jsCoq: Towards Hybrid Theorem Proving Interfaces

Emilio Jesús Gallego Arias
MINES ParisTech
PSL Research University, France
e@x80.org

Benoît Pin
MINES ParisTech
PSL Research University, France
benoit.pin@mines-paristech.fr

Pierre Jouvelot
MINES ParisTech
PSL Research University, France
pierre.jouvelot@mines-paristech.fr

We describe jsCoq, a port of the Coq interactive theorem prover to ECMAScript® 2015. jsCoq typically runs inside a browser, allowing the user to interact with arbitrary Coq proof scripts in the usual way, along a few new possibilities.

jsCoq is targeted at educational use and is self-contained, easing distribution and installation. The only system requirement is a standard-compliant browser.

The current release ships more than 10 popular Coq libraries, and supports popular books such as *Software Foundations* or *Certified Programming with Dependent Types*.

1 Introduction

Interactive Theorem Proving (ITP) relies on mutual human-machine feedback to build proofs by refinement. Typically, the user first requests the proof assistant to validate or guess some proof step. Depending on the output of the tool, she will continue the proof or correct the just given step.

Teaching ITP is usually practice-led; students are encouraged to interact with the tools from the start, simultaneously discovering their way into the particularities of the implementations and the logical theories behind them. Popular ITP teaching material, such as [20, 10], is written in a *literate programming style*, pioneered by Donald Knuth. In literate programming, source code and comments meld into one to form a coherent, self-documenting book and program. An additional constraint in the context of proof assistants is that the user *must be able to interact with the book*. The same principles apply to documentation, which may be hard to understand without the ability to run and play with examples.

Narrowing down our attention to the Coq [25] proof assistant, most teaching material is consumed by students with the help of specific Integrated Development Environments (IDE). Popular options are CoqIde or ProofGeneral [5, 21], but more alternatives exist. IDEs are typically derived from text editing programs to provide specialized support for proof development.

This approach works well in practice, however, IDEs are mostly focused on proof development and provide varying degrees of support for document-like features. For instance, not all IDEs support images, advanced formatting, hyperlinking, or dynamic content creation. This hinders readability and it is not uncommon to see students with the same document doubly opened in a document reader and the IDE, as it is difficult to make sense of the document without the possibility of interaction.

Additionally, the specialized nature of these tools makes installation sometimes heavy, especially if third-party addons are involved, posing a barrier to the casual learner.

jsCoq aims to tackle both of these problems from the document point of view: instead of equipping an IDE with document-like features, we extend documents to provide IDE-like capabilities. Our

Submitted to:
UITP 2016

approach is to profit from modern browsers and web standards, embedding Coq as an application inside the browser. In this setting, Coq scripts are plain HTML documents, and a full Coq instance is run locally in the browser for every document. A document manager — written in JavaScript — manages the communication between the browser and the Coq instance.

This setup provides a working Coq environment in a transparent way for the user. She will just click a link, and have an interactive Coq document ready without any other special action or installation, even if the script depends on exotic libraries or addons.

jsCoq main design constraints were to:

- provide a complete port of Coq to the browser platform;
- support existing teaching material;
- support large, real-world developments in an usable way;
- use modern web technologies for the front end;
- run over an unpatched Coq version;
- provide an up-to-date Coq experience.

So far, we believe we have met all these goals.

Structure of the Paper We start with a brief introduction to the "Web-Based Hybrid Document Model" in Sec. 2. Sec. 3 gives an overview of jsCoq's main components; we briefly compare with related work in Sec. 4 and conclude.

2 The Web-Based Hybrid Document Model

We use the term *hybrid document* [1] to denote documents capable of containing objects whose interpretation is given by an external program. A typical example is a word processor embedding a spreadsheet or, in our case, a HTML page containing a Coq proof script.

The hybrid-document platform of choice for jsCoq is the so-called *modern web platform* of ECMAScript® 2015 [11] standard. The platform is composed of the HTML markup language, used to define base documents, and JavaScript — a Turing-complete, object-oriented and functional language — used to manipulate HTML documents by means of the *Document Object Model* API.

The web platform enjoys wide adoption and support, and increasing compatibility and standardization is making it more future-proof. The ubiquity of JavaScript code in the web has led to huge improvements in browsers, which can now run applications of a size and complexity thought infeasible some years ago. The process known as *transpiling*[2] allows developers to port native applications without a full rewrite. The number of libraries available is staggering; a document can easily gain advanced 3D mathematical plotting [3], editing capabilities [16], or even LaTeXsupport [4], with a few lines of code.

