Vrije Universiteit Amsterdam

Bachelor Thesis

Disseminating a Library of Formal Proofs for Flexible and Persistent Searching

Author: Polina Boneva

1st supervisor: *Robert Y. Lewis*
2nd reader: *Gabriel Ebner*

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science

June 1, 2021

Abstract

Mathlib is a continuously growing library of theorems and their proofs, axioms, definitions and other instances of formalized mathematics (The mathlib Community, 2020). Mathlib is readable both by the human eye and Lean, the programming language and proof-assistant system that does the automated job of reading and validating mathematical concepts (Avigad, De Moura, & Kong, 2021). A problem the current space holds is that there are no automated capabilities of Lean to organize the global structure of the written proofs. Thus, the correct purpose and place of each more complex declaration included in the library's architecture must be checked directly by experts who have other research and work to do (van Doorn et al., 2020).

The complete contents of mathlib are translated to a documentation in an HTML format (van Doorn, F., Ebner, G., & Lewis, R. Y., 2020). To help to traverse the data in mathlib, we have created an index that represents some elements from the library partially, just enough for it to allow a quick search in the most easily recognizable elements to both experts and novices. These elements include the name of the collection the theorem is in, the theorem's declarative primary name, and its human-readable description. Ultimately, flexibility is just as important when it comes to the exploration of mathlib, and for that, we have included two extra parts of each declaration of a theorem: its attributes and kind. Both serve as filtering mechanisms to distinguish between all the described types of mathematical concepts in the library.

Introduction

Mathlib is currently one of the main places where human knowledge in theoretical mathematics is being translated for the computer to keep, use, and proofread automatically, whilst being written in a programming language designed specifically for those purposes - Lean. Naturally, it is becoming increasingly difficult to traverse the data with ease, which is only becoming more noticeable because of its ever-growing usability and user load. Ten months ago, there were a total of over 60 000 declarations of all kinds: theorems, axioms, definitions, constants, structures, constructors, etc. Now their count is over 85 000. Experts' manual work on proofreading new theorems inevitably calls for quick navigation in the realm of existing data of pure mathematics (van Doorn et al., 2020). As this project is entirely dependent on the community of people who care for it and use it in their work, they are the ones who have created scripts in both Lean and Python. These scripts serve to take the entire contents of mathlib and translate them in a unified structure in a single JSON object used to create HTML files for displaying the data with helpful color-coding on a per-module basis, as they are in the library itself (van Doorn et al., 2020). It is a perfect reflection of the written declarations in mathlib (Lean-Prover Community, n.d.).

The continuous integration of mathlib into its documentation involves collecting all of its components and differentiating between them in order to have a syntax that covers each part of the data concisely. User's needs, however, are growing with their number: to modify, extend, query, and maintain the source as best as possible so that the library proves sustainable in the long run (van Doorn et al., 2020). The ever-expanding corpus of mathematical knowledge is gradually being translated into machine languages, all of which do their best to allow for quick traversal of the whole system, even with the growing complexity in included declarations. The following paper will describe a way to approach a collection of formal proofs so that one stays on top of the information and understands its structure so they can use it in all ways imaginable. This will be achieved by extracting the most necessary parts of the data for easy exploration of the whole library and preparing for future modification of any variety.

Related Literature

The previous search offered a string-based lookup of the query into the names of each declaration. Only the 20 top results are shown, which means that any others cannot be reached in any way. It was a good beginning and served its purpose of suggesting what can be done in the future. The current search also searches the syntax of the indexed data directly. Unlike SErAPIS (Stathopoulos, Paulson, & Koutsoukou-Argyraki, 2020), it does not offer any semantic suggestions. It does not translate mathematical concepts to similar concepts of a related kind. Still, it does provide more functionality in terms of the type of data that can be searched, as well as filtering and prioritization. It does, much like FindFacts, rely on a ready-made index which is to be accessed by the loaded documentation only once (Huch, F., & Krauss, A., 2020). It cannot work offline at the moment, even though the way the index is made by a pre-processing script, it could be made to function without an internet connection, as one is not needed when one makes no use of a separate server. Currently, we have seen no need to make the index offline, but should there be such a request, it will not be difficult to redirect the automated creation of searchable data towards each computer that needs access to it without an internet connection.

