

# Big Math and the One-Brain Barrier

## A Position Paper and Architecture Proposal

Jacques Carette · William M. Farmer ·  
Michael Kohlhase · Florian Rabe

the date of receipt and acceptance should be inserted later

**Abstract** Over the last decades, a class of important mathematical results have required an ever increasing amount of human effort to carry out. For some, the help of computers is now indispensable. We analyze the implications of this trend towards “big mathematics”, its relation to human cognition, and how machine support for big math can be organized.

The central contribution of this position paper is an information model for “doing mathematics”, which posits that humans very efficiently integrate four aspects: inference, computation, tabulation, and narration around a well-organized core of mathematical knowledge. The challenge for mathematical software systems is that these four aspects need to be integrated as well. We briefly survey the state of the art.

### 1 Introduction

In the last half decade we have seen mathematics tackle problems that lead to increasingly large developments: proofs, computations, data sets, and document collections. This trend has led to intense discussions about the nature of mathematics, ventilating questions like:

- i)* Is a proof that can only be verified with the help of a computer still a mathematical proof?
- ii)* Is a mathematical proofscape that exceeds what can be understood in detail by a single expert a legitimate justification of a mathematical result?
- iii)* Can a collection of mathematics papers — however big — adequately represent a large body of mathematical knowledge?

The first question was first raised by Appel and Haken’s proof of the four color conjecture [AH89] that in 400 pages of regular proof text reduced the problem

---

Computing and Software, McMaster University · Computing and Software, McMaster University · Computer Science, FAU Erlangen-Nürnberg · Computer Science, FAU Erlangen-Nürnberg

to approximately 2000 concrete configurations, which had to be checked by a computer. Later it arose again from Thomas Hales’ proof of the Kepler Conjecture [Hal05], which contained substantial algebraic computations as part of the proof. The second question is raised, e.g., for the classification of finite simple groups (CFSG), which comprises the work of a large group of mathematicians over decades and which has resisted dedicated efforts to even write down consistently — see the discussion below. The third question comes from the ongoing development of digital mathematics libraries (DMLs) — such as the huge collection of papers that constitute the proof of the CSFG — that fail to capture the abundant interconnections in mathematical knowledge that are needed to find and apply knowledge in the DMLs relevant to new problems.

Let us call such developments **Big Math** by analogy to the “big data/big everything” meme but also alluding to the “New Math” movement of the 1960s that aimed to extend mathematical education by changing how mathematics was taught in schools. In contrast to such Big Math, we will call mathematical developments that can be done with the proverbial “pen and paper” **Pen Math**. The emerging consensus of the mathematical community seems to be that, while the methods necessary for dealing with Big Math are rather problematic, the results so obtained are too important to forego by rejecting such methods. Thus we need to take on board Big Math methods and understand the underlying mechanisms and problems.

In what follows, we analyze the problems, survey possible solutions, and propose a unified, high-level model. We claim that computer support will be necessary for scaling mathematics, and that suitable and acceptable methods should be developed in a tight collaboration between mathematicians and computer scientists — indeed such method development is already under way, but needs to become more comprehensive and integrative.

We propose that all Big Math developments comprise four main aspects that need to be dealt with at scale:

- i) Inference:* deriving statements by *deduction* (i.e., proving), *abduction* (i.e., conjecture formation from best explanations), and *induction* (i.e., conjecture formation from examples).
- ii) Computation:* algorithmic manipulation and simplification of mathematical expressions and other representations of mathematical objects.
- iii) Tabulation:* generating, collecting, maintaining, and accessing collections of examples that suggest patterns and relations and allow testing of conjectures.
- iv) Narration:* bringing the results into a form that can be digested by humans, usually in mathematical documents like articles, books, or preprints, that expose the ideas in natural language but also in diagrams, tables, and simulations.

Computer support exists for all of these four aspects of Big Math, e.g.,

- i)* theorem provers like Isabelle, Coq, or Mizar;
- ii)* computer algebra systems like GAP, SageMath, Maple, or Mathematica; and

- iii*) mathematical data bases like the L-functions and Modular Forms Data Base (LMFDB) [Cre16, LMF] and the Online Encyclopedia of Integer Sequences (OEIS) [Slo03];
- iv*) online journals, mathematical information systems like zbMATH or MathSciNet, preprint servers like arXiv.org, or research-level help systems like MathOverflow.

While humans can easily integrate these four aspects and do that for all mathematical developments (large or otherwise), much research is still necessary into how such an integration can be achieved in software systems. We want to throw the spotlight on the integration problem to help start off research and development of systems that integrate all four aspects.

*Overview* In the next section we will discuss some of the state of the art in computer support for Big Math by way of high-profile mathematical developments, which we use as “case studies” for Big Math and present the issues and methods involved. In Section 3 we present a proposal for the integration of the four aspects plus a fifth aspect, organization, into a tetrapod structure and discuss that in more detail. Section 4 concludes this position paper.

