Ideas for a MathWiki Editor

Sebastian Reichelt
SebastianR@gmx.de

Abstract

We present some functional and non-functional requirements and wishes for a web-based editor for formalized mathematics, in particular for use in the MathWiki project at RU Nijmegen [13]. We discuss possible implementation alternatives, and argue for a holistic design of the entire wiki with editor features in mind.

1 Introduction

Since the invention of proof assistants, researchers have argued for a library of mathematics formalized in a machine-readable format; this goal is stated and explained in the QED manifesto [1], for example. Such a library would have a vast amount of use cases from the verification of complicated proofs (as in the Flyspeck project [4]), to computer algebra systems with strong correctness guarantees, and to learning environments for students to become familiar with mathematical proofs and check their results [17]. Most importantly, it could serve as a uniform repository for present and future mathematical theories, to ensure that no mathematical developments become “lost” in the ever-growing body of results.

Formal math differs from other variants of mathematics done on a computer in that definitions, statements, and proofs are built from a limited set of basic principles, so that the computer can be said to actually “understand” the contents of the library (to the extent possible). This enables to a degree of correctness and homogeneity that cannot be achieved any other way.

However, the amount of work necessary to build such a library is prohibitive for any single person or project [16]. One possible solution is to organize its development in a collaborative fashion, using wiki-like technology [13]. In contrast to regular wikis, only meaningful formal definitions and correct proofs can be entered. Still, a combination with informal content is possible and beneficial, potentially bringing together formerly separate user communities.

Compared to desktop-based proof assistant IDEs, the wiki approach trades performance and simplicity for ease of use and availability. The major benefits of a web-based solution are the lack of client-side installation and the ability to work directly on a single consistent library. Performance is not expected to be a large problem as long as the number of contributors is low. If the required server-side computation becomes too expensive in the future, a hybrid desktop/internet solution (i.e., client software accessing a remote library) may become a better alternative. At the moment, we favor a fully web-based solution because of its potential to attract more contributors in the first place.

A major caveat, however, is that formalization is not only time-consuming but also rather difficult. In particular, the learning curve is currently too steep to appeal to a significant number of novice users. Moreover, formal math is still closer to the source code of a computer program than to informal math [17]. For this reason, the features of a wiki editor strongly affect the potential user base: The more guidance and readily accessible information the editor provides, the easier and quicker the input of formalized mathematics will become, all other things being equal.

In addition, since new users first face the obstacle of having to learn about already existing formal content, it is even more important for the static (non-editing) part of the wiki to provide as many cues as the accompanying editor; one way to achieve this is to use the same rendering mechanisms in viewing and editing mode wherever possible. Since websites showing appropriately post-processed formal mathematics already exist (e.g. `isarmathlib.org`), the actual challenge lies in bringing the same features to an editor. Present systems such as the current MathWiki editor [13] or `wikiproofs.org` offer essentially a raw text editor, in contrast to feature-rich desktop IDEs like Proof General [2] or CoqIDE [12]. The ProofWeb system [6] contains an editor modeled after such IDEs, but it is specific to Coq at the moment and does not offer many of the features we envision. Especially, incorporating ProofWeb would imply that viewing and editing rely on entirely different code bases, so features would have to be implemented twice.

To develop an editor for the MathWiki project [13], we would especially like to leverage some of our experience gained from developing the HLM proof assistant [8, 9]: Its defining characteristic is a graphical user interface that is tightly integrated with the verifier component, so that many useful features can be implemented directly on top of the formal data structures. However, the intention of MathWiki is to provide access to a number of different existing proof assistants, such as Mizar [10] and Coq [12]. This is a compromise: Ideally, we would like to specify the formal content just once, using a rich user interface with HLM-like features and a general interchange format such as OMDoc [7], and use the resulting data to generate formalization for different proof assistants. However, since the conversion of formal mathematics between different proof assistants is problematic [18] (at the moment, at least), supporting different proof assistants in parallel seems to be the safer route to take.

