The Evolution of Proof Assistant Math Repositories

Mahsa Bazzaz
Northeastern University
Boston, Massachusetts, USA

Minsung Cho
Northeastern University
Boston, Massachusetts, USA

Gwen Lincroft
Northeastern University
Boston, Massachusetts, USA

1 INTRODUCTION

Proof assistants (also known as interactive theorem provers) are programming languages with the primary purpose of providing computational meaning to mathematical proof through code [2]. The first proof assistants, developed over fifty years ago, were designed to help reason about the correctness of software and hardware through mathematical abstraction. However, as proof assistants became more expressive and user-friendly, they have gained traction in numerous fields, including formal methods, artificial intelligence, and computer science education [3–5].

However, over the past 20 years, proof assistants garnered attention from communities beyond computer science. In particular, the mathematics community took interests in proof assistants with their potential to bridge software development and mathematical proof. This potential has been realized through multiple open-source, well-documented libraries of formalized proofs as programs. Although there have been many such libraries over time, the predominant ones today are written in three languages: Lean, Coq, and Isabelle/HOL.

Lean has been the focal point of math library development in the past five years, gaining the attention of prominent mathematicians by demonstrating its ability to prove research-level mathematics [1]. Its math library, known as `mathlib` , has been open-sourced on GitHub under an Apache license. It has extremely rigid conventions for contributing, including guidelines on style, naming, commit messages, and pull request labeling.

Coq is one of the most widely-used proof assistants in computer science today. However, its math library is not centralized like `mathlib` . Instead, it offers a *loosely federated* list of open-source GitHub repositories. Many of these repositories are under the same GitHub organizations, for example `math-comp` or `coq-community`, but not all of them are and their interdependence is not obvious.

Isabelle/HOL is another major proof assistant that offers an extremely strong general automation, which the other proof assistants lack. Although its math library is open source, it is not hosted natively on GitHub and external contribution is not done through GitHub's standard means. In particular, contribution is vetted behind closed doors by academics and is not done through a standard pull request-merge system. Furthermore, there is no centralized repository á la `mathlib` and most are hosted on the website *Archive of Formal Proofs*, where the proofs lack version control but give a detailed account of authors, dependencies, and proof techniques.

It is clear that each of these proof assistants and their math libraries have different software development practices. However, they all attempt to demonstrate the power of proof assistants by formalizing known mathematics through code. This project aims to compare and contrast the evolution of these three main repositories through a software engineering lens. In particular, for this project we have two main research questions:

(1) What mathematical theorems did different formalized mathematics libraries prove? How has this developed over time? How have the proofs changed?
(2) How has the popularity of contributing to theorem provers over time changed? Are there any factors we need to take into account to evaluate proof assistant software different than traditional software?

2 PROJECT RESULTS

2.1 Methodology

To investigate research problem 1, we used Freek Wiedijk's *list of 100 theorems*[6], an online list of "top 100" mathematical theorems that proof assistants to attempt to formalize, which we will refer to as the *100 Theorems list*.

We first gathered exactly what proportion of the 100 theorems have been formalized fully in the respective proof assistants. Then, we used `SimpleGit` to collect data of relevant commits to the file containing each theorem. For Lean, we collected data only from the *mathlib* repository, where all the proofs were held. For Coq, we looked at 30 different repositories where the proofs were held. For Isabelle, we either looked at the *mirror-afp-2022* repository or had to take data directly from the *Archive of Formal Proofs* website.

Using the data, we selected the following information:

- when each theorem was proven,
- which theorems required the most commits
- and what type of commits–code refactoring, dependency updating, or simply writing new supporting definitions and lemmas–were typically made.

To investigate the research problem 2, we used `Octokit` to take data on pull requests and issues. However, this was only doable on Lean and Coq, as Isabelle does not have any pull requests or issues. We again used `SimpleGit` to collect data on commits. Using the data, we selected the following information:

- the proportion of pull requests merged to the main or master branch, closed, and still open,
- the proportion of issues opened or closed,
- the rate of growth of the repositories, measured as

$$\text{\# of commits} + \text{\# of opened PRs} + \text{\# of opened issues}, \tag{1}$$

- the general type of pull request and issues seen in math repositories, such as suggestions, bugfixes, or new additions.

With this, we present our results.

2.2 Results for research problem 1

We first observed that although Lean is the focal point of math library development in the past five years, Isabelle actually has the most theorems proved on the 100 Theorems list, with 87. Coq follows with 79, and Lean is actually the least populated with 76.

