An Alternative Approach to Formal Mathematics that Focuses on Communication and Accessibility

William M. Farmer*

March 20, 2026

1 Introduction

Formal mathematics is mathematics done within the framework of a formal logic.[1] Formal mathematics differs from traditional mathematics in several ways. In formal mathematics all mathematical statements are expressed as formulas in a formal language with a precise semantics, while in traditional mathematics mathematical statements are expressed in a natural language like English that does not have a precise semantics. In formal mathematics all assumptions and definitions are explicitly stated, while in traditional mathematics many assumptions and definitions are stated vaguely or not at all. And in formal mathematics all reasoning is performed in accordance with the logic's notion of logical consequence, while in traditional mathematics reasoning is often not grounded on a precise notion of logical consequence. Formal mathematics is based on ideas from logic, the foundations of mathematics, and axiomatic systems. These ideas were developed over the last few centuries by mathematicians and logicians such as (in chronological order) Leibniz, Lobachevsky, Boole, Dedekind, Peirce, Cantor, Frege, Peano, Hilbert, Zermelo, Russell, Curry, Tarski, Church, Gödel, and Bourbaki.

Before the advent of electronic computers, interest in formal mathematics was primarily a theoretical endeavor to develop a solid foundation for mathematical thinking and to better understand what mathematics is. A salient example is the three-volume *Principia Mathematica* [20] by Alfred North Whitehead and Bertrand Russell that was intended to show how mathematical ideas could be expressed in a formal logic and how the set-theoretic and semantic paradoxes identified in the late 19th and early 20th centuries could be resolved.

*The author is a Professor in the Department of Computing and Software at McMaster University. His email address is wmfarmer@mcmaster.ca.

[1] Formal mathematics is also called "formalized mathematics", but this latter term is often reserved for the product that formal mathematics produces.

In the late 1960s, Nicolaas G. de Bruijn started the Automath project [19] whose aim was the design of a formal language in which mathematical ideas could be readily expressed and checked by a computer. The goal was to assist mathematicians in developing new mathematics while remaining close to mathematical practice. The project was the first attempt to do formal mathematics for practical purposes, and the Automath system was both the first proof assistant[2] and the first logical framework[3]. Following Automath, several other proof assistants and logical frameworks have been developed including (in roughly chronological order) Mizar, Edinburgh LCF, TPS, Nuprl, HOL, Isabelle, Rocq (formerly Coq), ACL2, PVS, Metamath, IMPS, HOL Light, LEGO, Agda, Twelf, Lean, and Naproche.

Although interest in using proof assistants to construct and mechanically check formal proofs has been steadily growing [4], formal mathematics is still not a prominent component of mathematical practice. We know of no studies that have investigated the number of mathematics practitioners — mathematicians, computing professionals, engineers, and scientists — who work with formal mathematics. However, a back-of-the-envelope estimate is that no more (and probably far less) than 1% of all mathematics practitioners use a proof assistant in their work.[4] It is safe to say that only a very small minority of mathematics practitioners are pursuing formal mathematics today.

We believe formal mathematics holds great promise for mathematics. In particular, formal mathematics offers mathematics practitioners five major benefits: (1) *greater rigor*, (2) *the systematic discovery of conceptual errors*, (3) *software support*, (4) *mechanically checked results*, and (5) *mathematical knowledge as a formal structure* (see Section 3 for detailed descriptions of these benefits). However, the promise of formal mathematics has largely been unrealized since so few mathematics practitioners ever do mathematics with the aid of a formal logic.

The purpose of this paper is to explain what formal mathematics is, what benefits it offers, why it is unpopular, and finally and most importantly, why an alternative approach is needed to make the benefits of formal mathematics more widely available.

2 What is Formal Mathematics?

We have said that formal mathematics is mathematics done within the framework of a formal logic. So what does that mean? To answer this question we have to first define a "formal logic", then define a "theory" of a formal logic, and finally define a "theory morphism" from one theory to another.

[2] A *proof assistant* is an interactive software system for developing and checking formal proofs.

[3] A *logical framework* is a software system that provides the means to define and implement multiple logics.

[4] There is very likely more than 30 million mathematics practitioners in the world and many, many more if mathematics students are included. On the other hand, there is very likely fewer than 300,000 people in the world who use proof assistants.