This paper presents the result of a first investigation into the possibilities of implementing auxiliary features (for example those known from proof assistant IDEs or from HLM) in a MathWiki editor, on top of different provers.

2 Requirements

As indicated in the introduction, an editor for a mathematical wiki must be both easy to use and flexible with respect to the underlying prover technology. In this regard, different modes of interaction of existing proof assistants present a special challenge. Even just considering technological differences and ignoring mathematical foundations, there are actually several dimensions along which provers differ:

- The most well-understood dimension concerns the proof language, which can be either declarative or procedural [19]. Roughly speaking, declarative proofs consist mostly of lists of statements that are proved to hold, along with hints that guide the prover towards the verification of these statements. Procedural proofs contain commands (or “tactics”) that tell the prover exactly how to proceed (in contrast to the hints in declarative proofs, which merely need to contain enough information to prove the statement in question). Since these commands fully describe the proof, intermediate results are usually not included in the proof script, and can only be obtained by “replaying” the proof in the prover. (The Proviola tool [11] aims to address this shortcoming.) One requirement for a MathWiki editor is that procedural proofs should be just as easy to edit as declarative proofs. In other words, the editor needs to provide additional information to the user in order to make procedural proofs readable.

- A related but separate dimension is whether the prover maintains some internal state in addition to the input text. In most procedural provers, every line of input constitutes a state change. This state influences the information that is shown to the user and is necessary to understand procedural proofs. Coq, in particular, also has an “undo” feature to revert the last step [5]; this enables an editor like CoqIDE [12] to maintain a movable cursor indicating the portion of the input text that has been sent to the prover. If the proof language is declarative, no additional state is necessary; Mizar [10] is an example of a stateless prover. However, declarative languages have been developed for stateful proof assistants as well. Also, HLM [9] can be described as procedural but stateless: Its user input comes from context menus instead of text, and the contents of these menus contain the information that would normally depend on the prover state (but in this case merely depends on the the location or context of each menu, i.e. on the library contents).

- The graphical input method in HLM presents another challenge: It would seem that a text editor would be a basic ingredient of a generic MathWiki editor, but HLM proofs cannot feasibly be edited as text. On the other hand, an additional graphical interface would be helpful even if the primary input is in text form.

- Finally, a web-based front end is usually inherently asynchronous, while most existing software operates in a synchronous fashion (recent Isabelle developments being a notable exception [15]). There are two reasons for the asynchronous nature of websites: the potentially large latency of all operations that require client/server communication, and the standard mode of operation of text editor controls in browsers. In a declarative and stateless scenario, asynchronous verification is unproblematic: Whenever the input text changes, the prover can re-verify it and show the results to the user when they are available. Stateful provers are more difficult to connect to an asynchronous front end because the intended, actual, and observed state can diverge quickly. Finally, HLM constitutes a special case again: It is inherently synchronous because possible inputs are determined by menu contents that change after every operation which modifies the library.

Some compromises are necessary to accommodate all flavors of proof assistants, even in principle. In addition, the editor must integrate well with the rest of the wiki. A number of features (most of which are available in HLM, for example) would be desirable both in the wiki and its editor. We will briefly characterize their value according to the methodology of Cognitive Dimensions (CD) [3]. Of the four types of user activity mentioned in CD literature, incrementation, transcription, and exploratory design seem especially important at the current stage, whereas modification (of existing formal content) will most likely remain the job of a few experts for the foreseeable future. In addition, the ability to understand and browse existing data is vital, even though it is not classified as a user activity in the CD sense.

- The most obvious enhancement in a web-based viewer and editor is the use of (automatically generated) hyperlinks to navigate to referenced definitions and theorems. In terms of Cognitive Dimensions, such links improve the "visibility," or accessibility, of referenced objects, enabling exploration and modification. Since placing links in a text editor may be difficult to impossible, links may need to be shown separately in editing mode.

