OPAM for Coq

Guillaume Claret

OPAM for Coq

Guillaume Claret

2015

Contents

1 Abstract

More and more people are using the Coq system to develop theories or verify theorems and softwares. Unfortunately, there are no central places to share Coq developments which offer a unified and mechanized way to install new libraries or to express dependencies between projects. The existence of such a central place would foster the Coq community, by enabling more re-usability, more visibility and more competition among the Coq projects.

We will present our work on setting up a package manager infrastructure for Coq. We reuse the package manager OPAM from the OCaml community and design a specific repository architecture, an automated bench system and a website to present the list of the Coq packages. Today, the OPAM infrastructure is usable and used by more and more people to share their developments. To the best of our knowledge, this infrastructure is the first package management system able to organize mathematical proofs in the large.

2 Introduction

In the programming language communities, the package managers have long been considered as one of the fundamental tools to create an ecosystem of re-usable libraries. Noticeable examples are the npm[1] package manager for JavaScript, the Gems[2] repository for Ruby or the CRAN[3] repository for the language for statistics R. It happens that the challenges faced by the users of Coq to install external libraries, share their developments or express complex dependency chains between projects are very similar to those encountered with the programming languages. However and surprisingly, to the best of our knowledge, no modern package management systems have been proposed for a theorem prover.

When I started working on the idea of a package manager for Coq, we realized that we were many trying to reach this goal[4]. As the project started to grow, more people of the community got involved in order to add new ideas and reach a general consensus[5]. Thus, this work on a package manager for Coq is mostly a team work.

Instead of redeveloping a new package manager, we decided to use the existing OPAM package manager from the OCaml community. The Coq system or plugins being implemented in OCaml and most of the OCaml users being used to OPAM, the OPAM package manager appeared as the simplest choice for us, as much technically as socially.

In this chapter we will present what I mostly contributed to:

- an architecture of repositories[6] which combines a repository for the stable versions, a repository for the development versions and a system of distribution (page 4). Most of the work was actually devoted to the maintenance of these repositories (accounting for more than 400 commits);

- a port of the existing Contribs repository[7], containing more than 150 projects, to the OPAM system (page 6);

[1] www.npmjs.com
[2] rubygems.org
[3] cran.r-project.org
[4] Mainly Thomas Braibant and Cyril Cohen.
[5] In particular Enrico Tassi and Guillaume Melquiond.
[6] Available on github.com/coq/opam-coq-archive.
[7] The Contribs project is hosted on GForge on coq-contribs.gforge.inria.fr.

- a bench system (page 6) to continuously monitor the status of the packages by verifying that they can all by installed without errors and that they respect the policy of the Coq repositories;
- a website (page 8) listing the descriptions of the stable packages available for Coq[8]. We generate this website using the program OpamWebsite[9], entirely written in Coq thanks to our library Coq.io[10] to implement the concurrent inputs–outputs operations.

3 OPAM

The package manager OPAM[11] was introduced by the company OCamlPro[12] to simplify the distribution and the installation of OCaml libraries. OPAM is entirely written in OCaml and can use various dependency constraint solvers including aspcud[13]. This is a source package manager, meaning that the packages are installed by compiling the sources.

This package manager is provided to the user through the command opam. We first initiate an OPAM folder where we will install our packages:

```
mkdir opam
opam init --root=opam
```

and setup the environment variables:

```
eval `opam config env --root=opam`
```

Then we have access to the standard OPAM operations:

```
opam show coq
opam install -v -j4 coq
opam list
opam search http
opam update && opam upgrade
```

By setting up OPAM in different folders, we can setup different development environments. Indeed, two different projects may need different versions of Coq or incompatible libraries. Another equivalent way to handle several development setups is to use the switch mechanism of OPAM, which creates one new folder per switch in the current OPAM folder.

[8]The list of the OPAM packages for Coq is available on coq.io/opam.

[9]The program OpamWebsite is available under MIT license on github.com/coq-io/opam-website.

[10]The Coq.io library is available under MIT license on coq.io.

[11]opam.ocamlpro.com

[12]www.ocamlpro.com

[13]The constraint solver aspcud is available on www.cs.uni-potsdam.de/wv/aspcud.