Thus, in objective number of theorems proved, Lean is actually not the dominant proof assistant for formalized math.

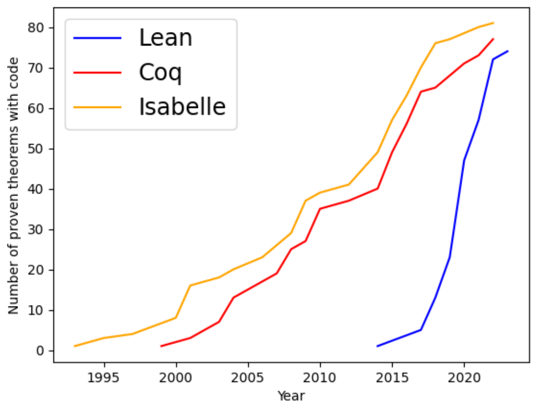


Figure 1: Number of theorems on the 100 Theorems list proven via Lean/Coq/Isabelle over time.

However, Figure 1 demonstrates that Lean is catching up fast. Isabelle and Coq's main advantage is the longer history of development, which lends itself easily to more code. Furthermore, we see that Lean's `mathlib` is growing significantly faster in the past five years than both Coq and Isabelle's libraries, with contributions even as recent as this year, which both Coq and Isabelle lack.

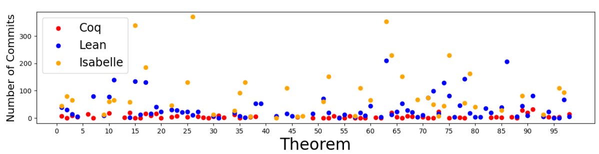
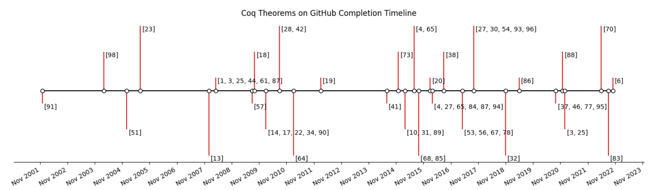


Figure 2: Number of commits per proven theorem on the *100 Theorems* list.

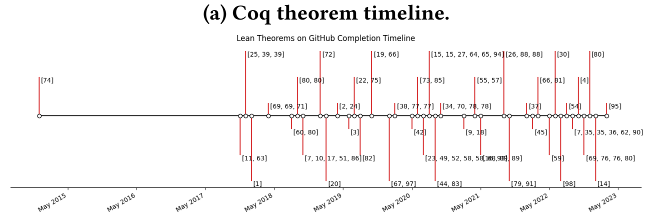
Figure 2 demonstrates that, although Coq has more theorems proven than Lean, per theorem it only has few commits. On the other hand, Isabelle seems to have many theorems with hundreds of commits to the source file. In general, we see little correlation between the different proof assistants and the number of commits to prove each theorem on the 100 Theorems list.

We see in Figure 3 that there is also little correlation between the different proof assistants and the order in which the theorems were proven. However, we see a common pattern in all three proof assistants: there are times when several theorems are proven in a very short span of time. We believe this is due to similarities in the definitions and proof techniques used in the theorems rather than pure coincidence.

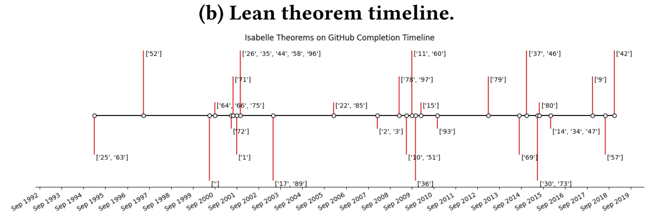
In general, the commit contents were mostly proofs of the theorem and related mathematics. We observed that often the theorems themselves are only consequences of an underlying theory, and not a standalone goal. This can be likened to mathematics, where usually theorems are consequences of studying an underlying theory and not a goal to build towards. We believe that this is also the reason why there is little correlation in the order in which theorems were proven. Besides proofs, other commit contents included dependency updates, refactoring of code, and comments to explain the proofs in English. We leave it to future work for a more sophisticated analysis of commit contents via code differences.



(a) Coq theorem timeline.



(b) Lean theorem timeline.



(c) Isabelle theorem timeline.