Some problems we have encountered are also described by the researchers and developers of Isabelle, another great mathematical library that aims to reach the goals of flexible querying with a semantic translation of mathematical concepts in the full range of data they cover (Stathopoulos, Y., Koutsoukou-Argyraki, A., & Paulson, L., 2019). The biggest challenge in terms of syntax is the look and feel of formal language, as it is described by various programming languages to be read by machines (Stathopoulos, Koutsoukou-argyraki, & Lawrence, 2019). It cannot be searched easily because the symbols are not easily accessible on keyboards, unlike Latin letters. This makes the distinction between them a challenge more suited for creating two different search mechanisms: one for the search of declarative proofs with mathematical symbols and one for the search of theorem statements, descriptions, and names where the English language serves a good purpose. Last but not least, each created functionality should be extensively tested by the users of the system in an ordered and well-defined manner (Stathopoulos et al., 2019).

Work Process

The mathlib library is at its beginning stage. It has yet to reveal its full potential for automated self-checking of proofs, building proofs by itself when needed and designing its own structure as it evolves, even though its size grows exponentially and the importance of the data gathered until now cannot be understated (Hartnett, 2020). For the past ten months, its size has grown by about 30% of what it used to be. This increase in size can only serve as a sign that the people interested in its usefulness and reliability are growing fond of it. The recent escalating development of libraries like mathlib brings this future to us even faster. The sheer hope people hold in their hearts and minds is what drives them forward to learn a new language, be it Lean, Coq or Isabelle, to describe their hand-written knowledge for a machine to store and read. It can then be used for our current and future not yet imagined purposes (Hartnett, 2020). All this leads us to the natural question of: how can people find their way in the ever-growing library of formal proofs easily, effectively, consistently and in the most useful way for all types of human knowledge: novice, expert, mid and even curious non-mathematicians who might one day be part of it all.

The maintainers of the system are also its users. Still, not all of its users are its maintainers, and all of them do not have this as the primary job but use it as a mere tool to do their actual job of discovering, proving, and extending mathematical knowledge for the world (van Doorn et al., 2020). Hence, their needs are not equal. It can be pretty challenging to find their common denominator and work in its direction so that everyone is at least somewhat satisfied, as opposed to some people having no use of what others find entirely in their lane of work.

Communal Application and Goals

The core components of amazing documentation include a flexible, easily extensible, all-covering search functionality which offers filtering on demand by the user, useful and maybe even changeable by the user prioritization, and automated self-development, meaning it can handle data expansion without breaking and needing manual readjustment. In the same fashion, it should offer sleek design, color-coding of results that match the color-coding of the documentation and highlights on the matched terms in the full text of the results. It should also be fast and offer a small set of results at first search, as

Figure 1

A Declaration in Mathlib

```
{
   Args,
   Attributes,
   Constructors,
   Doc_string,
   Equations,
   Filename,
   Is_meta,
   Kind,
   Line,
   Name,
   Structure_fields,
   Type
}
```

Note. A sample set of the keys a declaration in Mathlib is comprised of. Kind and Attributes are required for filtering the data.

Figure 2

A Tactic in Mathlib

```
{
   Category,
   Decl_names,
   Description,
   Import,
   Name,
   Tags
}
```

well as the full scope of results on demand. Searching should happen easily in a search box, filtering should be obvious and easy to use, and one should be able to combine filters if needed. Each of the components of a good search: good indexing, query and data refinement in the act of searching, filtering, prioritization, and user experience, can be improved on its own in a wide variety of ways, but we will mention only some of them which according to us are to be developed as part of the next stage of enhancements. Likewise, when it is done, it will become apparent what should be covered in its next step, and so on.