The OPAM packages are available on some repositories, activated with the command:

```
opam repo add repo-name repo-url
```

A repository is composed of one folder per package, containing three files: descr for a short textual description, `url` for the download link and opam for all the other metadata such as the compilation commands, the license, the homepage, etc.

4 Repositories

WE ORGANIZE THE OPAM REPOSITORIES for Coq[14] into three main repositories plus several distribution repositories. This complex architecture is set to fit the requirements of the users and the wills of all the Coq developers.

4.1 Stable packages

We put the stable packages in the released repository, which can be activated by:

```
opam repo add coq-released https://coq.inria.fr/opam/released
```

We consider as stable a package whose sources do not change (by opposition to a development branch for example, which is updated at every commits). The stable packages must follow some simple rules[15] in order to be accepted.

Installable The installation of a package must succeed when we start from a fresh OPAM setup. The uninstallation must revert the installation process by removing all the installed files.

Version The version names must be a triple of numbers separated by dots, like for example `3.2.0`. This convention is proven to fit a wide variety of projects, as it is the convention used by every npm packages.

The main rationale behind this convention is that, since we express package dependencies using inequalities over version numbers, the ordering over the version names should be as clear as possible. The ordering over triples of numbers in OPAM corresponds to the natural lexicographic order. By contrast, the OPAM ordering[16] over mixes of digits, letters and special characters is much

[14] Available on github.com/coq/opam-coq-archive.
[15] We continuously verify these rules thanks to the bench system.
[16] The OPAM ordering is the Debian ordering, described on www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Version.

more complex[17]. The other rationale is to follow the SemVer guidelines[18], which recommend to use three numbers to identify the *major* changes, *minor* changes and *patch* level changes.

Namespace To separate the Coq packages from the OCaml packages, we introduce a concept of *namespace* for OPAM by using the colon separator `":"`. In particular, all Coq packages must by prefixed by `"coq:"` to indicate that they are in the `"coq"` namespace. This idea of namespaces is just a convention which we can use for other projects too. For example, we prefix all the packages of the Coq.io project by `"coq:io:"`.

Name To ensure uniformity, the package names must be in small caps and use the dash `"-"` as a word separator. An example of a complete package name with a version name is the following:

$$\texttt{coq}:\texttt{io}:\texttt{hello}-\texttt{world}.1.1.0$$

Description Each package must contain a link to its homepage and provide its license information. The license information is mandatory because so many developers forget about specifying their license, making their projects *de facto* proprietary[19].

Parallel compilation Since OPAM is a source package manager, we must compile each package during the installation. In order to reduce the compilation time, we enforce the use of the `%{jobs}%` option in the build commands. This option represents the number of parallel processes allowed for the compilation, as specified by the `-j` or the `-jobs=` option of OPAM. The `Makefile` generated by the `coq_makefile` command can use this option to speedup the compilation process.

Checksum A package in the stable repository must come with the checksum of its sources, to ensure that the sources cannot be silently updated.

4.2 Development packages

We put the development packages in the two repositories extra-dev and core-dev:

```
opam repo add coq-extra-dev https://coq.inria.fr/opam/extra-dev
opam repo add coq-core-dev https://coq.inria.fr/opam/core-dev
```

[17]This led for example to the bug github.com/coq/opam-coq-archive/issues/8.
[18]semver.org
[19]See this study on the license usage for the GitHub repositories: github.com/blog/1964-open-source-license-usage-on-github-com.

The extra-dev repository is for the user packages, the core-dev repository for the development versions of Coq. Most development packages follow a Git branch of a project, and thus evolve at each new commit. We try to apply the same rules as for stable repository (except for the checksum and the version names, which typically contain strings such as `"beta"` or `"dev"`).

4.3 Distribution mechanism

There is a project of a distribution mechanism for the Coq packages. The idea is to provide a subset of mutually compatible packages, with at most one version per package, following the policy of the Debian distribution. To implement the fact that distributions are subsets of the stable packages, the distribution repositories would be composed of symbolic links to packages of the released repository.

5 Contribs

Historically, the user packages of Coq were regrouped on a single SVN repository[20], containing the source code of each package. To install a package, someone would download the source code and compile it using a standard:

```
make
make install
```

The source code of the Contribs was graciously maintained by some of the Coq developers and represents more than 150 projects.