Figure 3: Theorems proved by Coq/Lean/Isabelle in chronological order.

2.3 Results for research problem 2

We observed in Figure 1 that Lean, in the context of math formalization, has exploded in growth over the past five years. We attempt to characterize this growth using different metrics derivable from looking at the history of Git activity for the proof assistants and their math libraries.

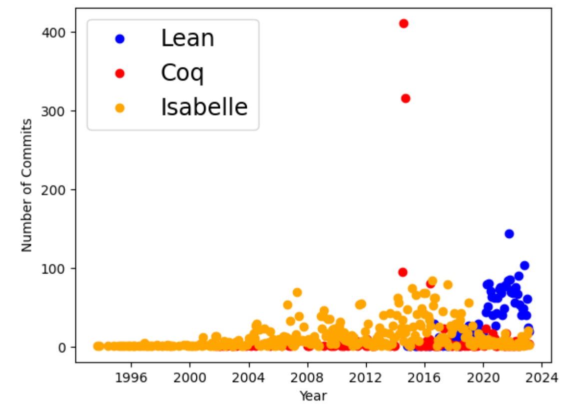


Figure 4: Number of commits per year for Lean, Coq, and Isabelle.

We see in Figure 4 that in the past five years Lean has significantly more commits to its main repositories than both Coq and Isabelle. Before that, Isabelle had a commanding lead, although its activity seems to have diminished with the emergence of Lean. The slight increase in Isabelle activity in 2012-2016 corresponds to a large growth in proven theorems in Figure 1 over the same time frame.

Figure 4 also shows two outlier points for Coq: there are two points, occuring at August and September 2014, where the number of commits surpasses 300. Most of these commits were from one user and their solo project to prove Puiseux's Theorem, one of the theorems on the 100 Theorems list. Interestingly, none of the commit messages for these commits seem to have relevant information–the messages are simply –.

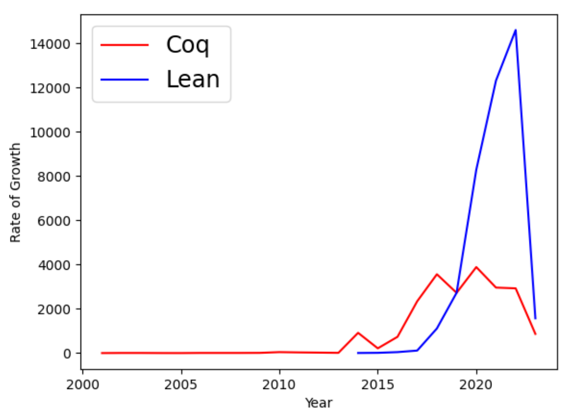
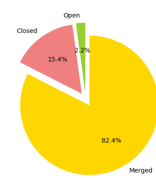
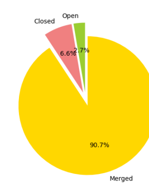


Figure 5: Rate of growth for Coq and Lean via Equation 1

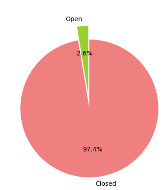
Figure 5, which shows code growth by Equation 1 per year, reaffirms similar conclusions to Figure 4. However, we noticed that Isabelle does not have a standard system for pull requests and issues. Instead, any changes in its library must be done through a form submitted to TU Munich, where Isabelle development is centered at. As such, the following analysis of pull request and issue will be on Lean and Coq exclusively.



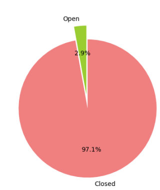
(a) Coq pull requests.



(b) Lean pull requests.



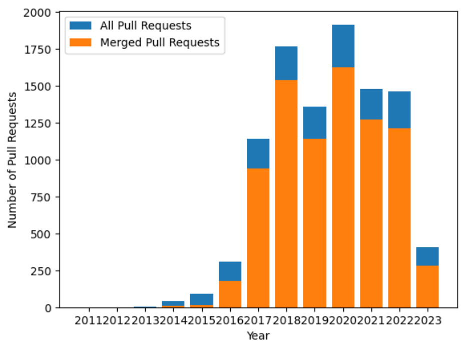
(c) Coq issues.