Because this library is subject to a communal effort by people who have great use of it, it is natural that their recommendations and needs should be covered extensively. They, however, have many other things to do and so can only give feedback with regards to their currently existing needs and the capabilities of each ready-made functionality (van Doorn, Ebner, & Lewis, 2020). Both transform with time, so this ongoing process has only just started. In the light of this here searching mechanism to be helpful in the future, we also have in mind that our work here can be used not only in Lean-based libraries but can manage to divide and collect all kinds of mathematical knowledge into atoms of data which can be used as a base in all sorts of libraries of extensive theoretical and practical knowledge, as long as it is described in a similar manner as the JSON we extract from mathlib.

The mechanism for traversing and mapping the library has to take care of covering a wide range of users because this will only serve for the better of the whole project (van Doorn et al., 2020). To divide and conquer the library in such a way that the atoms, formed after breaking up its contents into most fundamental pieces of value, cover all types of data structures together seems a daunting task at first. One has to start from the beginning and answer the question: what is mathlib comprised of now, and what are its uses for the mathematicians and their development? On the one hand, the theorems, axioms, constants, constructors, structures, and definitions are all constituted from the data described in Fig. 1. On the other hand, there are the tactics in mathlib that are the beginning of automated theorem processing by Lean: they can self-sufficiently suggest ways to prove a theorem as one is writing it (Avigad et al., 2021). Their structure in the single JSON export from mathlab's complete write out of its contents – let's call it the JSON, as it is the only one and currently the most crucial, is comprised of the elements described in Fig. 2.

We have aimed at creating a basis for further searching developments where the full data of mathlib is taken into account. The main focus of our work is to create a single JSON of five items, three of which are found in both searchable structures and two - in the more extensive one. To be more exact, the pieces of information we have included in the searchable data structure carry the most fundamental value of the largest piece of information described in mathlib, which means that less extensive data in mathlib may not carry values for each item in the index that is used for searching. However, this is not a problem because wherever information is missing, it is simply left empty, and it does not affect the search capabilities of the query at all. The lowest common denominator happens to be of the highest demanding structure, as expected.

Work Method

A few things need to be taken into consideration before deciding how to build the base of the index for searching. First and foremost, we need to answer the "What?" of the goals the searching mechanism has for the community and its aims for the future. Next, we need to answer the "Why?" for every part of the fundamental structures required for the index that will be used for traversing mathlib's contents. Lastly, we should answer the "How?" of the goals accomplishment journey and reaching the destination most flexibly and understandably so that future development can be done by others as well, not just by the people who have started the process.

Fundamental Structures

In order to accomplish our goals, we will begin with extracting the core of what constitutes a theorem and its proof and developing a stand-alone categorization of the components, which is

Figure 3

The keys of an item in the index for querying

```
{
   Name,
   Description,
   Module,
   Attributes,
   Kind
}
```

Note. This is the lowest common denominator set for all sets of keys that comprise the mathematical knowledge covered in mathlib.

irrelevant to the way they are implemented. The name of any declarations and tactics is of immediate importance not only because it is unique and most distinguishable for every declaration in the library but also because it is one of the two leads one needs to have access to to be taken to the place of the documentation where the theorem, axiom, etc., or tactic is described in detail. The name of the module the declaration is in, together with the declaration's name, combined with the root of the website, comprise together the domain path to the exact location of the item in the documentation. Hence, whichever parts of the total data are included besides those two, the names of the declaration and module they are in must always be part of the indexed item for searching. At this time of development of the full-ranged searchable data structure, we have included only a single piece of information on top of those because it is most useful at this time and easiest to look into: the human-readable comment that is carried along with each declaration/tactic in the library, should there be any at all (Fig. 3). Not all declarations have the privilege of having a description, but most of them do, thanks to the communal effort of many caring theoretical and practical scientists who have decided to include one (van Doorn, Ebner, & Lewis, 2020).

