# Extended Abstract: Dynamic Proof Pages

Carst Tankink James McKinna

**Abstract**

Reading a formal proof script written in a procedural (command-driven) style is difficult, bordering on impossible, without access to an interpreter for the script, that generates the states based on the commands. The Proviola system replaces the need for this interpreter, storing responses and displaying them on-demand.

This abstract describes a natural extension to the Proviola: instead of working on plain text files, the improved system takes an HTML rendering of the script, and decorates it with the proof states. The decorated page can be used as a Wiki page, generated out of a script file.

## 1 Introduction

Collaboration on formal (computer-verified) mathematics can be supported by taking a Wiki-based approach: a Wiki provides a repository of reusable results (*e.g.*, lemmas, theorems and definitions) and methodology, and it also lowers the threshold for contributing to the repository by novice users.

The software for such a Wiki can be designed in two directions:

- Bottom-up, or technology-centric, focussing on the technological problems one might encounter, such as consistency of the database, version control or file management.
- Top-down, or user-centric, first determining what activities to support for **authors** and **readers** of Wiki pages, by way of the interfaces (the pages in the Wiki) and developing the technology that underlies the interface.

Both approaches can be interleaved, as technological advances can inspire new interaction styles, while a fleshed-out interface drives the direction of technological development.

In this abstract, we will mainly work user-centric, describing a revised and redesigned approach to rendering the pages a reader might see in a Wiki for formal mathematics, henceforth simply a **MathWiki**. While we describe a design, the interface describes the behaviour desired of (formal) Wiki pages and can be considered top-down. Our work at present leaves open how an author can write such pages, both in terms of proof script content as well as Wiki commentary, but our approach emphasises upward compatibility, preserving and then enhancing existing author workflows.

### 1.1 Proviola

Because the MathWiki contains the artifacts of formal, computer-evaluated mathematics, it is possible to use the tools which manipulate these artifacts to enhance the Wiki pages. As one example of this, we developed the **Proviola** [6, 5].

The Proviola is a tool for displaying the dynamics of (script-based) formal proof development. A proof script is the result of an interactive session between an author and a proof assistant (PA). Such a session consists of the author writing commands to which the assistant responds with a proof state. The author proceeds by writing a new command, which modifies the proof state, and this game continues until the theorem is proven. The transcript of the session are the commands written by the author, which can be replayed by a reader at a later moment.

Because proof state is important in understanding such scripts, a reader typically cannot read the script without first loading it into the proof assistant and inspecting it step by step: the script represents

one side of the interaction (the author's), but to understand the proof coded in it, the reader needs access to the full interaction, including the PA responses.

When such scripts are published on the web, including as pages created for a Wiki, this may become a nuisance: the reader should have both a PA and a web browser open, using the browser to explore the links between notions, and using the PA to inspect the proof states and better understand the proof. To save the reader from having to flip between programs just to explore a repository, we create a new page for the Wiki. This page contains both the proof script and the state, as generated by the PA.

**Declarative proofs** An alternative to the Proviola would be to write proofs in a 'declarative' style, such as is done in the Mizar system [1] or when writing Isar scripts for the Isabelle PA [9]. Proofs in this style look more like traditional mathematics: the author provides the proof states and the rules from which these states follow, and the system checks that the use of these rules was allowed.
This style can allow for readable proofs, but many proofs are already written in a procedural style, and it is our aim to make those proofs more accessible, without requiring authors to rewrite their scripts.

The tool that creates these dynamic pages is a **camera**: it parses a proof script, sends the commands to a PA and records the responses. The resulting list of (command, response) pairs is stored in an XML file, which is transformed into an HTML page with minimal JavaScript and CSS. This HTML page allows the reader to have direct access to the state, by just pointing at the commands he is interested in.

The first prototype for the Proviola worked with scripts for the Coq proof assistant, sending the commands through the ProofWeb [3] system. The resulting pages were undecorated, containing only the commands and responses as plain text.