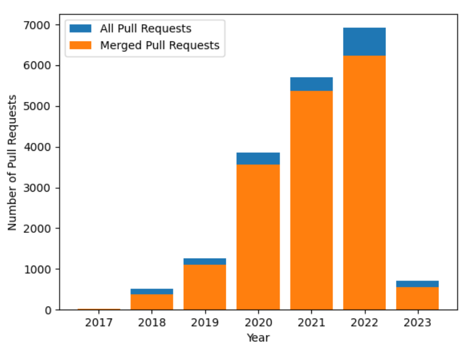
(d) Lean issues

Figure 6: The state of pull requests and issues for Coq and Lean. Red=closed, Green=open, Yellow=merged.

Figure 6 demonstrates that how Coq and Lean handle issues is relatively similar by proportion. It is interesting to note that although Lean arguably has more stringent style and pull request guidelines than Coq, its proportion of merged pull requests is slightly more than Coq. Indeed, Figure 7 reinforces the conclusion that although Coq and Lean are different in many ways, they follow similar open source guidelines.



(a) Coq.



(b) Lean.

Figure 7: All vs. merged pull requests for Coq and Lean.

3 PROJECT CHALLENGES

3.1 Limits in project scope

The original goal of this project was to investigate proof assistants from four different math libraries in addition to Lean, Coq, and Isabelle. However, this proved challenging as many proof assistants predate GitHub and modern software engineering practice, including style and version control. In particular, we observed that some prominent theorems according to the 100 Theorems list, such as Mizar and Metamath, only host proofs on the proof assistant website itself. We chose to limit the scope of our project to the most popular math libraries hosted on GitHub. Additionally, each of these libraries was originally hosted and maintained outside of GitHub before being migrated or mirrored. Some repositories, like Coq language, contain the entire pre-GitHub repository history and others, like Lean, do not. This poses an additional challenge towards collecting complete data and making comparisons between repositories. We chose to not look outside of GitHub for additional history for each library studied. However, we do believe that scraping the websites for relevant statistics about the proofs is worthwhile; we leave it to future work beyond the time frame of this project.

An additional original goal of the project was to investigate the use cases of each proof assistant over time by examining what type of projects on GitHub were written in the language. However, this was difficult to do considering the time constraint. Not only does the GitHub rate limit affect the speed at which we can gather information about individual projects, but it would also be time-consuming to parse the information and categorize it into appropriate metrics for use cases. As such, we also delegate this goal to future work.

3.2 Challenges with data collection

Our original methodological plan was to use GHTorrent to collect the github metadata for each math library explored. However, GHTorrent was no longer maintained at the start of this project and we were instead forced to use Octokit requests to the GitHub Rest API for data mining. Exploring pull requests and issues with the Rest API required additional queries to be made to the API as metadata results were often in the form of API links. The rate limit of 1000 requests per hour imposed by the Rest API slowed progress significantly as Coq and Lean both had thousands of pull requests and issues. Our final mining implementation is able to acquire in-depth data on pull requests, issues, and commits from a repository, but may take multiple days to run.

Our GitHub mining tool also originally assumed that each math library would use built-in Github conventions for tracking the status of pull requests. However, this was not the case for the Lean library, in which merges of all pull requests were handled by BORS, that merged code externally before committing it to the main branch, closing the pull request, and updating the title of the pull request to include “Merged by Bors”. The impact of BORS on pull request mining was that the status displayed by GitHub (merged, closed, or open) was not trustworthy. In order to discover which pull requests had actually been merged into the library, we had to check each pull request for BORS activity.

An additional challenge regarding the collection of the 100 theorems in Isabelle was that many of the links pointing to proofs in Isabelle were broken. We overcame this challenge by manually finding the source files for each of the 100 theorems proved in Isabelle.

REFERENCES

[1] Kevin Buzzard. 2020. Liquid tensor experiment. https://xenaproject.wordpress.com/2020/12/05/liquid-tensor-experiment/

[2] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. 2015. The Lean theorem prover (system description). In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*. Springer, 378–388.

[3] Cordell Green. 1981. Application of theorem proving to problem solving. In *Readings in Artificial Intelligence*. Elsevier, 202–222.

[4] Warren A Hunt Jr. 1994. *FM8501: A verified microprocessor.* Springer.

[5] Maria Knobelsdorf, Christiane Frede, Sebastian Böhne, and Christoph Kreitz. 2017. Theorem provers as a learning tool in theory of computation. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. 83–92.

[6] Freek Wiedijk. [n.d.]. Formalizing 100 Theorems. https://www.cs.ru.nl/~freek/100/