## 2 Computer Support of Mathematics

The Classification of Finite Simple Groups (CFSG) is one of the seminal results of 20<sup>th</sup> century mathematics. Its usefulness and mathematical consequences give it a prominent status in group theory, similar to that of the fundamental theorem of arithmetic in number theory. The proof of the CFSG was constructed through the coordinated effort of a large community over a period of at least half a century; the last special cases were only completed in 2004.

The proof itself is spread over many dozens of contributing articles summing up to over 10,000 pages. As a consequence, work on collecting and simplifying the proof has been under way since 1985, and it is estimated that the emerging “second-generation proof” can be condensed to 5000 pages [WP].

It seems clear that the traditional method of “doing mathematics” which consists of well-trained, highly creative individuals deriving insights with “pen and paper”, reporting on them in community meetings, and publishing them in academic journals or monographs is reaching the natural limits posed by the amount of mathematical knowledge that can be held in a single human brain — we call this the “one-brain barrier<sup>1</sup>” (OBB).

---

<sup>1</sup> It might be argued that much mathematical research is now carried out in small groups instead of by individuals and that this should rather be called a “Small Group Brain-pool Barrier” (SGBB), but there are natural limits to collaboration on complex topics, as has been epitomized in the seminal 1975 book “*The Mythical Man-Month*” [Bro75]. The main result of this study is that adding members to a team can even slow down progress, because it induces communication overhead; the main solution proposed is to introduce individuals to the team that achieve “a detailed understanding of the whole project”; making the difference between an OBB and a SGBB gradual rather than fundamental.

We posit that transcending the OBB will be a crucial step towards future mathematics. The space of mathematical knowledge on this side of the OBB is bounded by the amount of time (and memory capacity) that a single individual can devote to learning the necessary scaffolding before reaching “the edge”. More specifically, the point at which a mathematical domain grows so much that it would take a candidate mathematician more than 25 years of work before being able to contribute new ideas, would likely signify the end of research in that domain. Indeed, we are seeing a gradual increase of proof size, which might point to a dearth of “interesting mathematics” inside the OBB. In fact, many of the important open conjectures (e.g., the Riemann conjecture) might be elusive because they are beyond the OBB.

There are two obvious ways around the OBB: 1. breakthroughs in the structural understanding of wide swaths of mathematics, so that the effort of learning about a particular domain can be greatly reduced; and 2. computer support. These are not mutually exclusive — and computer support may indeed enable such breakthroughs.

## 2.1 Computers vs. Humans in Mathematics

Humans and computers have dual performance characteristics: Humans excel at **vertical tasks** that involve deep and intricate manipulations, intuitions, and insights but limited amounts of data. In contrast, computers shine where large data volumes, high-speed processing, relentless precision, but shallow inference are required: **horizontal tasks**.

Vertical tasks include the exploration, conceptualization, and intuitive understanding of mathematical theories and the production of mathematical insights, conjectures, and proofs from existing mathematical knowledge.

Horizontal tasks include the verification of proofs, the processing of large lists of examples, counterexamples, and evidence, and information retrieval across large tracts of mathematical knowledge. Enlisting computers for horizontal tasks has been extremely successful — mathematicians routinely use computer algebra systems, sometimes to perform computations that have pushed the boundary of our knowledge. Other examples include data-driven projects like the L-functions and Modular Forms Data Base (LMFDB) [Cre16, LMF] which tries to facilitate Langland’s Program [Ber03] in number theory by collecting and curating objects like elliptic curves and their properties or the Online Encyclopedia of Integer Sequences [Slo03]. These already form “big computation” and “big tabulation” approaches, but they do not help in the case of the CFSG, which is more a “big proof”, and “big documents” problem though it involves the other two “big  $X$ ” aspects as well. Let us now consider the issues involved using a “big proof” effort.

## 2.2 A Computer Proof of the Odd-Order Theorem

In 2014, Georges Gonthier’s team presented a machine-checked proof of the Feit-Thompson Odd-Order Theorem (OOT) in the Coq theorem prover [GAA<sup>+</sup>13]. Even though Gonthier characterizes the OOT as the “*foothills before the Himalayas constituted by the CFSG*”, the Coq proof was a multi-person-decade endeavour and the Coq verification ran multiple hours on a current computer. It seems that computer supported formal proof has also hit some kind of OBB — formalization is a human endeavour after all. In this article, we want to analyze the kind of system we would need for transcending the OBB and pushing the boundaries of mathematical knowledge. But before we do, let us recap how proof verification works.