A *formal logic* (*logic* for short) is a *family of languages* such that:

1. The languages of the logic have a *precise common syntax*. The languages have the same syntactic structure, but they have different vocabularies. That is, they have the same logical symbols, but they have different non-logical symbols.

2. The languages of the logic have a *precise common semantics with a notion of logical consequence*. If L is a language of the logic, A is a sentence of L, and Γ is a set of sentences of L, then "A is a logical consequence of Γ" means that in every possible situation, if all the sentences in Γ are true, then A is also true.

3. There is a sound *formal proof system* for the logic in which proofs can be syntactically constructed. Let L be a language of the logic, A a sentence of L, and Γ a set of sentences of L. Soundness means that, if a proof of A from Γ can be constructed in the proof system, then A is a logical consequence of Γ.

This is a very general definition of a logic. The key constituent of a logic is its notion of logical consequence. The definition covers the various versions of first-order logic, simple type theory, and dependent type theory as well as the various versions of set theory. Note that a family of languages can be a family of one language by itself.

A *theory* of a logic is a pair $T = (L, \Gamma)$ where L is a language of the logic and Γ is a set of sentences of L. The members of Γ are the *axioms* of T; they serve as background assumptions about the ideas that can be expressed in L. A theory is a fundamental unit of mathematical knowledge that is *developed* by defining new concepts in the theory and by posing and then proving conjectures in the theory. Mathematical knowledge can be organized either as a single, large, highly developed theory or, much better, as a network of interconnected theories of varying size and development. We will look next at the mechanism for connecting theories.

Let T and T' be theories of a logic. A *theory morphism from T to T'* is a translation Φ from the expressions of T to the expressions of T' that is meaning preserving in the sense that each theorem of T is translated to a theorem of T'. The concepts (established by definitions) and facts (established by theorems) of T can be transported to T' via Φ. For example, a theory morphism $\Phi_{\mathsf{MON} \mapsto \mathsf{COF}^+}$ from a theory MON of monoids to a theory COF of complete ordered fields (which specifies real number mathematics) that maps the binary operator $\cdot$ and the identity element e in MON to $+$ and 0, respectively, in COF can be used to transport the theorem id-elt-is-unique of MON that says e is the unique identity element with respect to $\cdot$ to the theorem of COF that says 0 is the unique identity element with respect to $+$. That is, $\Phi_{\mathsf{MON} \mapsto \mathsf{COF}^+}$ translates a definition or theorem (which can be an axiom) X of MON to an instance of X that is a definition or theorem of COF. (Note that the theory morphism $\Phi_{\mathsf{MON} \mapsto \mathsf{COF}^*}$ from MON to COF that translates the binary operator $\cdot$ and the identity element e to

$*$ and 1, respectively, can transport id-elt-is-unique to the theorem of COF that says 1 is the unique identity element with respect to $*$.) Theory morphisms are thus unidirectional conduits through which information in the form of concepts and facts flows from usually more abstract to usually more concrete theories.

The *little theories method* [12] is a method for organizing mathematical knowledge as a network of interconnected theories called a *theory graph* [18]. 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 level of abstraction and the most convenient vocabulary. 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 other theories.

Figure 1 shows an example of a theory graph associated with monoid theory that is taken from [14]. A theory morphism that is an inclusion (i.e., an identity mapping) is designated by a $\hookrightarrow$ arrow and a noninclusion is designated by a $\rightarrow$ arrow. There are many, many more useful theory morphisms that are not shown, including $\Phi_{\mathsf{MON}\mapsto\mathsf{COF}^+}$ (and $\Phi_{\mathsf{MON}\mapsto\mathsf{COF}^*}$) as well as a vast number of other theory morphisms to the theory COF.

Revisiting our example above, the theorem id-elt-is-unique about the uniqueness of the identity element in a monoid is stated and proved in MON and then transported to COF via $\Phi_{\mathsf{MON}\mapsto\mathsf{COF}^+}$ instead of directly stating and proving the instance of this theorem in COF or in any other theory that embodies a monoid structure. The little theories method thus provides a very strong form of *polymorphism*: Any definition or theorem X produced in a theory T can be reused in every theory T' for which there is a theory morphism from T to T'.