- When the user moves the mouse over a clickable link, an abbreviated version of the linked item can be displayed as a tooltip. This feature has the potential to greatly increase usability of a mathematical wiki because it reduces the number of pages that need to be opened in order to understand a given item. The general user-friendliness of tooltips stems from their non-disruptiveness: They typically disappear whenever they would stand in the way. However, since mathematical definitions are often complex, the size of the tooltip area can become a problem, especially since tooltips will typically appear at locations where they hide relevant content of the current page. Thus, an even less disruptive alternative would be a dedicated area on the page instead of a floating tooltip.

  In CD terms, temporarily showing the contents of a referenced item corresponds to the "juxtaposition" of that item with the one the user is currently viewing or editing. This helps the user understand the contents of the current item more quickly, and can also prevent errors due to incorrect definitions or theorem statements.

- Definitions, theorems, and proofs should be rendered in a visually pleasant form. One important ingredient is the use of common mathematical symbols; for example, the author of a definition could specify a custom symbol that represents the defined object at all places where it is used. Ideally, it should be possible to reproduce the usual mathematical notation even in cases where that involves more than a single symbol. In HLM, the notation for a definition in the library can be specified as a two-dimensional “layout” that includes placement of arguments and is augmented by further information such as rules for parentheses. The inclusion of arguments in the notation is possible because library entries cannot be referenced as mathematical entities; like “functors” and “predicates” in Mizar they are always referenced with specific arguments. In type-theoretical proof assistants, such definitions yield functions in the mathematical sense, which can be referenced on their own. In this case, the notation feature can only be reproduced approximately.

A user-defined notation for mathematical objects is a “secondary notation” according to CD, as it provides visual hints beyond the raw formal content, aiding in transcription and modification (assuming it is actually available in the editor, not just in the viewer). In this context, it especially reduces the “hard mental operation” of deciphering formal mathematics, by relating it to known informal math. This is desirable for all possible user activities.

- Although outside of the scope of this paper, we propose a tree or a tree-like menu of all definitions and theorems for navigation in the wiki, which should be available in all situations. As HLM shows, such a tree is much easier to browse if items are shown in their custom notation, and previews of items are shown as tooltips prior to opening them.
- The text editor should include syntax highlighting, at least for basic keywords (which is the prime example of a “secondary notation”).

3 Design Alternatives

Different approaches are possible depending on the importance of each requirement. In the current MathWiki implementation, the wiki and editor are entirely separate. Since the editor is a raw text editor at the moment, non-essential features like hyperlinks are available only for finished formalizations, after they have been submitted and verified. The simplest approach would be to enhance the text editor with as many features as possible, for example automatic asynchronous verification and highlighting of errors.

Although this approach is compelling especially because of its incremental nature, it bears two significant problems: First, most of the information that the user needs in addition to the raw text has to be displayed separately, and updated through a complex client/server protocol. Second, such an individual piece of software easily becomes more and more detached from the rest of the wiki as more features are added to it. For example, to accommodate stateful provers with procedural proof languages, there needs to be a display of the current state, but this display is then unavailable on the main page even though it would be equally important there.

In other words, such a design would be feasible but rather short-sighted: Over time, similar features would be desired on both wiki pages and editor pages, but most of the features described in section 2 would need to be implemented twice. In addition, the differences between provers could lead to separate editor implementations, requiring further duplication of features.

At the other end of the spectrum, there is the possibility of displaying and editing everything at a higher level, hiding the underlying textual representation. If the high-level representation is fully equivalent to its textual counterpart, it can be used equally for viewing and editing formal content, and all conceptual differences between proof assistants can be concealed by this abstraction layer.

