Reliably Reproducing Machine-Checked Proofs with the Coq Platform

Karl Palmskog, Enrico Tassi, Théo Zimmermann

Reliably Reproducing Machine-Checked Proofs with the Coq Platform

Karl Palmskog[1], Enrico Tassi[2], and Théo Zimmermann[3]

[1] KTH Royal Institute of Technology, 100 44 Stockholm, Sweden
palmskog@acm.org
[2] Univ. Côte d'Azur, Inria, France
enrico.tassi@inria.fr
[3] Inria, Univ. de Paris, CNRS, IRIF, UMR 8243, F-75013 Paris, France
theo@irif.fr

Abstract. The Coq Platform is a continuously developed distribution of the Coq proof assistant together with commonly used libraries, plugins, and external tools useful in verification projects. The Coq Platform enables reproducing and extending Coq-based artifacts in research, education, and industry, e.g., formalized mathematics and verified software systems. In this paper, we describe the background and motivation for the Platform, and outline its organization and development process. We also compare the Coq Platform to similar distributions and processes in the proof assistant and wider open source software community.

Keywords: Coq · proof assistants · software engineering · software delivery · reproducible builds.

1 Introduction

The Coq Platform (*Platform* for short) is a continuously developed distribution of the Coq proof assistant together with commonly used Coq libraries, Coq plugins, and external tools useful in Coq-based verification projects. The Platform enables reproducing and extending Coq-based artifacts in research, education, and industry, such as formalized mathematics and verified software systems.

In this paper, we describe the background and motivation for the Platform, and outline its organization and development process. Through the adoption of Coq Enhancement Proposal (CEP) 52 [16], Platform development has become an integrated part of the development and release process of Coq itself, with Coq core team members comprising half the Platform team. The Platform also provides release coordination for projects in the Coq ecosystem, both explicitly by asking maintainers for Platform project releases and implicitly by announcing new Platform versions that become a basis for releases of non-Platform projects.

We compare the Platform to similar initiatives for reliably reproducing software artifacts, both in the proof assistant community and in the wider open source software community. Our description and comparisons are based on Platform release 2022.01.0 [2], which provides Coq versions 8.12 to 8.15, along with curated and tested packages for each version.

2 Coq Overview

Coq is a proof assistant based on a dependent type theory, implemented mainly in OCaml and maintained as open source software on GitHub [3]. To an end user, Coq provides a formal language to write datatypes, functions, and theorems, together with an environment for semi-interactive development of machine-checked proofs. Coq consists of several parts:

- Coq's surface-level language is the *vernacular*, which is an extensible language of commands written as a sequence of *sentences*. Commands can be queries into definitions, declarations of new definitions using the Gallina specification language (e.g., of functions and datatypes), or tactics to run.
- Coq *elaborates* user-written vernacular into a logical formalism, which is an extension of the Calculus of Inductive Constructions, a foundational theory of dependent types.
- After elaboration, Coq's *kernel* certifies that a *term* has a proposed *type*. For example, a type may be the statement of a theorem, and the term its proof.

Both Coq's elaborator and kernel can take significant time to execute for some vernacular code, e.g., due to long-running custom tactics that build terms or heavy use of Coq's built-in proof search methods. Moreover, Coq tactic commands (proof scripts) are seldom written to explicate the intuition behind proofs.

Consequently, a user may not be convinced that a formal Coq proof (term) of a statement (type) exists unless it is reproducible locally. Hence, a typical mode of use of Coq is to check publicly available vernacular code related to specific applications in mathematics and computer science. However, this code typically depends on many general public *libraries*, in addition to Coq's own standard library (Stdlib). For example, Stdlib contains formalizations of natural numbers, integers, and real numbers, but formalizations of floating points, regular expressions, and matrices are found in third-party libraries.

The Platform provides a way to install both Coq and such libraries in a controlled way. Notably, the Platform provides the CompCert compiler [9] and the Verified Software Toolchain [1] and their dependencies, which enables formal verification of functional correctness of C programs in Coq.

3 Platform Usage

Since 2021, the Platform is the officially recommended way to install Coq on its website [6], due to its accommodation of a large range of use cases. The Platform comes in two flavors:

- *Binary installers* are the simplest and fastest way for a new user to get a working installation of Coq, batteries-included (as the platform includes many third-party packages). However, its main limitation is that it is not extensible (users cannot change the set of packages available to them).