There are many interesting examples of web-based hybrid documents. Jupyter[3] allows to create and share documents that contain live code, equations, visualizations and explanatory text; Eloquent JavaScript [17] is an interactive book where all the examples can be run and edited inline.

Web-Based Coq Documents For jsCoq documents, we have chosen a simple solution: the user provides an arbitrary HTML document, with editable Coq code wrapped in `textarea` elements. At the end

[1] or *rich document*, *interactive document*, or any of the many names used in the literature for this concept

[2] the process of compiling a language to another, in this case, to JavaScript

[3] previously known as IPhyton

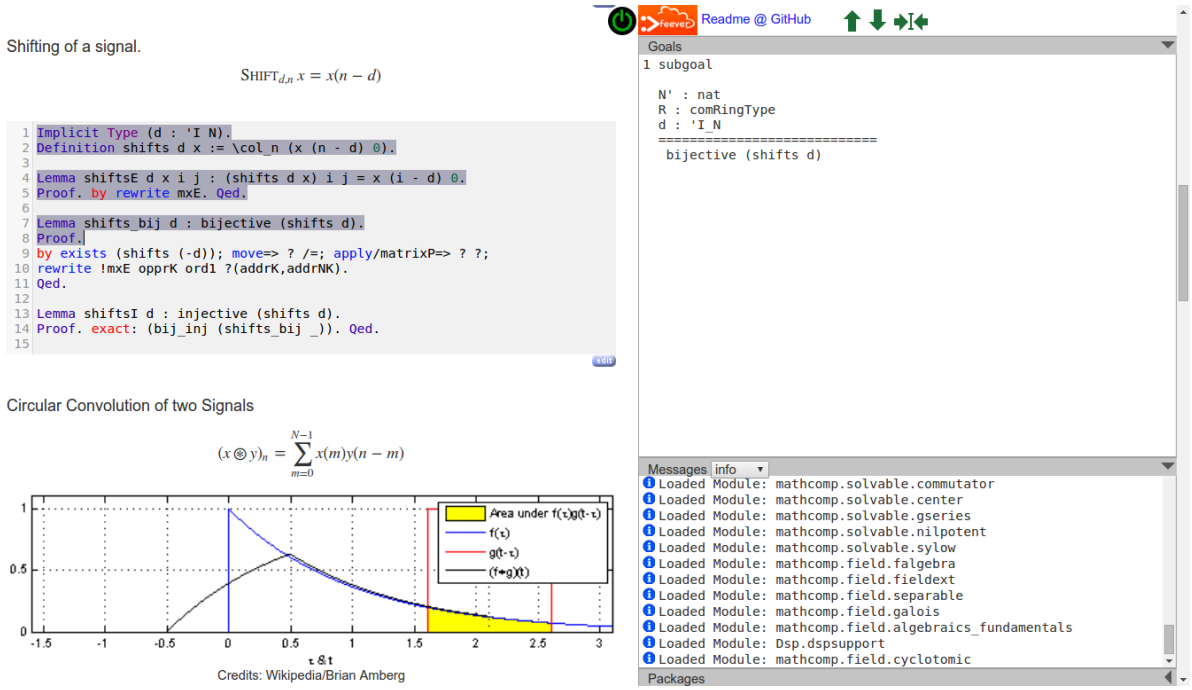

Figure 1: jsCoq 0.6 running a DFT formalization

of the document, the user loads jsCoq and provides the list of the `textarea` ids that will conform the Coq document:

```
<script src="js/jscoq-loader.js" type="text/javascript"></script>
<script type="text/javascript">
  loadJsCoq('./').then( () => new CoqManager (list_of_ids, [options]) );
</script>
```

After loading, a panel containing the goals and log is attached to the document and the rest of the content is ignored, as shown in Fig. 1. The jsCoq landing page[4] provides an example with a simple proof. Additional features are being developed (see Sec. 5).

The best way to get a feeling of the system is to try it out. Readers are advised to have a look at the following use cases:

- DFT, a small development of the theory of the Fourier Transform, following [23];
- Mtac, the Mtac [30] tutorial;
- STLC, the "Simply Typed Lambda Calculus" chapter from [20];
- StackMachine, the first chapter of [10].