In a nutshell, the theorem and all the prerequisite background knowledge are expressed in a logic, i.e., a formal language  $\mathcal{L}$  equipped with a calculus  $\mathcal{C}$ . In  $\mathcal{L}$  the well-formedness of an expression is rigorously defined, so that it can be decided by running a program in finite time. A calculus  $\mathcal{C}$  over  $\mathcal{L}$  is a set of rules that transform  $\mathcal{L}$ -expressions into other  $\mathcal{L}$ -expressions, and a proof of a theorem  $t$  (an  $\mathcal{L}$ -expression) is a series of applications of  $\mathcal{C}$ -rules to  $\mathcal{C}$ -axioms that end in  $t$ . Crucially, the property of being a  $\mathcal{C}$ -proof can also be checked by running a program in finite time. Essentially, a  $\mathcal{C}$ -proof gives us absolute assurance that  $t$  is a theorem in  $\mathcal{C}$ . The cost of producing the proof is significant since the proof must be made explicit enough so that a machine can check it. Of course for a “big proof” development all lemmata that lead up to a “big theorem” are checked as well, so that — unlike in informal mathematics — we are sure that all pieces fit together exactly.

## 2.3 The Engineering Aspect: Inference and Knowledge Organization

But this is only half the story. To cope with the complexity of calculus-level proofs<sup>2</sup> it is much more convenient to use expressive logics — the “calculus of inductive constructions (CIC)” in the case of Coq — and programs that support the user in proof construction. A proof like the one for the OOT can have billions of steps and is only ever generated in memory of the Coq system during proof checking. Programs like the Coq proof assistant are engineering marvels, optimized to cope with such computational loads.

Optimization is also needed in the organization of the knowledge, if one is going to achieve the scale necessary for the OOT. Without care, we frequently end up re-proving similar lemmata, resulting in an exponential blow-up of the work required. To alleviate this problem, we follow the (informal) mathematical practice of generalizing results, and proving any lemma at “the most general level possible”. However, in the formal methods setting, we need to extend the logics, calculi, and proof construction machinery involved to take

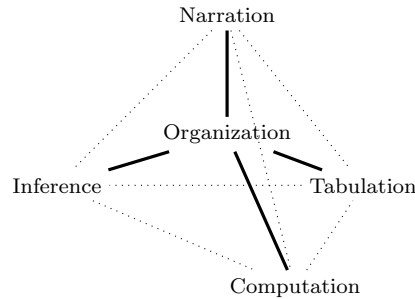
<sup>2</sup> Proofs in mathematics are expressed in a natural language: mathematical vernacular [dB94], a stylized form of English with interspersed mathematical formulae, tables, and diagrams.

modular development into account and optimize them accordingly. For the OOT, Gonthier and his team developed the method of “mathematical components” [MT] (akin to object-oriented modeling in programming languages, but better suited to mathematics) on top of CIC and used that to control the combinatorics of mathematical knowledge representation. Indeed, the development of the “library” of reusable intermediate results comprised about 80% of the development effort in the OOT proof [Gon17].

### 3 Five Aspects of Big Math Systems

We have seen two essential components of computer systems that can scale up “doing mathematics” to the Big Math level: 1. efficient and expressive theorem proving systems and 2. systems for organizing mathematical knowledge in a modular fashion. Already in the introduction, we mentioned the five basic *aspects* of mathematics:

- i) inference*, i.e., the acquisition of new knowledge from what is already known;
- ii) computation*, i.e., the algorithmic transformation of representations of mathematical objects into more readily comprehensible forms;
- iii) tabulation*, i.e., the creation of static, concrete data pertaining to mathematical objects and structures that can be readily stored, queried, and shared.
- iv) narration*, i.e., the human-oriented description of mathematical developments in natural language; and
- v) organization*, i.e., the modular organization of mathematical knowledge.



**Fig. 1** Five Aspects of Big Math Systems, a Tetrapod Structure

These aspects — their existence and importance to mathematics — should be rather uncontroversial. Figure 1 may help convey the part which is less discussed, and not less crucial: that they are tightly related. For a convenient representation in three dimensions, we choose to locate the organization aspect at the barycentre of the other four since they are all consumers and producers of mathematical knowledge. A four dimensional representation might be

more accurate, but less intuitive. We note that the names of the aspects are all derived from verbs describing aspects of “doing mathematics”: *i*) inferring mathematical knowledge, *ii*) computing representations of mathematical objects, *iii*) tabulating data about mathematical objects and structures, *iv*) narrating how mathematical results are produced, and *v*) organizing mathematical knowledge.

We will look at each aspect in turn using the CSFG and related efforts as guiding case studies and survey existing solutions with respect to the tetrapod structure from Figure 1.

### 3.1 Inference

We have already seen an important form of machine-supported mathematical inference: *deduction* via machine-verified proof. There are other forms: automated theorem provers can prove simple theorems by systematically searching for calculus-level proofs (usually for variants of first-order logic), model generators construct examples and counterexamples, and satisfiability solvers check for Boolean satisfiability. All of these can be used to systematically explore the space of mathematical knowledge and can thus constitute a horizontal complement to human facilities.

Other forms of inference yield plausible conclusions instead of provable facts: *abduction* (i.e., conjecture formation from best explanations) and *induction* (i.e., conjecture formation from examples). Machine-supported abduction and induction have been studied much less than machine-supported deduction, at least for producing formal mathematics. However, there is now a conference series [AIT] that studies the use of machine learning techniques in theorem proving.