– *Interactive scripts* provide a cross-platform solution to get Coq and third-party packages installed with the opam package manager, by handling the installation of any required dependency, and by taking care of running the appropriate opam commands. After that, users can fully customize the set of available packages (and their versions) by using opam directly.

Hence, as long as people using the scripts are not editing the resulting set of packages, the Platform provides a reliable cross-platform solution for making sure that different people running different operating systems get the same set of Coq package versions. This can simplify the life of professors teaching Coq to their students, but also of scientific reviewers trying to reproduce (re-execute) a Coq artifact. However, this assumes using a specific, agreed-on version of the Platform. Without the Platform, compatibility between Coq packages is not guaranteed, and users may end up in "dependency hell" [1].

4 History and Goal

4.1 History: Coq Contribs and the Coq opam archive

Coq Contribs (CCs), which are standalone libraries or plugins for Coq, have been around since 1993 [18]. CCs followed a model where the author of a Coq formalization hands over all maintenance to Coq developers. CCs were available for public download, but the repository hosting them was private in the sense that no contributor other than the Coq developers had write access (thus, excluding the original author). Hence, actively developed projects were forced to live outside this walled garden.

Around 2013, Thomas Braibant came up with the idea of using OCaml's package manager, opam [13], also for Coq packages. This idea was implemented one year later by Guillaume Claret and the second author of this paper in the form of the Coq opam archive on GitHub [4] and the Coq Package Index [5]. Officially adopting a package manager greatly simplified the distribution of libraries from Coq users to other Coq users, and currently the Coq opam archive hosts thousands of packages, including most of the current Platform packages. The remaining Platform packages are hosted in the general OCaml opam archive.

4.2 History: Coq's Windows installer

Coq has primarily been developed for Unix-like operating systems, and its implementation language OCaml has had limited Windows support. As a result, it has always been necessary to provide a pre-built Windows installer.

In 2017, Michael Soegtrop started to include third-party Coq packages into the Windows installer to encourage evaluation and eventually adoption of Coq in industry. He also started to distribute the sources that were used to build the installer for reproducibility (and licensing) concerns. This augmented installer received positive feedback, in particular from non-academic users. In fact, many users would not consider using any Coq library not part of the installer—both

due to the complexity of installing new dependencies and lack of trust that the dependency would be maintained in the future.

In 2019, during a Coq developers meeting in Nantes (and following some earlier discussions on the Coq-Club mailing list), Soegtrop announced the Platform project and published the first revision of its charter [14].

Up until the release of Coq 8.12 in summer 2020, the Coq release process included the task of building the Windows installer (and the MacOS dmg archive). The effort of this task had grown considerably over time, and the skills and motivation needed in order to fix problems arising from this task were not abundant in the Coq development team (from which the Coq release managers are chosen). In addition, the fact that the Windows installer shipped with several external packages created a synchronization problem, since Coq developers had to select compatible versions of some packages that depend on Coq *before* the Coq release.

At the time where Coq 8.13 was supposed to be released in January 2021, the scripts to build binary installers were not functional anymore. At the same time, the installers in the Coq Platform were functional. As a result, 8.13 was the first release of Coq where binary packages were built using the Platform scripts, although some missing binary packages for 8.12 were also later built using the Platform a posteriori.

These events finally led to the publication of CEP 52 [16] by the second author, that documents how the release of Coq and the release of its Platform is currently organized, which is described in Section 5.1.

4.3 Platform goal and problems addressed

The current goal of the Platform is to provide a distribution for developing and teaching with Coq that is *operating system independent*, *dependable*, *easy to install*, and *comprehensive*. As part of the process of achieving this goal, we believe the Platform can partially address a number of long-standing problems that we have experienced in the Coq community and ecosystem:

Release coordination. Coq libraries may not provide releases for a new Coq version. By being included in the Platform, developers are incentivized to provide releases and can easier collaborate with Coq developers and users to support new Coq versions.

Compatibility. Coq libraries and plugins may not provide mutually compatible releases. In the Platform, projects are continually tested for compatibility, and Platform releases do not include mutually exclusive packages.