Now we can illustrate with a simple example how "mathematics is done within a formal logic". We will use the little theories method. Suppose that we would like to show that a conjecture C is true within some logic. Let G be a theory graph of the logic that has been constructed using the little theories method. We proceed by obtaining a theory $T = (L, \Gamma)$ that best captures the context in which we want to consider C. We obtain T by either selecting a theory that is already in G or by constructing a new theory that is added to G and connected as needed to other theories in G via theory morphisms. Then we express C as a sentence A_C of the language L augmented with the definitions made in T and the definitions transported to T from other theories via theory morphisms. And finally we try to prove A_C is a theorem of T by showing A_C is a logical consequence of the axioms of T. This can be done either by composing a traditional-style proof that A_C follows from Γ (the axioms of T), the theorems proved in T, and the theorems transported to T via theory morphisms or by constructing a formal proof in the formal proof system of the logic.

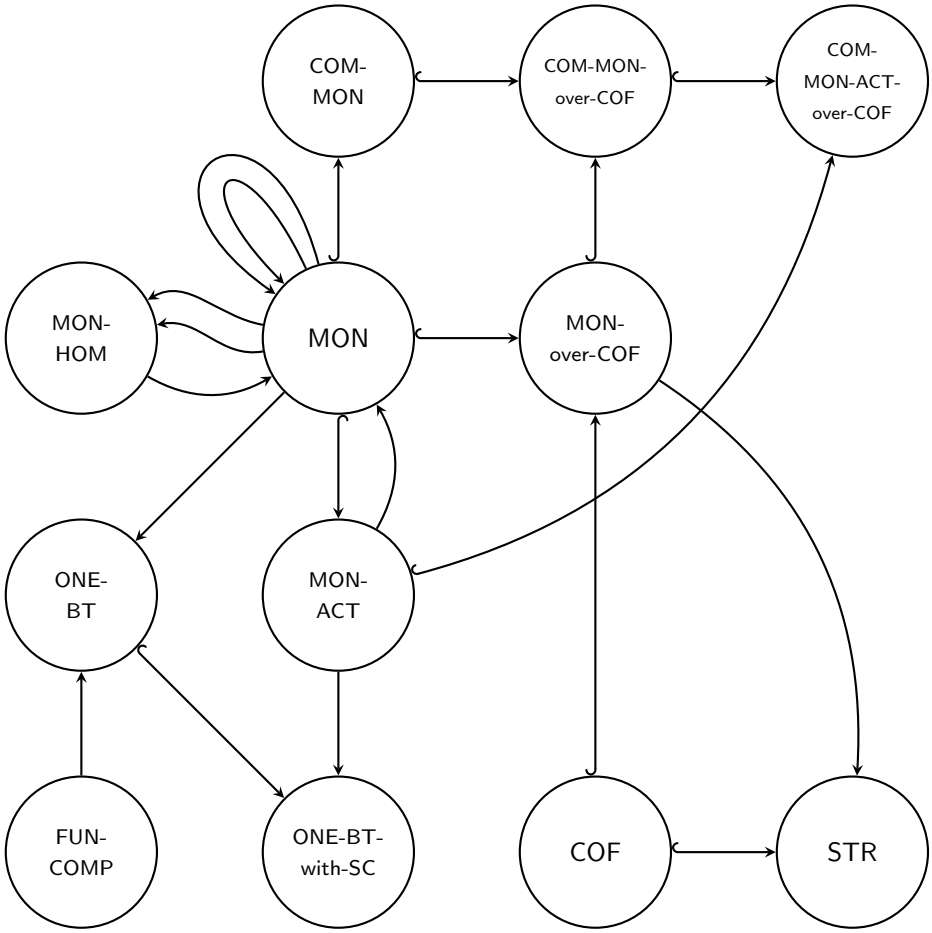

Figure 1: A Theory Graph for Monoid Theory

We thus see that doing formal mathematics involves the following five activities:

1. Constructing theories in a logic.
2. Defining new concepts in a theory.
3. Posing and then proving conjectures in a theory.
4. Constructing theory morphisms.
5. Transporting definitions and theorems via theory morphisms.

(Of course, the later two activities are not needed if all the work is done in one theory.)

3 What are the Benefits of Formal Mathematics?

There are five major benefits of formal mathematics.