One of the main problems with computer-supported inference for Big Math is that systems are (naturally and legitimately) specialized to a particular logic. For instance, interactive proof assistants like Coq and HOL Light have very expressive languages, whereas automated proof search is only possible for simpler logics where the combinatorial explosion of the proof space can be controlled. This makes inference systems very difficult to inter-operate, and thus all their libraries are silos of formal mathematical knowledge, thence leading to duplicated work and missed synergies — in analogy of the OBB we could conceptualize this as a *one-system barrier of formal systems*. There is some work on logic- and library-level interoperability — we speak of *logical pluralism* — using meta-logical frameworks (i.e., logics to represent logics and their relations) [Pfe01, KR16, Sai15]. We contend that this is an important prerequisite for organizing mathematical knowledge in Big Math.

### 3.2 Computation

Computer scientists have a very wide view of what *computation* is: be it  $\beta$  reduction (in the case of the lambda calculus), transitions and operations on a

tape (for Turing machines) or rewrites, all of these are somehow quite removed from what a mathematician thinks when “computing”. Here, for the sake of simplicity and familiarity, we’ll largely be concerned with *symbolic computation*, i.e. manipulation of expressions containing symbols that represent mathematical objects. Of course, there are myriad computations also done with various flavours of numbers — scientific computing, simulation/modelling, statistics and machine learning being eager consumers of vast quantities of numbers — but these represent a different flavor of computation.

In principle, mathematical computation can be performed by inference, e.g., by building a constructive proof that the sum  $2 + 2$  exists. But this is not how humans do it — they are wonderfully flexible in changing gears and switching between computational and inferential aspects of “doing mathematics”. Current systems in wide use have not achieved this flexibility, although Lean [dMKA<sup>+</sup>15], Agda [Nor07, Nor], and Idris [ID] are all making inroads. In any case, casting computation as inference and realizing inference as computation, are computationally intractable and, even if they were, human unfriendly. But gains have been made by this separation: computer algebra systems like Maple, Mathematica, or GAP can tackle computations that are many orders of magnitude larger than what humans can — often in mere milliseconds.

But computational systems face the same interoperability problems as inference systems do – open standards for formula representations on the web like OpenMath [Ope] or MathML [ABC<sup>+</sup>10] notwithstanding. Just to name a trivial, but symptomatic example: a particular dihedral group is called  $D_4$  in Sage and  $D_8$  in GAP due to differing conventions in the respective communities. More mathematically involved and therefore more difficult to fix is that most of the implementations of special functions in computer algebra systems differ in the underlying branch cuts [CDJW00]. Inference during computation would enable some of these problems to be fixed, but this has been sacrificed on the altar of computational efficiency. Another source of complexity is that today’s most feature-full symbolic computation systems are both closed-source and commercial, which makes integrating them into a system of trustable tools challenging.

Lastly, there is the question of acceptability of certain computations in proofs. In part, this derives from the difficulty of determining if a program written in most mainstream programming languages is actually correct, at least to the same level of rigor that other parts of mathematics are subjected to. Some of this problem can be alleviated by the use of more modern programming languages that have well understood operational and denotational semantics, thus putting them on a sound mathematical footing. Nevertheless it will remain that any result that requires thousands of hours of computation (or more) that cannot be verified by humans is likely to be doubted unless explicit steps are taken to insure its correctness. The Flyspeck project again provides an interesting case study [HAB<sup>+</sup>17].



### 3.3 Tabulation

If we look at the CFSG, then already the result contains an instance of tabulation: the collection of the 26 sporadic groups, which are concrete mathematical objects that can be represented by giving the generators, e.g., as particular matrices, which are in the end combinations of numbers. Even more importantly, many of the insights that led to the CFSG were only reached by constructing (examples of) particular groups, which were tabulated (as parts of journal articles and lists that were passed around). We also see this in other Big Math projects, e.g., Langland’s program of trying to relate Galois groups in algebraic number theory to automorphic forms and representation theory of algebraic groups over local fields and adèles. This is supported by the L-Functions and Modular Forms Database (LMFDB) [Cre16,LMF] which contains about 80 tables with representations of mathematical objects ranging from elliptic curves to Hecke fields – almost a terabyte of data all in all — which are used, e.g., to find patterns in the objects and their properties or to support or refute conjectures. The well-known Online Encyclopedia of Integer Sequences [Slo03] with over 300.000 sequences is another, well-known example. Finally — and more in sync with the CFSG — the Small Groups Library [SGL] contains more than 450 Million groups of small order of various classes. See [Ber] for a work in progress survey of math databases.