3 Overview of the System

jsCoq is mainly implemented in Ocaml and ECMAScript 2015. The project is open-source and code/builds can be accessed at [1].

The system consists of three main parts.

[4]The current version works best with a recent Google Chrome web browser.

Ocaml wrapper from Coq to JavaScript. Current Coq IDE protocols rely on Unix features that are not available on a browser's environment, so we have chosen to use Coq's Ocaml API. This saves a considerable amount of interfacing and provides more flexibility than other options.

The small Ocaml wrapper sees Coq as a library and takes care of creating the JavaScript object for use from the browser. The whole Ocaml bytecode object is then compiled by js_of_ocaml [27] to a monolithic JavaScript file exposing a `jscoq` object. This object can be explored in the standard JavaScript debugging console.

Coq package manager. This is necessary as Coq library search requires a Unix filesystem is not available in the browser. so jsCoq needs to know the available libraries and their location. The manager is currently written in Ocaml but parts of it could be rewritten in JavaScript.

User interface. Written in ECMAScript, this part scans the document, builds an internal view, and takes care of parsing, document navigation, and communication with the Coq JavaScript object.

We proceed to describe the parts in more detail.

3.1 Coq Protocol

The `jscoq` provides the following methods and events to the JavaScript side:

```
1  method init   : initInfo t -> Stateid.t
2  method add    : Stateid.t -> int -> js_string t -> Stateid.t
3  method edit   : Stateid.t -> unit
4  method commit : Stateid.t -> unit
5  method query  : Stateid.t -> js_string t -> unit
6
7  method version: js_string t
8  method goals  : js_string t
9
10 method add_pkg : js_string t -> unit
11
12 method onLog   : js_string t
13 method onError : Stateid.t
14 method onInit  : unit
15 ...
```

The API closely follows the one of the STM [6], with slight differences in error handling.

There are plans to make Coq itself serialize its STM protocol, removing the need for the wrapper almost entirely (see Sec. 5 for more details).

3.2 The Document Manager

The document manager relates the document to the proof engine and has three distinct components:

- a `CoqPanel` object, providing the user interface for the goal and query buffers;
- a `CoqProvider` abstract object, that encapsulates the management of Coq statements. In particular, it takes care of selecting the next statement, highlighting, change notifications...;
- a `CoqManager` object that queries the providers and coordinate them with the panel and Coq itself, propagating errors and logs, and keeping track of the proper Coq state.

The CodeMirror Provider A key feature of our approach is the use of CodeMirror as an instance for the `CoqProvider` component. Indeed, the Coq CodeMirror mode is capable of parsing and recognizing Coq statements, and will notify the manager when a particular part is invalidated by the user.

3.3 Package Manager

The package manager takes care of loading the needed packages and Coq .vo files into the Coq instance. Packages are described as JSON files, stating their dependencies and the set of Coq logical paths the package provides. jsCoq doesn't allow unqualified modules.

Each logical path consists of the module id, .vo files and .cma files, as for example:

```
{
 "pkg_id": [ "Coq", "extraction" ],
 "vo_files": [
   "ExtrHaskellNatInt.vo", "ExtrOcamlString.vo", "ExtrHaskellBasic.vo",
   "ExtrOcamlIntConv.vo", "ExtrHaskellZNum.vo", "ExtrOcamlNatBigInt.vo",
   "ExtrOcamlBigIntConv.vo", "ExtrHaskellZInteger.vo",
   "ExtrOcamlNatInt.vo", "ExtrHaskellNatNum.vo", "ExtrHaskellString.vo",
   "ExtrOcamlZInt.vo", "ExtrHaskellZInt.vo", "ExtrOcamlBasic.vo",
   "ExtrOcamlZBigInt.vo", "ExtrHaskellNatInteger.vo"
  ],
 "cma_files": [ "extraction_plugin.cma" ]
}
```

The package manager will load the needed files in the background and notify the IDE when they are ready. We currently support more than 10 popular Coq packages, including the full Mathematical Components library [13].

3.4 Serialization

As a test-bed, jsCoq provides serialization of the internal `constr` Coq type to both sexps and JSON. This is done automatically using Ocaml's ppx technology.

The idea is that, in the future jsCoq can work with a finer representation of Coq terms and scripts than the one currently possible. This would allow for instance to recognize special operators and print them specifically in a reliable way.