The first benefit is that *mathematics can be done with greater rigor*. A prime goal of mathematics is to express and reason about mathematical ideas in a highly rigorous manner. There are two aspects of traditional mathematical practice that stand in the way of rigor. The first is that mathematical ideas are usually expressed in a natural language, such as English, that does not have a precise semantics, and the second is that assumptions, definitions, theorems, and even the notion of logical consequence are often expressed imprecisely or implicitly. These shortcomings do not arise in formal mathematics since (1) a logic has a precise notion of logical consequence, (2) all mathematical concepts and statements are expressed in a language that has a precise semantics, and (3) all assumptions, definitions, and theorems are expressed explicitly as axioms, definitions, and theorems of a theory.

The second benefit is that *conceptual errors can be systematically discovered*. It is easy to make conceptual errors in mathematics, especially in mathematics that is complex or not well understood. A logic offers a precise conceptual framework to a mathematics practitioner. The process of expressing ideas in the logic can systematically lead to the discovery of many conceptual errors. Let us consider the following very simple example. Suppose that a definition is employed early in a mathematical development, and a closely related but inequivalent definition is employed by mistake late in the development in place of the first definition. This mistake, if not discovered, could lead to invalid results. If the development were done within a formal logic, this mistake would be easily caught since every definition would be expressed as a sentence with a precise syntax and semantics. The process of expressing mathematical ideas in a formal logic naturally leads to many conceptual errors being caught similarly to how type errors are caught in a modern programming language by type checking. Thus conceptual errors can be discovered systematically in formal mathematics in a way that is largely not possible in traditional mathematics.

The third benefit is that *mathematics can be done with software support*. Since the languages of a formal logic have a precise common syntax, the expressions of a language can be represented as data structures. The expressions can then be analyzed, manipulated, and processed by software applied to the data structures that represent them. This, in turn, enables the study, discovery, communication, and certification of mathematics to be done with the aid of software. The software can provide many useful services such as performing numerical and symbolic computations, solving various kinds of problems, and displaying mathematical data and expressions. Since the languages also have a precise common semantics, there is a precise basis for verifying the correctness of this software.

The fourth benefit is that *results can be mechanically checked*. Formal proofs can be represented as data structures, and software can be used to check that

these data structures represent actual proofs in the formal proof system of the logic. *Proof assistants* can be used to help humans construct formal proofs, and *automated theorem provers* can be used to automatically discover formal proofs. Since the software needed to check the correctness of the formal proofs is often very simple and easily verified itself, it is possible to verify the correctness of the formal proofs with a very high level of assurance. Thus mathematical results can be mechanically checked by constructing formal proofs of them in the formal proof system of the logic and then using software to check the correctness of the formal proofs. High assurance of mathematical correctness is especially needed for cutting-edge mathematics research [2] and the development of safety-, security-, and mission-critical software [4].

Finally, the fifth benefit is that *we can regard mathematical knowledge as a formal structure*. Using the little theories method, a body of mathematical knowledge can be represented as a theory graph built by creating theories, defining new concepts, posing and then proving conjectures, constructing theory morphisms, and transporting definitions and theorems via theory morphisms. The knowledge embodied in a formal structure of this kind can be studied, managed, searched, and presented using software.

The benefits of formal mathematics are huge. Greater rigor and discovering conceptual errors have been principal goals of mathematicians for thousands of years. Software support can greatly extend the reach and productivity of mathematics practitioners. Mechanically checked results can drive mathematics forward in areas where the ideas are poorly understood (often due to their novelty) or highly complex. And mathematical knowledge as a formal structure can enable the techniques and tools of mathematics and computing to be applied to mathematical knowledge itself.

4 Why is Formal Mathematics Unpopular?

One would expect that, by virtue of these five benefits, formal mathematics would be very popular among mathematics practitioners. Yet the opposite is true. So why is formal mathematics so unpopular?

The answer lies in 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 standard approach has three major strengths:

1. It achieves all five benefits of formal mathematics mentioned above.

2. All theorems are verified by mechanically checked formal proofs. Thus there is a very high level of assurance that the results produced are correct.

3. There are several powerful proof assistants available, such as ACL2, HOL, HOL Light, Isabelle/HOL, Lean, Metamath/ZFC, Mizar, PVS, and Rocq (formerly Coq), that support the approach.