Unfortunately, such math databases are typically not integrated with the systems for mathematical inference, narration, or the knowledge organization level; and only weakly integrated into the systems for computation. Usually these math databases supply a human-oriented web interface, and if they offer an API access to the underlying database, they only offer database-level interaction (where, e.g., an elliptic curve is a quadruple of numbers, the last of which are encoded as decimal strings to work around size restrictions of the underlying database engine). What we would need for an integration in a Big Math system would be an API that supplies access to the mathematical objects — e.g., the representations of elliptic curves as they are treated in a proof assistant instead of quadruples of numbers — see [WKR17] for a discussion.

As usual, there are exceptions. The GAP Small Groups Library system and the LMFDB are notable examples: the first is deeply integrated with the GAP computer algebra system and the LMFDB provides a data base of almost a thousand “knowls”, informal representations of the mathematical knowledge underlying the LMFDB content.

### 3.4 Narration

Consider Figure 2, which shows the formalization of an intermediate result in the OOT in the foreground and the corresponding mathematical vernacular in the background. Even though great care has been taken to make the CIC text “legible”, i.e., short and suggestive of traditional mathematical notation,

# Recurrence

142 B. The Puig Subgroup

Proof. Again we use induction for (a). For  $n = 0$  we know (a) is true by hypothesis. Now suppose that  $n > 0$  and  $L(G)$  Then

$$L(G) \rightarrow L_{2n}(H).$$

Hence

$$L_{2n}(H) \subseteq L_{2n}(L(G)) = L_n(G).$$

Furthermore,

$$L_{2n}(H) \rightarrow L_{2n}(L_n(G)) = L(G) \subseteq H.$$

Thus

$$L(G) \subseteq L_{2n+1}(H).$$

Again, (b) follows from Lemma B.1(c).  $\square$

By Step 1 and Step 2 we can now conclude that  $L(G)$  is sired.  $\square$

Lemma B.3. Assume  $p$  is odd,  $G$  is solvable of odd order, and suppose that  $S$  is a Sylow  $p$ -subgroup of  $G$  and  $T = \mathcal{O}_p(G)$

$$L_n(S) \subseteq L_n(T) \subseteq L(T) \subseteq L(S).$$

Proof. First we show by induction on  $n$  that for all  $n \geq 0$ ,

$$(B.1) \quad L_{2n}(S) \subseteq L_{2n}(T) \subseteq L_{2n+1}(T) \subseteq L_{2n+1}(S).$$

For  $n = 0$  the statement reduces to

$$1 \subseteq T \subseteq S,$$

which is trivial.

Assume (B.1) holds for some  $n$ . Since  $L_{2n+1}(S) \rightarrow L_{2n+1}(T)$

$$(B.2) \quad L_{2n+1}(T) \rightarrow L_{2n+1}(S).$$

Now  $L_{2n+1}(T)$  is a normal  $p$ -subgroup of  $G$  and, by Lemma B.2,

$$L_{2n+1}(T) \supseteq C_T(L_{2n+1}(T)).$$

Thus, by (B.2) and Theorem A.5, (2)

$$L_{2n+1}(S) \subseteq T.$$

Hence, by (B.2),

$$(B.3) \quad L_{2n+1}(S) \subseteq L_T(L_{2n+1}(T)) = L_{2n+1}(T).$$

Consequently, by Lemma B.1(a),

$$(B.4) \quad L_{2n+1}(T) = L_T(L_{2n+1}(T)) \subseteq L_T(L_{2n+1}(S)) \subseteq L_S(L_{2n+1}(S)).$$