Towards improving the display of the proof movies, we revised the tool to send each command instead to Coqdoc. This tool, part of the Coq [7] distribution, marks up commands and annotated comments in HTML, allowing syntax highlighting and hyperlinks to referred lemmas and definitions. In essence, for this version of our camera, we intended as output an HTML page as Coqdoc would render it, but enhanced with the proof states as in our original movies.

This approach works, but has two drawbacks:

- It does not 'play nice' with other tools or workflows: the camera is a wrapper around Coqdoc, replacing that tool, so it cannot integrate easily into existing workflows that use Coqdoc produce HTML documentation, such as the "Software Foundations" course notes of Benjamin Pierce and collaborators [4]. Remedying this drawback is necessary to allow others to make good use of our tool.
- Because of the way Coqdoc works, namely on a per-file basis, instead of on a per-command basis, it is difficult to generate hyperlinks, which is essential to support readers in browsing a Wiki containing the enriched pages: the information Coqdoc uses to generate hyperlinks is stored in a separate file, that uses offsets in the original script for identifying location. This workflow presents each command as a stand-alone file to Coqdoc, uncoupling the presented data from the linking information.

We have recently re-evaluated our workflow for creating movies, to tackle both problems at the same time, and redesigned the tools accordingly.

## 2 Proviola: New Workflow

Instead of using the camera as a tool that calls Coqdoc, the camera was redesigned to be able to instead read the files generated by Coqdoc. Its implementation is straightforward: extending the parser for Coq

scripts to extract the commands from an HTML tree, by erasing the markup information. The resulting commands are sent to the prover, and the marked-up tree is stored together with the response.

Next to implementing a new parser, we also added communication with a locally installed Coq interpreter: instead of having to use ProofWeb, the camera can also use a local prover, allowing the use of special directives and libraries. We envisage that potential users and authors may wish to bind in custom PAs for document checking, beyond the vanilla distribution supported by for example ProofWeb.

This new workflow allows a camerawoman to create movies out of existing Coqdoc-generated HTML files, solving the first drawback. The second drawback is also solved: Coqdoc puts links into the HTML file, and these links are copied into the movie, possibly modified to refer to other movies.

There is a question of correctness arising from our redesigned process: how does a reader know that the movie is faithful to the original script? In the plain-text setting, this could be easily verified: the 'command' section of the movie should match up with the script. In the case of the new Coqdoc-reader, this requirement becomes a bit more complex: the reader should make sure that the transformation from script to HTML preserves the code, and that the transformation from the HTML tree back to the commands also remains faithful. To assist in this, the movie also contains the extracted commands, which might make it easier for the reader to verify this.

## 3 Movies as Wiki Pages

The Proviola camera can be used to generate pages in the MathWiki: it is a tool transforming a source file (the proof script) into an HTML page. This means we can generate a Wiki out of a repository of script files, where the proof states can be inspected on the pages themselves.

We have designed a prototypical Wiki system that creates HTML movies when a page is requested. It can be found at `http://mws.cs.ru.nl/mws/`. Because generating a movie can take some time, it runs as a background process, caching the generated movie for future use. Until that completes, the Coqdoc-generated HTML is shown. The movie-based Wiki page contains links to other Wiki pages (or pages in the Coq standard library). This prototype does not currently address any of the more technical issues, such as maintaining consistency of the repository when a page is changed, but serves as the basis for future research and experimentation.

The prototype has a basic editor: the entire proof script can be changed through a text box, where storing the script triggers a verification of the content, and clearing the cached copy of the movie. The content is only stored if it is deemed correct by the proof assistant.

This prototype does not provide any sophisticated support for (collaboration of) page authors, but focuses on the readers. These readers might become authors themselves at a certain point, but because formal development has a steep learning curve, they need all the help they can get in understanding the material that is already there.