But the standard approach also has two important weaknesses:

1. It prioritizes certification over communication.

2. It is not accessible to the great majority of mathematics practitioners.

Proof assistants are designed primarily for formally *certifying mathematical results* and only secondarily for *communicating mathematical ideas*. Formal proofs are great for certifying that something is true but poor for 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 for applications that involve well-understood mathematics, the kind of mathematics that arises in mathematics education and routine applications.

To apply the standard approach, a mathematics practitioner needs a solid understanding of formal logic and how to use a proof assistant to develop theories in the underlying logic of the proof assistant. Since the focus of the standard approach is on certification, the proof assistants supporting the approach are generally complex, based on unfamiliar logics, difficult to learn how to use, and far removed from mathematical practice. The investment needed to learn a specific proof assistant is so high that the user is usually tied to just one proof assistant and is proficient with just one logic, one formal proof system, one set of notation, and one set of software tools. This makes it difficult for a proof assistant user to share their results with users of other proof assistants. Having to express and check all details in an often unfamiliar logic using a complex proof assistant that utilizes certification-oriented ways of expressing and reasoning about mathematics is a bridge too far for many mathematics practitioners.

Thus the standard approach to formal methods does not adequately serve the average mathematics practitioner. As a result, the average mathematics practitioner does not have access to the benefits of formal mathematics since the standard approach is pretty much the only game in town.

We are not saying that proof assistant developers care only about certification. They want to improve support in their systems for communication and to make their systems more accessible. Significant progress has been made in both directions, and this has helped accelerate the growth of the proof assistant user community. There has been particularly strong recent growth in the number of people using the Lean proof assistant and working on the development of Lean's Mathlib library of mathematical knowledge and programming infrastructure. Nevertheless, nearly all contemporary proof assistants prioritize certification over communication and accessibility. (The Naproche system is one notable exception.)

5 Why is an Alternative Approach Needed?

We strongly believe an alternative to the standard approach to formal mathematics is needed that focuses on two goals, *communication* and *accessibility*, the two weaknesses of the standard approach. We propose an alternative approach of this kind called the *free approach to formal mathematics* since it is *free* of the obligation to formally prove and mechanically check all details of a mathematical development using a proof assistant. To achieve the goals of communication and accessibility, an implementation of the free approach needs to satisfy the following requirements:

R1. *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 mathematics statements easier to read, write, and reason about — which helps the developer to identify mistakes and see connections.

R2. *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 easier to read and write than formal proofs and usually better suited for communicating the ideas behind proofs.

R3. *There is support for organizing mathematical knowledge using the little theories method.* This enables mathematical knowledge to be formalized to maximize clarity and minimize redundancy (as explained above).

R4. *There are several levels of supporting software.* The levels can range from just LaTeX support to a full proof assistant to an *interactive mathematics laboratory (IML)* [11] that provides support for inference as well as other aspects of mathematics [5]. 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. Such a system would be much simpler and thus easier to both implement and learn how to use than a typical proof assistant. The user can thus choose the level of software support they want to have and the level of investment in learning the software they want to make.

These four requirements characterize the free approach.

The free approach is intended to achieve all five benefits of formal mathematics mentioned above, but it cannot achieve the same level of assurance, as the standard approach, that the results produced are correct. This is because, in order to prioritize communication and accessibility over certification, the free approach does not require that all details are formally proved and mechanically checked.

An implementation of the free approach — with support for standard mathematical practice, traditional proofs, the little theories method, and several levels

of software — is likely to serve the needs of the average mathematics practitioner 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. Thus it would bring the benefits of formal mathematics to vastly more people than the standard approach.

In recognition that the standard approach has been adopted by only a very small part of the mathematics community, other alternative approaches to formal 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 [15] 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, 17] initiative in which mathematics is a mixture of traditional and formal mathematics and proofs can be either traditional or formal.

The free 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 using proof assistants and hold open the possibility adding formal proofs later, while the free approach can be done with different levels of software support. The objective of the flexiformal mathematics approach is to give the user the flexibility to produce mathematics with varying degrees of formality. In contrast, the free approach aims to produce mathematics that is fully formal except possibly for proofs.

6 An Implementation of the Free Approach

We have developed an implementation of the free approach based on Alonzo [11], a practice-oriented classical higher-order version of predicate logic that extends first-order logic. Named in honor of Alonzo Church, Alonzo is a version of Church's type theory [6], Church's formulation of simple type theory [3, 10]. It is closely related to Peter Andrews' $\mathcal{Q}_0$ [1] and LUTINS [7, 8], the logic of the IMPS proof assistant [13].