In the light of us being at the beginning stage of mathlib's development, the two main distinctions in the data we are looking to traverse easily, tactics and declarations, do not cover nearly all the cases a theorem-prover has to make use of. Still, they are a fundamental part of the library that will only grow in importance (van Doorn et al., 2020). We begin by extracting the part of the syntax that would be necessary for searching, filtering and prioritization of results. The values described in Fig. 3 show what has been taken into account from each set of values in Fig. 1 and Fig. 2. This way, we are closer to building a solid foundation for further information retrieval handling like combining references between theorems in a knowledge graph, transcribing mathematical concepts and symbols into semantically understandable human-language texts of mathematical information, stemming, lemmatization and suggesting related queries.

Naturally, since we are on our way to create a single index for diverse types of structures, we can make distinctions between them already while creating it. Later we can use each structure's data to differentiate between them when querying. For example, the datasets in Fig. 1 and Fig. 2 are separate while in the pre-processing script. Some kind of a distinction can be included as part of the searchable data structure in order to later filter tactics out of the theorems, axioms, definitions, structures, and other mathematical concepts not part of the processing engine but part of the system. This filtering will be done at a later stage in development. As for now, we focused primarily on what is readily available in the data extracted directly from mathlib: in our case, the winners of the filtering mechanism were the attributes and kind of each declaration.

Filtering

The *kind* of a declaration, as described in Fig.1, holds one of the values of ['structure', 'theorem', 'definition', 'constant', 'axiom', 'inductive']. Moreover, the *attributes* of a declaration can be one of ['simp', 'nolint', 'ext', 'instance', 'class']. No kinds or attributes are described in the tactics' part of the system scope; thus, the *attributes* and *kind* keys of the resulting index will be empty for the tactics in the resulting index. This is not a problem because it will further distinguish between a tactic and a declaration when one wants to look up anything, should they be using any of the filters. The search functionality offers the following combination of filters at the moment of writing: each filter (*attributes*, *kind*) has its options, and upon selecting some of them, the user is presented with results that combine the selected options with an *OR.* Should one choose a combination of filters of both *attributes* and *kind,* one will receive results that match each of the filters together with an *AND* between their two sets of *OR*-combined selection of filters. For example, a user might select 'definition' and 'constant' of the attributes filter and 'nolint' and 'class' of the kind filter. Hence, they will see results that match their query as a string and have the following combination of attributes and kind: 'definition' or 'constant' and 'nolint' or 'class'.

Prioritization

Prioritization of results will be based on ordering the importance of each component in the index and the number of matches in each component. The highest priority is given to the name of the declaration, followed by its descriptive comment, and lastly, the name of the module it is in. This is because we have focused on exact matches of the query into the data. After all, many names of declarations hold stand-alone word combinations like "iff", "sa", "la", "lt", etc. Those mathematical concepts happen to be part of often-used words like "difficult", "same", "later", "halt", etc. Hence, it would be best if the search method distinguishes between searching into the name of the declaration, where it would be most beneficial to search for exact query matches, and its description, where it would be best to search for partial matches. This is set as part of the system's further development and will be used to prioritize direct and full matches over partial ones. The number of matches in the name are scored three times higher than their count in the module name. Moreover, the number of matches in the description is scored twice as high as those in the module names. In conclusion, the hierarchy of scores is sufficient at this time to distinguish between the best and worst results found. Further development can be done in many aspects, one of which is evaluating the number of instances found all over mathlib for each declaration.

Advantages of the Presented Index

As a result, we extracted a single unified index containing the essential information carried in the declarations and tactics. The searchable data structure is big enough to cover the highest-demanding structure but small enough to be used as an atom for further combinations in collections of any kind. Here we mean any collected mathematical knowledge, which hopefully will become more accessible and easier to traverse regardless of the language it is described in because its size only grows bigger and increases in complexity.

Tactics and theorems might be completely different and serve unrelated purposes. Still, they are part of the same corpus and are equally crucial for the development and correctness of mathlib as a trustworthy library of formal proofs. Hence, we have included them both in the same index hoping that this will bridge the gap between them and will allow users to traverse them both or separately, as they wish. Further improvements have to be made to distinguish between tactics and theorems to be part of the user-inputted filters, but this will be easy to add to the automated generating script as part of the ready-made index before it is created and sent to the client for querying.