3.5 Document Generation

Writing manual HTML documents can be tedious in some cases. We provide a fork of the CoqDoc tool that can directly generate jsCoq documents. For the time being, the tool is aimed at achieving maximal compatibility with existing developments, but plans for extensions exist.

4 Related Work

We do not attempt here to survey the huge amount of related work in this domain, but we list some additional related tools known to the authors.

PeaCoq [22] is a web-based front end for Coq, and indeed their editor support was the base for jsCoq's one. PeaCoq provides much richer proof-building capabilities than jsCoq; however it relies on a central server. The proof-by-pointing Coq environment [8, 9] allows the user to develop proofs by

interacting with logical connectives. The Logitext [29] approach is related, but web-based. Several design considerations of jsCoq have been influenced by these efforts.

ProofWeb [18] provides a web interface to a Coq server and many other theorem provers. Some other IDEs efforts for Coq are [12, 28, 7].

5 Conclusions and Future Work

jsCoq is still a work-in-progress. We believe the possibilities are exciting and so far the result has behaved in a stable way, filling a niche for casual Coq users, or people willing to try Coq add-ons in an easy way, and even providing support for a Coq course[5].

However, much work remains to be done until the product can be considered "final". In particular:

- the HTML user interface and panels need a complete rework to allow for more flexibility – in particular, we'd like to provide diffs and history for the goal buffer;
- editor support must be improved, since the current CodeMirror mode has some parsing inefficiencies and may behave strangely in the presence of delimiting braces;
- content-generating features need to be merged into the main distribution – a "Coq box" is a special HTML div that is filled with the result of some Coq query, and is a first step towards providing Proviola-like [24] functionality in jsCoq;
- the **udoc** document generator needs more work before production-ready books can be produced;
- our ad-hoc serialization of Coq internals needs to be turned systematic, given that this will be an effort that other IDEs should benefit from.

We hope some of these improvements would be available around the timeline of the workshop.

Acknowledgments

This research has been partially funded by the ANR FEEVER project.

References

[1] *jsCoq development page*. `https://github.com/ejgallego/jscoq`. Accessed: 2016-05-13.

[2] *The Jupyter Notebook is a web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text.* `https://jupyter.org/`.

[3] *mathbox: Presentation-quality WebGL math graphing*. `https://gitgud.io/unconed/mathbox`.

[4] *MathJax: A JavaScript display engine for mathematics that works in all browsers.* `https://www.mathjax.org/`.

[5] David Aspinall (2000): *Proof General: A Generic Tool for Proof Development*. In Graf & Schwartzbach [14], pp. 38–42. Available at `http://dx.doi.org/10.1007/3-540-46419-0_3`.

[6] Bruno Barras, Carst Tankink & Enrico Tassi (2015): *Asynchronous Processing of Coq Documents: From the Kernel up to the User Interface*. In Urban & Zhang [26], pp. 51–66, doi:10.1007/978-3-319-22102-1. Available at `http://dx.doi.org/10.1007/978-3-319-22102-1_4`.

[5] https://team.inria.fr/marelle/en/advanced-coq-winter-school-2016/

[7] Jesper Bengtson & Hannes Mehnert (2013): *Kopitiam–a unified IDE for developing formally verified Java programs*. Technical Report.

[8] Yves Bertot (1999): *The CtCoq System: Design and Architecture*. *Formal Asp. Comput.* 11(3), pp. 225–243. Available at http://dx.doi.org/10.1007/s001650050049.

[9] Yves Bertot, Gilles Kahn & Laurent Théry (1994): *Proof by Pointing*. In Hagiya & Mitchell [15], pp. 141–160. Available at http://dx.doi.org/10.1007/3-540-57887-0_94.

[10] Adam Chlipala (2011): *Certified Programming with Dependent Types*. MIT Press. Available at http://adam.chlipala.net/cpdt/. http://adam.chlipala.net/cpdt/.

[11] Ecma International (2015): *ECMAScript 2015 Language Specification*, 6th edition. Geneva. Available at http://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf.

[12] Alexander Faithfull, Jesper Bengtson, Enrico Tassi & Carst Tankink (2016): *Coqoon An IDE for interactive proof development in Coq*. In: *TACAS*, Eindhoven, Netherlands. Available at https://hal.inria.fr/hal-01242295.