Alonzo is designed to be as close to mathematical practice as possible. By virtue of being a form of predicate logic, it is built on ideas that are widely familiar to mathematics practitioners. It is a type theory, but its type system is simpler than the type systems of most of the type theories employed in proof assistants. It is equipped with facilities, including higher-order quantification and definite description, for reasoning about functions, sets, tuples, lists, and mathematical structures. Unlike traditional predicate logics, Alonzo admits partial functions and undefined expressions in accordance with the approach employed in mathematical practice that we call the *traditional approach to un-*

definedness [9, 11]. With this approach, non-Boolean expressions that naturally have no value — like the top of an empty stack or $\mathsf{lim}_{x \to 0}\, \mathsf{sin}(1/x)$ — are considered *undefined* and denote nothing at all, but Boolean expressions — even if they contain undefined non-Boolean expressions — always denote either true or false.[5]

Alonzo has a simple syntax with a *formal notation* for machines and a *compact notation* for humans that closely resembles the notation found in mathematical practice. The compact notation is defined by the extensive set of *notational definitions and conventions* given in [11]. Alonzo has two semantics, one for mathematics based on *standard models* and one for logic based on Henkin-style *general models*. Taken as a whole, it offers a very good balance between *theoretical expressivity* (the measure of what ideas can be expressed without regard to how the ideas are expressed) and *practical expressivity* (the measure of how readily ideas can be expressed).

Figure 2 shows part of the development of the Alonzo theory COF of complete ordered fields taken from Chapter 13 (Real Number Mathematics) of [11]. It contains several definitions and theorems from calculus presented using various notational definitions including those given in Figure 3. R is the type of real numbers and $N_{\{R\}}$ denotes the set of natural numbers as a subset of the set of real numbers. The reader should note that the functions in these definitions and theorems are "Curryed". For example, a binary function that takes values of type α and β as input and returns a value of type γ as output is represented in Curryed form as a function of type $\alpha \to (\beta \to \gamma)$ or, more simply, as $\alpha \to \beta \to \gamma$. (A binary function could also be represented in Alonzo, if desired, as a function of type $(\alpha \times \beta) \to \gamma$.) The reader should also note that the definitions of a limit of a function and a limit of a sequence employ definite descriptions of the form $\mathrm{I}\, b : R\ .\ \mathbf{A}_o$ that denotes the unique real number b that satisfies the formula $\mathbf{A}_o$ if such a real number exists and is undefined otherwise.

We believe that, by virtue of its syntax, semantics, and notational definitions and conventions, Alonzo satisfies requirement R1 as well or better than almost any other formal logic.

There is a formal proof system for Alonzo [11] which is derived from Andrews' elegant proof system for $\mathcal{Q}_0$ [1]. Proofs, however, are not required to be formal in our implementation of the free approach, and so requirement R2 is satisfied.

Alonzo has a module system for organizing mathematical knowledge using the little theories method [11]. (Since partial functions naturally arise from theory morphisms [8], the little theories method works best with a logic like Alonzo that supports partial functions.) 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. Thus Alonzo fully satisfies requirement R3.

Our implementation of the free approach provides only the simplest level of software support: LaTeX macros for presenting Alonzo types and expres-

[5] The traditional approach to undefinedness is exemplified in Michael Spivak's famous textbook *Calculus*; see the discussion in [9].

Def10: $\mathsf{lim}_{(R\to R)\to R\to R} =$
$\lambda\, f : R \to R\,.\,\lambda\, a : R\,.\,\mathrm{I}\, b : R\,.$
$(\forall\, e : R\,.\,0 < e \Rightarrow$
$(\exists\, d : R\,.\,0 < d \wedge$
$(\forall\, x : R\,.\,0 < |x - a| < d \Rightarrow$
$|f\, x - b| < e)))$ (limit of a function).

Def13: $\mathsf{lim\text{-}seq}_{(R\to R)\to R} =$
$\lambda\, s : N_{\{R\}} \to R\,.\,\mathrm{I}\, b : R\,.$
$(\forall\, e : R\,.\,0 < e \Rightarrow$
$(\exists\, m : N_{\{R\}}\,.$
$(\forall\, n : N_{\{R\}}\,.\,m < n \Rightarrow$
$|s\, n - b| < e)))$ (limit of a sequence).