An important part of our index integration is that it is created automatically during the generation of the documentation of mathlib, which by itself is also an automated process and is run often enough to be up-to-date with any new additions. We believe this first step towards having a greater and better overview of mathlib's contents is sufficient to be submitted as a stand-alone basis of future development because the fundamental parts of the system are included, and many thoughts on further development have emerged in the process.

Last but not least, we need to mention that there are some mathlib components to be added to the searchable index in the future in the next stage of enhancements. These include the number of instances of each theorem for prioritization of results and the descriptive texts found in the modules, which are not connected to a specific declaration but constitute an important part of the documentation and library itself. It will be easy to include them in the index because they will have a module name and a description, but they do not have an anchor for themselves on the webpage. This has to be included first to be able to reach the exact place of the descriptive comment from the set of results. It can be hidden, meaning it will not show itself as a "name" to the querying user, but once they select the result because of the matches in the module name or description itself, the name and the module name will be used in combination with the website root to take the user to the anchor we have created specifically for the chosen module comment.

Conclusion

Altogether, this paper describes the beginning stages of dividing, structuring, and re-combining a mathematical body of knowledge to search through it extensively. Many limitations have risen from the journey of this work, and most of them have been overcome, but a lot of improvements are to be included in future developments to help bypass the rest of them. Future work will include better-searching mechanisms that cover both strict and partial matching; filtering between different types of mathlib contents; user-inputted prioritization dependent on individual needs and preferences; color-coding and visual enhancements to ease the work process of mathematicians and novices in the field; semantic as well as syntactic search; conceptual suggestions on query typing; and much more. This communal effort proves what humanity is capable of once we work together. We are hopeful that the future holds many bridges between different programming languages used for the formalization of theoretical knowledge, which will serve each other and help inspire each other for better, more flexible, more abstract, and simultaneously more profound development of ideas, concepts, and correct, reliable proofs of those of them that stand to serve higher purposes in our lives.

Bibliography

Avigad, J., De Moura, L., & Kong, S. (2020). *Theorem Proving in Lean* (Release 3.4.0).

Hartnett, K. (2020). *Building the Mathematical Library of the Future*. Quanta Magazine. https://www.quantamagazine.org/building-the-mathematical-library-of-the-future-20201001/

Huch, F., & Krauss, A. (2020). *FindFacts : A Scalable Theorem Search*. In Isabelle Workshop 2020. https://files.sketis.net/Isabelle_Workshop_2020/Isabelle_2020_paper_3.pdf

Lean-Prover Community. (2021). *Index - Mathlib* . Mathlib Documentation. https://leanprover-community.github.io/mathlib_docs/

The mathlib Community: The Lean mathematical library. In: CPP. ACM, New York, NY, USA (2020). https://doi.org/10.1145/3372885.3373824

Stathopoulos, Y., Koutsoukou-Argyraki, A., & Paulson, L. (2019). *Developing a Concept-Oriented Search Engine for Isabelle Based on Natural Language : Technical Challenges*. In Artificial Intelligence and Theorem Proving (AITP) 2020. https://www.researchgate.net/publication/340390131_Developing_a_Concept-Oriented_Search_Engine_for_Isabelle_Based_on_Natural_Language_Technical_Challenges

Stathopoulos, Y., Koutsoukou-Argyraki, A., & Paulson, L. (2020). *SErAPIS : A Concept-Oriented Search Engine for the Isabelle Libraries Based on Natural Language*. In Isabelle Workshop 2020. https://files.sketis.net/Isabelle_Workshop_2020/Isabelle_2020_paper_4.pdf

van Doorn, F., Ebner, G., & Lewis, R. Y. (2020). *Maintaining a library of formal mathematics*. In Lecture Notes in Computer Science. https://doi.org/10.1007/978-3-030-53518-6_16