This is very similar to the approach taken by the HLM proof assistant, although HLM goes one step further by omitting the textual representation entirely. The idea of HLM is that everything (including definitions, statements, and proofs) is displayed in a natural mathematical style, and input happens via menus which contain pre-rendered versions of the corresponding result. For example, if the goal of a proof is a universally quantified statement $\forall x \in S : P(x)$, the menu item corresponding to universal generalization will simply show "Let $x \in S$." In terms of Cognitive Dimensions, this reduces the "hard mental operation" of having to figure out the command for universal generalization (or even just realizing that universal generalization is the correct next step).

The existence of HLM shows that this method of proof input is viable but requires complex dialogs to input parameters of proof steps. One should also keep in mind that the HLM logic is designed especially to facilitate menu-based input; it is difficult to imagine how the same mechanism could be used as the primary or sole input method for an existing proof assistant.

However, a middle ground exists as well. The basic idea is to provide both a textual and a high-level representation side-by-side (or just the high-level representation if HLM is used as the underlying system). The additional high-level view provides all of the desired features such as hyperlinks, tooltips, custom notation, etc., but only limited editing facilities. It is shared between the main page and the editor page.

While this might seem like an obvious solution, the actual difficulty lies in the connection between these two views. With a declarative, stateless, and asynchronous system underneath, the high-level view can simply be updated at regular intervals. For example, if the Mizar system is used, an existing Mizar-to-XML translation [14] appears suitable as an intermediate representation from which a readable version of the document can be computed. With HLM, there is no text input, and all editing happens synchronously in the high-level view. However, procedural stateful provers present a challenge because the user expects to see the current state, and because a change at one position in the input text tends to break all commands beyond that position.

Our proposed solution is to limit the high-level view to the commands that have already been verified, and to merge the state display into it. In the case of proving $\forall x \in S : P(x)$ as above, at the beginning the high-level view simply contains this goal statement. After the user submits the appropriate command for universal generalization, a new line is added to the high-level view, showing "Let $x \in S$. Then $P(x)$:" to indicate the current hypothesis and goal.

In general, the contents of the high-level view are computed from all of the prover states after sending each command to the prover, up to the current state, rather than from the raw input text. Thus, no additional parsing of user input is necessary, and the display can be enhanced with all useful information that can be obtained from the prover.

If the user has to trigger every state change manually (for example using "up," "down," and "go to cursor" buttons as in existing proof assistant IDEs), the connection between both views becomes very loose, in contrast to the automatic updating in declarative mode. This problem becomes worse in a web front end because custom keyboard shortcuts are usually not available. However, because of the importance of the state display, the user presumably needs full control over the position that separates the verified and unverified parts.

Thus, we suggest that the input cursor be used to determine this position, which is equivalent to an automatic "go to cursor" operation whenever the cursor position changes. In particular, whenever the user finishes entering a command, that command is automatically sent to the prover. Besides requiring less keystrokes, a special advantage is that the verified part of the text does not need to be locked: If the user moves the cursor into this part in order to edit it, the prover will be instructed to backtrack to this position anyway. The lack of locking makes the editor "less synchronous," mitigating one of the differences between provers.

This feature can be regarded as an extended variant of the "electric terminator mode" available in Proof General [2]. The difference is that Proof General only reacts to the input of specific characters terminating the pieces of text that can be sent to the prover on their own; it does not change the prover state every time the user moves the cursor or presses backspace to remove a "terminator" character.

Although the elimination of all explicit navigation is a rather radical change from the user's point of view, first experiments with an implementation in CoqIDE look encouraging: The lack of interruptions from regular text input actually tends to make proof input somewhat smoother.

4 Conclusion and Future Work

We have presented requirements and design alternatives for an editor that is integrated into a mathematical wiki. The desire to support several proof assistants with different interaction styles, and to present formal content in a more high-level form than raw text, requires a compromise between a text editor and a structural view or editor. We have argued for a side-by-side presentation both in the editor and in the wiki itself, and described how a text editor can be connected to a high-level view, depending on the interaction mode of the underlying prover.

The next step will be to implement, within the MathWiki framework, the proposed method of interacting with provers. A particularly interesting question is how well the automatic "go to cursor" feature works together with asynchronous updating of the input text, and how much a delayed display of the current prover state (due to network latency) affects usability.