We split the single SVN repository of the Contribs into individual Git repositories with one repository per project, keeping the history of the changes for each project. We created one OPAM package per project, keeping as much metadata as possible. Since the Contribs are not released with a version number, they point to Git branches (with one branch per Coq version in each project) and are hosted on the development repository extra-dev. We fixed many of the Contribs because we realized, during the import to OPAM, that the `make install` commands or the generation of the `Makefile` were often broken.

6 Bench system

In order to make sure that no packages are broken, we continuously run a bench system[21] to check the OPAM packages for Coq. We present the results in

[20] Hosted on GForge on coq-contribs.gforge.inria.fr.
[21] The results of the bench are available on coq-bench.github.io.

a colored table with the installation times for valid packages. The *best* column contains the best score obtained by each package.

We check that the packages are well-formed, using our lint[22] checker. We also check the installation of the dependencies and of the package itself, verifying that the uninstallation effectively reverts the installation process. We record the duration of each operation and compute the size of the installed files.

6.1 Architecture

The bench system is hosted on the GitHub organization `coq-bench`[23]. We will detail the four projects of the organization.

Run This Ruby program runs the benchmarks. For each bench we generate a clean Docker[24] image. Then we install OCaml, OPAM and Coq and test each package. Either the released repository is activated or both the released and the extra-dev repositories. We save the results into a database.

Database The database is a textual table in the CSV[25] format. There is one file per bench and one row per package. The first row gives a legend for each of the 31 columns.

Make-HTML This Ruby program generates static HTML pages representing the content of the CSV database.

Coq-Bench.GitHub.io These are the static HTML pages of the bench website. GitHub provides us a nice and simple way to host web pages from a Git repository using the GitHub Pages[26] service.

6.2 Strategies

There are many possible strategies to check all the packages with respect to their installation order. The installation order is important because OPAM will choose to install different dependencies in different contexts. Ideally, we would like to optimize the installation order to reduce the total execution time of the bench, by always installing and testing the dependencies first, so that no packages are compiled twice. Unfortunately this is not possible in general. Here is a simple counter example. With the list of packages presented in the Table 1, we must compile `B` twice (once with `A.1.0.0` and once with `A.2.0.0`) to test both `C.1.0.0` and `C.2.0.0`. Instead, we provide the two following (non-optimal) installation strategies.

[22]The sources of the lint checker are on github.com/coq-bench/run/blob/master/lint.rb.
[23]github.com/coq-bench
[24]Docker is a system simplifying the manipulation of Linux containers. It is available under Apache license on www.docker.com.
[25]en.wikipedia.org/wiki/Comma-separated_values
[26]The GitHub Pages services is presented on pages.github.com.

Package	Dependencies
`A.1.0.0`	$\emptyset$
`A.2.0.0`	$\emptyset$
`B.1.0.0`	`A` (any versions)
`C.1.0.0`	`A.1.0.0` and `B`
`C.2.0.0`	`A.2.0.0` and `B`

Table 1: Counter example to the optimal installation strategy.

Clean This is the simplest strategy. We install each package in a fresh environment with only Coq installed. This strategy is robust and reproducible, but not really optimal for packages with large dependencies since the dependencies are always recompiled from scratch.

Tree This is a more complex strategy. We install as many packages as possible until all new packages are incompatible with the current environment. The main source of incompatibility is the fact that we cannot install two packages with the same name but different version numbers. Once we are blocked, we roll-back until new packages are installable. This strategy is more clever but also more fragile and harder to maintain, to subtle changes of the `opam` tool for example. We use the branch mechanism of Git on the OPAM installation folder to switch efficiently between OPAM states. At the end of the process, we obtain a tree of all the Git branches used to explore the space of the packages (Figure 1).

7 Website

We developed a website[27] to present the list of the OPAM packages available for Coq. We statically regenerate this website every hours using the program OpamWebsite[28]. We entirely wrote this program using our Coq.io library to handle the inputs–outputs operations[29]. The inputs–outputs operations essentially consist in calling the `opam` command with the right options to grab the metadata of each package and then creating the new HTML files. We formally verify the OpamWebsite program using our method of formal use cases.

[27]The website of the Coq packages is available on coq.io/opam.