Def14: $\mathsf{cont\text{-}at}_{(R\to R)\to R\to o} =$
$\lambda\, f : R \to R\,.\,\lambda\, a : R\,.\,\lim_{x\to a} f\, x = f\, a$ (continuous at a point).

Def18: $\mathsf{cont\text{-}on\text{-}closed\text{-}int}_{(R\to R)\to R\to R\to o} =$
$\lambda\, f : R \to R\,.\,\lambda\, a : R\,.\,\lambda\, b : R\,.$
$(\forall\, x : R\,.\,a < x < b \Rightarrow \mathsf{cont\text{-}at}\, f\, x) \wedge$
$\mathsf{right\text{-}cont\text{-}at}\, f\, a \wedge$
$\mathsf{left\text{-}cont\text{-}at}\, f\, b$ (continuous on a closed interval).

Def19: $\mathsf{deriv\text{-}at}_{(R\to R)\to R\to R} =$
$\lambda\, f : R \to R\,.\,\lambda\, a : R\,.\,\lim_{h\to 0} \frac{f\,(a+h) - f\, a}{h}$ (derivative at a point).

Def22: $\mathsf{deriv}_{(R\to R)\to(R\to R)} = \lambda\, f : R \to R\,.\,\lambda\, x : R\,.\,\mathsf{deriv\text{-}at}\, f\, x$ (derivative).

Def26: $\mathsf{integral}_{(R\to R)\to R\to R\to R} =$
$\lambda\, f : R \to R\,.\,\lambda\, a : R\,.\,\lambda\, b : R\,.$
$\lim_{n\to\infty} \sum_{i=1}^{n} (f\,(a + (\frac{b-a}{n} * i)) * \frac{b-a}{n})$ (definite integral).

Thm27: $\forall\, f, g : R \to R,\, a, b : R\,.$
$(\mathsf{cont\text{-}on\text{-}closed\text{-}int}\, f\, a\, b \wedge g = \lambda\, x : R\,.\,\int_a^x (f\, s)\, ds) \Rightarrow$
$((\forall\, x : R\,.\,a < x < b \Rightarrow \mathsf{deriv\text{-}at}\, g\, x = f\, x) \wedge$
$\mathsf{right\text{-}deriv\text{-}at}\, g\, a = f\, a \wedge$
$\mathsf{left\text{-}deriv\text{-}at}\, g\, b = f\, b)$ (fundamental theorem of calculus).

Thm28: $\forall\, f, g : R \to R,\, a, b : R\,.$
$(\mathsf{cont\text{-}on\text{-}closed\text{-}int}\, f\, a\, b \wedge f = \mathsf{deriv}\, g) \Rightarrow \int_a^b (f\, x)\, dx = g\, b - g\, a$
(corollary of fundamental theorem of calculus).

Figure 2: Some Calculus Definitions and Theorems from a Development of the Alonzo Theory COF

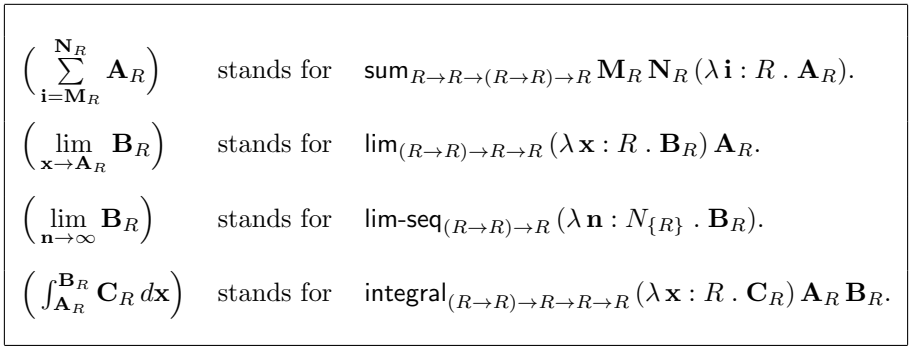

Figure 3: Some Notational Definitions for the Alonzo Theory COF

sions and LaTeX environments for presenting Alonzo modules. Other levels of software support are possible; see the discussion in Chapter 16 of [11]. Alonzo has not been implemented in a proof assistant, but since it is closely related to LUTINS, it could be implemented in much the same way that LUTINS is implemented in IMPS. Thus R4 is only partially satisfied now, but it could be fully satisfied with the addition of more levels of software support.