By Lemma B.1(b),

```

theorem Puig_center_normal : 'Z(L) <| G.
Proof.
have sLIST sLTS := pcore_Sylow_Puig_sub.
have sLIT: 'Z(L) \subset L(T) \subset L(S) by exact: Puig_sub_even_odd.
have sZY: 'Z(L) \subset Y.
rewrite subsetI andC subsetI ?cents ?orbT //.
suffices: 'C_S('L*(S)) \subset L(T).
by apply: subset_trans; rewrite setISS ?Puig_sub ?cents ?Puig_sub_even_odd.
apply: subset_trans (subset_trans sLIST sLIT).
by apply: sub_cent_Puig_at pS; rewrite double_gt0.
have chF: Y \char G = char_trans (center_Puig_char _) (pcore_char _).
have nsCY_G: 'C_G(Y) <| G by rewrite char_normal ?subcent_char ?char_refl.
have [C defC sCY_C nCG] := inv_quotientN nscY_G (pcore_normal p _).
have sLG: L \subset G by rewrite (subset_trans _ (pHall_sub sylS)) ?Puig_sub.
have nsL_nCS: L <| 'N_G(C :&: S).
have sYLis: Y \subset L*(S).
rewrite abelian_norm_Puig ?double_gt0 ?center_abelian //.
apply: normalis (pHall_sub sylS) (char_normal chY).
by rewrite subset // (subset_trans sLTS) ?Puig_sub.
have gYL: Y --> L := norm_abgenS sYLis (Puig_gen _).
have sLCS: L \subset C :&: S.
rewrite subsetI Puig_sub andBT.
rewrite -(quotientSGK _ sCY_C) ?(subset_trans sLG) ?normal_norm // -defC.
rewrite odd_abelian_gen_stable ?char_normal ?norm_abgen_pgroup //.
by rewrite (pgroups _ pT) ?subset // Puig_sub.
by rewrite (pgroups _ pS) ?Puig_sub.
rewrite -[L] (sub_Puig_eq _ sLCS) ?subsetIr //.
by rewrite (char_normal_trans (Puig_char _)) ?normalSG // subset // sSG orbT.
have sylCS: p.-Sylow(C (C :&: S) := Sylow_setI_normal nscG sylS).
have (defC) defC: 'C_G(Y) * (C :&: S) = C.
apply/eqP; rewrite eqSubset mulG_subG sCY_C subsetI //.
have nCY_C: C := C \subset Y.
by rewrite subsetI normG subsetI // sSG orbT.
exact: subset_trans (normal_sub nscG) (normal_norm nsCY_G).
rewrite -quotientSK // -defC /= -pseries1.
rewrite -(pseries_catr_id [[: p : nat_pred]]) (pseries_rcons_id [[:]]) /=.
rewrite pseries1 /= pseries1 defC pcore_sub_Hall // morphim_pHall //.
by rewrite subset ?nCY_C.
have defG: 'C_G(Y) * 'N_G(C :&: S) = G.
have sCS_N: C := C \subset N_G(C :&: S).
rewrite subsetI normG subsetI // sSG orbT.
by rewrite -(mulSGid sCS_N) mulG_a defC (Frattini_arg _ sylCS).
have ns2_N: 'Z(L) <| 'N_G(C :&: S) := char_normal_trans (center_char _) nsL_nCS.
rewrite /normal subsetI ?sLG // = -(1)defG mulG_subG /=.
rewrite cents_norm ?normal_norm // centsC.
by rewrite (subset_trans sZY) // centsC subsetIr.
Qed.

```

Fig. 2 Informal and Formal

there is still a significant language barrier for all but the members of the OOT development team.

Indeed mathematical tradition is completely different. Knowledge and proofs are presented in documents like journal articles, preprints, monographs, and talks for human consumption. While rigour and correctness are important concerns, the main emphasis is on communicating ideas, insights, intuitions, and inherent connections efficiently to colleagues well-versed in the particular topic or students who want to become that. As a consequence, more than half of the text of a typical mathematical document consists of introductions, motivations, recaps, remarks, outlooks, conclusions, and references. Even though the “packaging” of mathematical knowledge into documents leads to some duplication in the mathematical literature, it seems to be an efficient way of dealing with the OBB and thus a necessary overhead in scholarly communication.

In current proof assistants like Coq, the narration aspect is under-supported, even though tools like  $\text{\LaTeX}$  have revolutionized mathematical writing. Like in programming languages, source comments in the input logic are possible, but are not primary objects for the system, and are thus used sparingly. The main exception to this rule is the Isabelle system, which uses the formal framework to include marked text and turns the underlying ML into a deeply integrated document management system which allows formal islands in text [Wen18] recursively. Knuth’s *Literate Programming* idea [Knu92] has yet to take root in mathematics. Though it is worthwhile noting that one

of the earliest proof assistants, Automath [dB70], had extensive features for narration!

All other systems force users to learn the particular formal language first, before they can collaborate with the system. To add insult to injury, logics are notoriously bad for expressing the vague ideas and underspecified concepts that are characteristic in early phases of the development of mathematical theories and proofs, a task at which mathematical vernacular excels. As a consequence, the Flyspeck project [HAB<sup>+</sup>17] — a computer verification of Thomas Hales’ proof of the Kepler conjecture (KC) [Hal05] in the Hol Light and Isabelle proof assistants of comparable magnitude to the OOT proof effort — used a L<sup>A</sup>T<sub>E</sub>X-based book [Hal12] that refactored the original proof to orchestrate and track the formal proof via cross-references to the identifiers of all formal definitions, lemmata, and theorems.

Incidentally, the ongoing effort of establishing a second-generation proof of the CSFG has a similar book, consisting of seven volumes already published and five additional volumes that will be published in the future [WP].

### 3.5 Organization: Modular Representation of Mathematical Knowledge

In the discussion and survey of the four vertices of the tetrapod from Figure 1, we have seen that all aspects are based on represented knowledge (the *mathematical ontology*<sup>3</sup>, and that they can interact and interoperate through the ontology most effectively. And we have seen that a modular, redundancy-minimizing organization of the ontology is crucial for getting a handle on the inherent complexity of mathematical knowledge.