[13] Georges Gonthier, Assia Mahboubi & Enrico Tassi (2008): *A Small Scale Reflection Extension for the Coq system*. Research Report RR-6455. Available at https://hal.inria.fr/inria-00258384.

[14] Susanne Graf & Michael I. Schwartzbach, editors (2000): *Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings*. *Lecture Notes in Computer Science* 1785, Springer.

[15] Masami Hagiya & John C. Mitchell, editors (1994): *Theoretical Aspects of Computer Software, International Conference TACS '94, Sendai, Japan, April 19-22, 1994, Proceedings*. *Lecture Notes in Computer Science* 789, Springer.

[16] Marijn Haverbeke: *CodeMirror is a versatile text editor implemented in JavaScript for the browser.* https://codemirror.net/.

[17] Marijn Haverbeke (2011): *Eloquent Javascript*, 1st edition edition. No Starch Press.

[18] Cezary Kaliszyk (2007): *Web Interfaces for Proof Assistants*. *Electronic Notes in Theoretical Computer Science* 174(2), pp. 49 – 61. Available at http://www.sciencedirect.com/science/article/pii/S1571066107001697. Proceedings of the 7th Workshop on User Interfaces for Theorem Provers (UITP 2006).

[19] Greg Morrisett & Tarmo Uustalu, editors (2013): *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*. ACM. Available at http://dl.acm.org/citation.cfm?id=2500365.

[20] Benjamin C. Pierce, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjoberg & Brent Yorgey (2015): *Software Foundations*. Electronic textbook. http://www.cis.upenn.edu/~bcpierce/sf.

[21] Clément Pit-Claudel & Pierre Courtieu (2016): *Company-Coq: Taking Proof General one step closer to a real IDE*. In: *CoqPL'16: The Second International Workshop on Coq for PL*, doi:10.5281/zenodo.44331. Available at http://hdl.handle.net/1721.1/101149.

[22] Valentin Robert: *PeaCoq is a web-based front-end to the Coq proof assistant.* https://github.com/Ptival/PeaCoq.

[23] Julius Orion Smith III (2007): *Mathematics of the Discrete Fourier Transform (DFT): with Audio Applications*, 2nd edition. W3K Publishing. Available at https://ccrma.stanford.edu/~jos/mdft/.

[24] Carst Tankink, Herman Geuvers, James McKinna & Freek Wiedijk (2010): *Intelligent Computer Mathematics: 10th International Conference, AISC 2010, 17th Symposium, Calculemus 2010, and 9th International Conference, MKM 2010, Paris, France, July 5-10, 2010. Proceedings*, chapter Proviola: A Tool for Proof Re-animation, pp. 440–454. Springer Berlin Heidelberg, Berlin, Heidelberg. Available at http://dx.doi.org/10.1007/978-3-642-14128-7_37.

[25] The Coq development team (2016): *The Coq proof assistant reference manual*. LogiCal Project. Available at `http://coq.inria.fr`. Version 8.5pl1.

[26] Christian Urban & Xingyuan Zhang, editors (2015): *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*. *Lecture Notes in Computer Science* 9236, Springer, doi:10.1007/978-3-319-22102-1. Available at `http://dx.doi.org/10.1007/978-3-319-22102-1`.

[27] Jérôme Vouillon & Vincent Balat (2014): *From bytecode to JavaScript: the Js_of_ocaml compiler*. *Softw., Pract. Exper.* 44(8), pp. 951–972, doi:10.1002/spe.2187. Available at `http://dx.doi.org/10.1002/spe.2187`.

[28] Makarius Wenzel (2013): *PIDE as front-end technology for Coq*. *CoRR* abs/1304.6626. Available at `http://arxiv.org/abs/1304.6626`.

[29] Edward Z. Yang: *Logitext is an educational proof assistant for first-order classical logic using the sequent calculus, in the same tradition as Jape, Pandora, Panda and Yoda.* `http://logitext.mit.edu/main`.

[30] Beta Ziliani, Derek Dreyer, Neelakantan R. Krishnaswami, Aleksandar Nanevski & Viktor Vafeiadis (2013): *Mtac: a monad for typed tactic programming in Coq*. In Morrisett & Uustalu [19], pp. 87–100, doi:10.1145/2500365.2500579. Available at `http://doi.acm.org/10.1145/2500365.2500579`.