## 4 Future Work

The Proviola is a reader-centric tool: it does not provide any new possibilities for writing the pages in the Wiki. To extend the Wiki, we intend to tackle the following issues:

- How to write a narrative of a movie, without having to change the underlying script; this use case arises when working with non-editable third-party sources. The sources might be non-editable because they are critical for other developments (similar to Wikipedia's "protected pages"), but they might be included in an explanation of a formalization. Such narration might be achieved through some form Wiki-linking (specifying what objects to include in a marked-up document),

but to make this work for arbitrary regions of code raises some questions about how to specify the "comb": what we previously [5] called the data structure(s) necessary to provide a commentary overlay which avoids having to modify the underlying source.

We do not intend this narration to be limited to describing a single proof script in a single file. Instead, a narrative can span an entire repository, or describe the different approaches taken in different proof assistants;

- How to relate such technology and workflows to those supported by other literate tools such as lhs2TEX[2];
- How to extend this work to other provers: our prototype was easily adapted to the sequential interaction model of Isabelle, but this question still remains open for document-oriented PAs such as Mizar, Epigram or Agda, where the underlying interaction model is 'batch-mode' or even could be considered in terms of a 'rectangle-based' granularity of editing; in any case, such models challenge the simplicity of linear, script-based approaches;
- Integrate the prototype Wiki with the advanced methods for maintaining consistency in repositories. In particular, we will integrate the prototype with the version control-based workflows implemented for the Wiki for Mizar [8].

Our objective is to integrate such technology into a fully-fledged authoring framework, in which script authoring and checking, enhanced 'Coqdoc'-style documentation, and third-party 'narration' coexist within a MathWiki.

## References

[1] Mizar home page, 2011. Available through: `http://mizar.org`.

[2] Ralf Hinze and Andres Löh. Guide to lhs2TEX, version 1.17. Obtained from `http://people.cs.uu.nl/andres/lhs2tex/Guide2-1.17.pdf` on May 30, 2011.

[3] Cezary Kaliszyk. *Correctness and Availability. Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web.* PhD thesis, Radboud University Nijmegen, 2009.

[4] Benjamin C. Pierce, Chris Casinghino, and Michael Greenberg. Software foundations. Course notes, online at `http://www.cis.upenn.edu/~bcpierce/sf/`, 2010.

[5] Carst Tankink, Herman Geuvers, and James McKinna. Narrating formal proof (work in progress). In David Aspinall and Claudio Sacerdoti Coen, editors, *User Interfaces for Theorem Provers 2010. To appear in an issue of ENTCS*, 2010.

[6] Carst Tankink, Herman Geuvers, James McKinna, and Freek Wiedijk. Proviola: a tool for proof re-animation. In Serge Autexier, Jacques Calmet, David Delahaye, Patrick D.F. Ion, Laurence Rideau, Renaud Rioboo, and Alan P. Sexton, editors, *Intelligent Computer Mathematics*, volume LNAI 6167 of *Lecture Notes in Artifical Intelligence*, pages 440 – 454. Spring-Verlag Berlin Heidelberg, March 2010. Preprint available through `http://cs.ru.nl/~carst/files/proviola.pdf`.

[7] The Coq Development Team. The Coq proof assistant. Web page, obtained from `http://coq.inria.fr` on October 5, 2009.

[8] Josef Urban, Jesse Alama, Piotr Rudnicki, and Herman Geuvers. A Wiki for Mizar: Motivation, considerations, and initial prototype. In Serge Autexier, Jacques Calmet, David Delahaye, Patrick D. F. Ion, Laurence Rideau, Renaud Rioboo, and Alan P. Sexton, editors, *AISC/MKM/Calculemus*, volume 6167 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 2010.

[9] Makarius Wenzel. *Isabelle/Isar — a versatile environment for human-readable formal proof documents*. PhD thesis, Technische Universität München, Fakultät für Informatik, 2002.