As inference-centered systems most directly interact with representations of mathematical knowledge, most of them feature some kind of modularity features to organize their libraries. This was pioneered in the IMPS system [FGT93] in the form of theories and theory interpretations, and has been continued e.g., in the Isabelle system (type classes and locales), both for simply typed higher-order logic as a base logic. In systems like Coq or Lean [dMKA<sup>+</sup>15] where we have dependent record types, theories and their morphisms can be encoded inside the logic (variants of CIC in both cases or homotopy type theory HoTT). Finally, the MMT system [RK13, MMT] systematically combines modular representation with a meta-logical framework in which the logics and logic-morphisms can be represented themselves, yielding a foundation-independent (bring-your-own-logic) framework for mathematical knowledge representation that can be used to establish system interoperability.

---

<sup>3</sup> Note that we will use the word “ontology” in the wider — and original — sense where it means “a *set of concepts and categories in a subject area or domain that shows their properties and the relations between them.*”, not just for specific technologies of the Semantic Web.

## 4 Conclusion

Using the classification of finite simple groups as an example of Big Math, we have diagnosed the “one-brain barrier” as a major impediment towards large-scale results in mathematics. We have seen that the seemingly obvious answer to this problem: “employ computer support” is not without problems and a barrier itself. We have proposed that computer-based mathematical assistants should have a tetrapodal structure, integrating inference, computation, tabulation, and narration via a central mathematical ontology. We claim that only with special consideration of all five aspects will mathematical software systems be able to render the support that is necessary for Big Math projects like the CFSG to go beyond the simple “proof certification” service rendered by “big-formal-proofs” like the OOT or Flyspeck.

Many of the systems we mention above, while being particularly focused on a particular task in mathematics, and excelling at it, also do integrate features of the “other” aspects. We mention a few, but a complete survey would be much too long.<sup>4</sup>

While the holistic conception of a Big Math tetrapod is new, the general sentiment that mathematical assistant systems must escape their native corner is implicitly understood in the mathematical software community. We have briefly surveyed current efforts and emerging system functionalities. A thorough review of the state of the art which would more clearly delineate the progress on the roadmap implicitly given by the tetrapod proposal is beyond the scope of this position paper, but is under active development by the authors for publication elsewhere.

We have observed the central place of the ontology in the proposed system functionality architecture, and we claim that such systems are best served by a global digital mathematical library (GDML), which serves as a pivotal point for integrating systems and system functionalities. Again, a discussion of this important resource is beyond the scope of this paper, but see [Far04] for a motivation and [KR16] for an avenue on how it could be realized by marshalling existing resources like the libraries of proof assistants.

We acknowledge the fact that our tetrapod proposal does not incorporate the fact that “doing mathematics” is a social process (at least for humans) and that for Big Math problems we will need mathematical social machines, i.e.,

A social machine is an environment comprising humans and technology interacting and producing outputs or action which would not be possible without both parties present. [BLF99]

We conjecture that the social machine aspect is one that will live quite comfortably on top of tetrapod-shaped mathematical software systems, and can indeed not fully function without all of its five aspects interacting well; see [Mar16, CMMR<sup>+</sup>17] for a discussion and further pointers to the literature.

---

<sup>4</sup> **Note to the reviewers:** We are working on a more complete survey and hope to have an arXiv.org version to reference here for the final version of this article.

Finally, we remark that of course the OBB is not particular to mathematics and affects all scientific and engineering disciplines, and we conjecture that similar, tetrapodal paradigms as the one advocated in this paper, will apply. Mathematics is a very good first test case for the design of knowledge-based systems, since the knowledge structures and algorithms are so overt.

*Acknowledgments* The authors gratefully acknowledge fruitful discussions with many of our colleagues in the mechanized mathematics community. The work reported here was supported by the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541), DFG project RA-18723-1 OAF, NSERC Grants RGPIN-2018-05812 and RGPIN-5100-2015.

## References

- ABC<sup>+</sup>10. Ron Ausbrooks, Stephen Buswell, David Carlisle, Giorgi Chavchanidze, Stéphane Dalmas, Stan Devitt, Angel Diaz, Sam Dooley, Roger Hunter, Patrick Ion, Michael Kohlbase, Azzeddine Lazrek, Paul Libbrecht, Bruce Miller, Robert Miner, Murray Sargent, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 3.0. W3C Recommendation, World Wide Web Consortium (W3C), 2010.
- AH89. Kenneth Appel and Wolfgang Haken. *Every Planar Map Is Four Colorable (Contemporary Mathematics)*. American Mathematical Society, 1989.
- AIT. AITP: Conference on artificial intelligence and theorem proving. <http://aitp-conference.org/>.
- Ber. Katja Berčič. Math databases wiki. <https://github.com/MathHubInfo/Documentation/wiki/Math-Databases>.
- Ber03. Steve Bernstein, Joseph Gelbart, editor. *An Introduction to the Langlands Program*. Birkhäuser, 2003.
- BLF99. Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Harper, 1999.
- Bro75. Frederick Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975.
- CDJW00. R.M. Corless, J.H. Davenport, D.J. Jeffrey, and S.M. Watt. According to Abramowitz and Stegun. *SIGSAM Bulletin* 2, 34:58–65, 2000.
- CMMR<sup>+</sup>17. Joseph Corneli, Ursula Martin, Dave Murray-Rust, Alison Pease, Raymond Puzio, and Gabriela Rino Nesin. Modelling the way mathematics is actually done. In *Proceedings of the 5th ACM SIGPLAN International Workshop on Functional Art, Music, Modeling, and Design*, FARM 2017, pages 10–19, New York, NY, USA, 2017. ACM.
- Cre16. John Cremona. The L-functions and modular forms database project. *Foundations of Computational Mathematics*, 16(6):1541–1553, 2016.
- dB70. Nicolaas Govert de Bruijn. The mathematical language AUTOMATH, its usage and some of its extensions. In *Symposium on Automatic Demonstration*, number 125 in LNM, pages 29–61. Springer Verlag, 1970.
- dB94. Nicolaas Govert de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In R. P Nederpelt, J. H. Geuvers, and R. C. de Vrijer, editors, *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, pages 865 – 935. Elsevier, 1994.
- dMKA<sup>+</sup>15. Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015.