Many thanks go to Josef Urban for his support and very helpful discussions, and to the anonymous reviewers for their detailed comments (including a pointer to the concept of Cognitive Dimensions).

References

[1] The QED manifesto. In *Proceedings of the 12th International Conference on Automated Deduction*, CADE-12, pages 238–251. Springer-Verlag, 1994.

[2] David Aspinall and Thomas Kleymann. *Proof General user manual*. `http://proofgeneral.inf.ed.ac.uk/userman`.

[3] Thomas Green and Alan Blackwell. Cognitive dimensions of information artefacts: a tutorial. In *BCS HCI Conference*, 1998.

[4] Thomas C. Hales. Flyspeck: A blueprint of the formal proof of the Kepler conjecture. `http://flyspeck.googlecode.com/files/flypaper.pdf`.

[5] G. Huet, G. Kahn, and Ch. Paulin-Mohring. *The Coq Proof Assistant – A tutorial*, April 2004. `http://coq.inria.fr/getting-started`.

[6] Cezary Kaliszyk. Web interfaces for proof assistants. In S. Autexier and C. Benzmüller, editors, *Proc. of the Workshop on User Interfaces for Theorem Provers (UITP'06)*, volume 174[2] of *ENTCS*, pages 49–61, 2007.

[7] Michael Kohlhase. OMDoc *– An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, 2006.

[8] Sebastian Reichelt. The HLM proof assistant (website). `http://hlm.sourceforge.net/`.

[9] Sebastian Reichelt. Treating sets as types in a proof assistant for ordinary mathematics. (Unpublished note accompanying presentation at TYPES'10.) `http://hlm.sourceforge.net/types.pdf`, 2010.

[10] Piotr Rudnicki. An overview of the MIZAR project. In *Types for Proofs and Programs*, pages 311–332, 1992.

[11] Carst Tankink, Herman Geuvers, James McKinna, and Freek Wiedijk. Proviola: a tool for proof re-animation. In *Proceedings of the 10th ASIC and 9th MKM international conference, and 17th Calculemus conference on Intelligent computer mathematics*, AISC'10/MKM'10/Calculemus'10, pages 440–454. Springer-Verlag, 2010.

[12] The Coq Development Team. *The Coq Proof Assistant Reference Manual*. `http://coq.inria.fr/refman/`.

[13] Josef Urban, Jesse Alama, Piotr Rudnicki, and Herman Geuvers. A wiki for Mizar: motivation, considerations, and initial prototype. In *Proceedings of the 10th ASIC and 9th MKM international conference, and*

17th Calculemus conference on Intelligent computer mathematics, AISC'10/MKM'10/Calculemus'10, pages 455–469. Springer-Verlag, 2010.

[14] Josef Urban and Grzegorz Bancerek. Presenting and explaining Mizar. *Electron. Notes Theor. Comput. Sci.*, 174:63–74, May 2007.

[15] Makarius Wenzel. Asynchronous proof processing with Isabelle/Scala and Isabelle/jEdit. `http://www4.in.tum.de/~wenzelm/papers/async-isabelle-scala.pdf`, 2010.

[16] Freek Wiedijk. Estimating the cost of a standard library for a mathematical proof checker. `http://www.cs.ru.nl/~freek/notes/mathstdlib2.pdf`, 2002.

[17] Freek Wiedijk. The QED Manifesto revisited. *Studies in Logic, Grammar and Rhetoric*, 10(23):121–133, 2007.

[18] Freek Wiedijk. Encoding the HOL Light logic in Coq. `http://www.cs.ru.nl/~freek/notes/holl2coq.pdf`, 2010.

[19] Freek Wiedijk. A synthesis of the procedural and declarative proof styles of interactive theorem proving. `http://www.cs.ru.nl/~freek/miz3/miz3.pdf`, 2010.