We have illustrated in [14] how a substantial body of mathematical knowledge associated with monoid theory can be formalized as a theory graph in Alonzo using the free approach. A robust theory graph (see Figure 1) is constructed using Alonzo modules. It contain 12 theories and 18 theory morphisms. Each of the theories is developed by defining new concepts and stating and proving theorems. For example, the theorem id-elt-is-unique mentioned above that says the identity element of a monoid is unique is stated as the sentence

$$\forall\, x : M\,.\,(\forall\, y : M\,.\,x \cdot y = y \cdot x = y) \Rightarrow x = \mathsf{e}$$

of the language of the theory MON presented using Alonzo's compact notation and proved from the axioms of MON. The proofs that validate the definitions and the theorems are all included in the appendix of [14] and are written in a traditional style, but some make use of the axioms, rules of inference, and metatheorems of Alonzo's formal proof system. The formalization is done with only the simplest level of software support, just the LaTeX macros and environments mentioned above.

Our Alonzo-based implementation of the free approach to formal mathematics demonstrates that all the benefits of formal mathematics, except possibly mechanically checked results, can be achieved when communication is the primary focus and only minimal software support is utilized. In other words, it demonstrates that there is a path to formal mathematics that will better serve most mathematics practitioners than the prove-everything-with-a-proof-assistant path.

7 Conclusion

Formal mathematics, i.e, mathematics done within the framework of a formal logic, offers huge benefits to the mathematics practitioner. 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 a formal logic and supporting software. The free approach to formal mathematics has the potential to enable a much larger portion of mathematics practitioners to reap the benefits of formal mathematics than the standard approach in which mathematics is done using a proof assistant and all details are formally proved and mechanically checked.

The free approach can be an attractive option for mathematics practitioners who have not employed formal mathematics due to the high cost of the standard approach. As we have illustrated in [14], the cost of employing the free approach can be much lower than doing everything with a proof assistant. The free approach can also be an attractive option for mathematics practitioners for whom communication is more important than certification. When the mathematics is well understand, as in mathematics education and routine applications, there is much less need to construct and mechanically check formal proofs. Another benefit of the free approach is that it can be a stepping stone to help mathematics practitioners cross the void between traditional mathematics and fully certified formal mathematics.

The great, and largely unrealized, promise of formal mathematics cannot be achieved by pursuing just the standard approach to formal mathematics. The free approach is needed in addition. Therefore, we strongly encourage the mathematics community to develop logics, software, and libraries of formal mathematical knowledge to support the free approach and to train mathematics practitioners to use them.

We look forward to a time when mathematics practitioners routinely express and reason about their ideas within a formal logic and high school and university mathematics students routinely build bodies of mathematical knowledge as theory graphs.

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] J. Avigad. Mathematics and the formal turn. *Bulletin of the American Mathematical Society*, 61:225–240, 2024.

[3] 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/.

[4] J. Blanchette and A. Mahboubi. *Proof Assistants and Their Applications in Mathematics and Computer Science*. Computer Science Foundations and Applied Logic. Birkhäuser/Springer, 2026. (To appear.).

[5] J. Carette, W. M. Farmer, M. Kohlhase, and F. Rabe. Big math and the one-brain barrier: The tetrapod model of mathematical knowledge. *The Mathematical Intelligencer*, 43(1):78–87, 2021.

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

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

[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 (HOA 1993)*, volume 816 of *Lecture Notes in Computer Science*, pages 96–123. Springer, 1994.

[9] 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.

[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*. Computer Science Foundations and Applied Logic. Birkhäuser/Springer, second edition, 2025.

[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] 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.

[15] 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.

[16] M. Iancu. *Towards Flexiformal Mathematics*. PhD thesis, Jacobs University Bremen, 2017.

[17] 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.

[18] 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.

[19] R. Nederpelt. N.G. de Bruijn (1918–2012) and his road to Automath, the earliest proof checker. *The Mathematical Intelligencer*, 34(4):4–11, 2012.

[20] A. N. Whitehead and B. Russell. *Principia Mathematica,* 3 volumes. Cambridge University Press, 1910–13.