- Far04. William M. Farmer. MKM: A new interdisciplinary field of research. *Bulletin of the ACM Special Interest Group on Symbolic and Automated Mathematics (SIGSAM)*, 38(2):47–52, 2004.
- FGT93. William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11(2):213–248, October 1993.
- GAA<sup>+</sup>13. 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 Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, volume 7998 of *LNCS*, pages 163–179. Springer, 2013.
- Gon17. Georges Gonthier, 2017. private communication.
- HAB<sup>+</sup>17. Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Le Truong Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, Quang Truong Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, Thi Hoai An TA, Nam Trung Tran, Thi Diep Trieu, Josef Urban, Ky Vu, and Roland Zumkeller. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 5, 2017.
- Hal05. Thomas C. Hales. A Proof of the Kepler Conjecture. *Annals of Mathematics. Second Series*, 162(3):1065–1185, 2005.
- Hal12. Thomas Hales. *Dense Sphere Packings*. Number 400 in London Mathematical Society Lecture Note Series. Cambridge University Press, 2012.
- ID. Documentation for the idris language. <http://docs.idris-lang.org/en/latest/>.
- Knu92. Donald E. Knuth. *Literate Programming*. The University of Chicago Press, 1992.
- KR16. Michael Kohlhase and Florian Rabe. QED reloaded: Towards a pluralistic formal library of mathematical knowledge. *Journal of Formalized Reasoning*, 9(1):201–234, 2016.
- LMF. The L-functions and modular forms database. <http://www.lmfdb.org>. [Online; accessed 27 August 2016].
- Mar16. Ursula Martin. Computational logic and the social. *Journal of Logic and Computation*, 26:467–477, 2016.
- MMT. MMT – language and system for the uniform representation of knowledge. project web site at <https://uniformal.github.io/>.
- MT. Assia Mahboubi and Enrico Tassi. Mathematical components. <https://math-comp.github.io/mcb/book.pdf>. online book.
- Nor. Ulf Norell. The Agda Wiki. <http://wiki.portal.chalmers.se/agda>.
- Nor07. Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology and Göteborg University, 2007.
- Ope. OpenMath Home. <http://www.openmath.org>. seen May 2009.
- Pfe01. Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I and II. Elsevier Science and MIT Press, 2001.
- RK13. Florian Rabe and Michael Kohlhase. A scalable module system. *Information & Computation*, 0(230):1–54, 2013.
- Sai15. Ronan Saillard. *Typechecking in the  $\lambda\Pi$ -Calculus Modulo: Theory and Practice*. PhD thesis, École nationale supérieure des mines de Paris, 2015.
- SGL. The small groups library. [http://www.icm.tu-bs.de/ag\\_algebra/software/small/small.html](http://www.icm.tu-bs.de/ag_algebra/software/small/small.html).
- Slo03. Neil J. A. Sloane. The on-line encyclopedia of integer sequences. *Notices of the AMS*, 50(8):912, 2003.
- Wen18. Makarius Wenzel. Isabelle/jEdit as IDE for domain-specific formal languages and informal text documents. In Paolo Masci, Rosemary Monahan, and Virgile Prevosto, editors, *F-IDE 2018 – 4th Workshop on Formal Integrated Development Environment*, 2018.

- 
- WKR17. Tom Wiesing, Michael Kohlhase, and Florian Rabe. Virtual theories – a uniform interface to mathematical knowledge bases. In Johannes Blömer, Temur Kutsia, and Dimitris Simos, editors, *MACIS 2017: Seventh International Conference on Mathematical Aspects of Computer and Information Sciences*, number 10693 in LNCS, pages 243–257. Springer Verlag, 2017.
- WP. Wikipedia: Classification of finite simple groups – second-generation classification. [https://en.wikipedia.org/wiki/Classification\\_of\\_finite\\_simple\\_groups#Second-generation\\_classification](https://en.wikipedia.org/wiki/Classification_of_finite_simple_groups#Second-generation_classification).