[28]The program OpamWebsite is available under MIT license on github.com/coq-io/opam-website.

[29]This process of using our own tools is called dogfooding.

```
* b99f693 coq:concurrency:pluto.1.0.0
* a1ff2f1 coq:concurrency:system.1.0.0
* 27c7b93 coq:moment.1.0.0
* d950f53 coq:list-string.2.0.0
| * 4817264 coq:flocq.2.2.0
| | * 3d4105d coq:flocq.2.3.0
| |/
| | * 544de40 coq:concurrency:proxy.1.0.0
| | * 9a1989e coq:coqeal:refinements.0.9.1
| | * 0bfd44f coq:fpmods.0.2.0
| | * bda2af7 coq:coqeal:theory.0.9.1
| | * 27aa320 coq:plouffe.1.0.0
| | * 35961d5 coq:coquelicot.2.0.1
| | * 89db498 coq:error-handlers.1.0.0
| | * aa85f8c coq:flocq.2.4.0
| |/
| * 48f33a4 coq:iterable.1.0.0
| * 0a48735 coq:function-ninjas.1.0.0
| * e2980d1 coq:list-plus.1.0.0
| * 4122f1a coq:list-string.1.0.0
|/
* 89705ad coq:math-classes.1.0.2
* 5d3ec5d coq:math-comp.1.5.0
* 470eefb coq:ssreflect.1.5.0
* b0205c8 Initial files.
```

Figure 1: Installation of a subset of the released repository.

8 Related work

A lot of programming languages have their own package managers. Our architecture and the decision to setup a package manager for Coq is influenced by the experience of the other programming language communities. Of course, we mainly got inspiration from the OCaml community, which developed the OPAM package manager. We do not try to compete with other package management systems. Instead, we try to reuse as much existing work as possible to design a simple package platform for Coq.

Historically, the way to share developments in Coq was to use the Contribs mechanism (page 6). This approach had some limitations. For example, the original authors of each project could not access to the code hosted on the central SVN repository, introducing a fork of each project. Instead, OPAM separates the metadata from the source code of each package. The Contribs also lacked a versioning system and a dependency manager, to automatically compute and install the correct dependency of a project through a command line tool. Finally, the acceptation policy for the Contribs was private. By contrast, the OPAM packages are published using public pull-requests on the OPAM repository for Coq[30]. We believe that all discussions concerning the Coq packages must be public in order to preserve the fairness of the platform.

To the best of our knowledge, no other theorem provers propose a modern package management system. The ProofPeer[31] project aims to enable collaborative works at a large scale, by design a new theorem prover integrating all the tools required for collaboration. We take the opposite direction, by reusing existing tools and concepts from the open source community, such as the idea of a package manager. The Mizar system[32] proposes the MML library[33], which is a centralized repository including the sources of more than 1200 projects. Each project is published as an article on the Journal of Formalized Mathematics[34] and peer reviewed before acceptation. Projects can express dependencies to other projects (but with no versioning). The technical architecture of the MML library is similar to the architecture of the Contribs repository of Coq, in the sense that all the source code is centralized, and thus suffers from the same limitations. The idea of a peer reviewed conference for the user contributions is interesting.

The OCaml community also made a bench system for the OPAM packages[35], but this system was not available at the time we made ours. The OCaml bench has the advantage of presenting a log of the differences between two consecutive benches, what make it easier to read when there are hundreds of packages to monitor. The website listing the OCaml packages for OPAM is written in OCaml.

[30] Available on github.com/coq/opam-coq-archive.

[31] The ProoPeer project is hosted on www.proofpeer.net.

[32] The Mizar system is available on mizar.uwb.edu.pl.

[33] We can explore the theorems of the MML library on mmlquery.mizar.org.

[34] fm.mizar.org

[35] The results of the OCaml bench are available on opam.ocaml.org/builds.

We rewrote this website in Coq to experiment our Coq.io library[36].

9 Conclusion

We have presented our work on the use of OPAM as a package manager for Coq. We especially worked on the design of a repository architecture and on the implementation of a bench system and a website for the OPAM packages for Coq. As of today, OPAM is usable for Coq and we hope that in the future more and more people will use OPAM to share their Coq developments and reuse other's contributions.

[36]The Coq.io library is available under MIT license on coq.io.