birkhäuser/Springer, second edition, 2025.
10. W. M. Farmer. An alternative approach to formal mathematics that focuses on communication and accessibility. *Computing Research Repository (CoRR)*, abs/2603.20893, 2026.

11. 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.
12. W. M. Farmer and D. Y. Zvigelsky. Monoid theory in Alonzo: A little theories formalization in simple type theory. *Journal of Applied Logics — IfCoLog Journal of Logics and their Applications*, 12:1853–1939, 2025.
13. M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.
14. T. Hales. Formal abstracts in mathematics. In F. Rabe, W. M. Farmer, G. O. Passmore, and Y. Abdou, editors, *Intelligent Computer Mathematics (CICM 2018)*, volume 11006 of *Lecture Notes in Computer Science*, page xiii. Springer, 2018.
15. J. Harrison. HOL Light: An overview. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2009)*, volume 5674 of *Lecture Notes in Computer Science*, pages 60–66. Springer, 2009.
16. M. Iancu. *Towards Flexiformal Mathematics*. PhD thesis, Jacobs University Bremen, 2017.
17. M. Kaufmann, P. Manolios, and J. S. Moore. *Computer-Aided Reasoning: An Approach*. Advances in Formal Methods. Kluwer/Springer, 2000.
18. 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 Computer Society, 2012.
19. 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 (CICM 2010)*, volume 6167 of *Lecture Notes in Computer Science*, pages 370–384. Springer, 2010.
20. Metamath Proof Explorer Home Page. `http://us.metamath.org/mpeuni/mmset.html`. Accessed on 14 February 2026.
21. Naproche. `https://naproche.github.io/`. Accessed on 18 March 2026.
22. A. Naumowicz and A. Korniłowicz. A brief overview of Mizar. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2009)*, volume 5674 of *Lecture Notes in Computer Science*, pages 67–72. Springer, 2009.
23. 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 (CAV 1996)*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414. Springer, 1996.
24. L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.
25. The Rocq Prover. `https://rocq-prover.org/`. Formerly the Coq Proof Assistant; accessed on 14 February 2026.