Build automation Many Coq projects do not use best practices for Coq building and testing, and may not work across different operating systems. The Platform can help disseminate knowledge of such practices, and tests its packages on several operating systems.

Reproducibility of research artifacts. Coq research artifacts have historically been distributed as opaque vernacular file collections (e.g., tarballs). By advising authors to target and document a specific Platform version, scientific venues can ensure Coq artifact reproducibility.

Upgrade paths between Coq versions. Coq users may hold off upgrading to a new Coq version because one of the libraries they depend on is not available. The Platform ensures that there is an upgrade path available from one Coq version to another with a consistent collection of packages that should only increase over time.

5 Platform Development and Organization

5.1 Platform development and release process

Since CEP 52 [16], the release processes of Coq and the Platform (and the binary installers) are handled separately (by different, possibly overlapping, teams). This simplifies the job of the Coq release managers, who can now focus on the one piece of software they know best. The job of the Coq Platform team begins when the Coq team publishes a first release candidate Coq version, on top of which the Platform can start being built. When the new stable Coq version tag is set a few weeks later, it is not announced to the user community. The new Coq version is announced to the users only when it becomes included in a Platform release, first as a beta and then as a stable release.

In order to produce a new Platform release, maintainers of Platform packages need to be involved. As soon as a Coq release candidate is published, it is made available in a development-only opam repository, and Docker images are built on top, to facilitate compatibility testing in Continuous Integration workflows. The maintainers of packages that are part of the Platform are informed by the Platform team that a new release is happening and that their participation is needed to determine the version of their package that will be included. Based on the tests performed on the Platform (see subsection 5.3), they may be informed that a certain package version or commit is already compatible, or that there are no known compatible version of their packages. In both cases, they should decide which version they would like to have included, and make a new package release if needed. So far, for three Platform releases, the community of package maintainers has been responsive to this call, accepting the proposed version or providing a new working version in a matter of weeks, although, for a few large and complex packages, the Platform team had to assist the developers.

5.2 Platform versioning scheme and organization

During the discussion of CEP 52, it was decided that the Platform would use a calendar-based versioning scheme, independent of the Coq versions it includes, to reflect that the Platform is a distribution, not only of Coq, but also of many other packages, that have their own release cycle. Since September 2021, this has allowed an additional (technical and organizational) change in the way the Platform is maintained and published. Instead of being tied to a specific Coq version, Platform releases now include several "package picks", and the user can decide which one to install by selecting the appropriate binary installer, or the appropriate option in the interactive scripts.

This change serves several purposes. From the user point of view, old package picks are still available in the next versions of the Platform, and thus users can always rely on the latest version. The interactive scripts make it possible to install several picks in parallel, and thus, this provides a smooth upgrade experience from one pick (and one Coq version) to the next. From the Platform maintainers' point of view, this allows maintaining several Coq versions and package picks as part of a single branch, and thus to factorize any improvements to the infrastructure, but also any fixes that are independent of the Coq version (for instance, when a new Cygwin version or Ubuntu version introduce changes that break the compilation of Coq dependencies, such as OCaml).

5.3 Platform Continuous Integration and Delivery

The Coq Platform uses GitHub Actions for Continuous Integration and Delivery (CI/CD). Initially used only to build binary installers for Windows, it was then extended to build the MacOS dmg and finally a Snap package for Linux. It also serves to test the interactive installers. The Platform includes a "smoke test kit", a mild test, to be run on the binary installers, once installed: It double checks all shipped packages can be actually loaded and used, catching problems in the installers themselves.

CI was also greatly simplified when the same branch and scripts were able to support multiple package picks. The CI configuration files are thus factorized as well, and rely on the common "matrix" feature to test multiple picks.

Finally, users have used the CI/CD setup to build custom Platform installers, typically to override package versions and include extra packages for teaching purposes. By forking the repository, performing changes to the package pick definitions, and then letting CI run (in their fork or in a draft pull request on the main repository), they can produce binary installers for all the major operating systems, without needing to have these systems at hand.

6 Platform Role in the Coq Ecosystem

Historically, CCs served several roles at once: distribution of third-party developments to the users, long-term maintenance, and testing. In the current ecosystem, these roles have been split in three parts.

The Coq package index (based on the opam package manager) is the channel for distributing third-party developments. This means that these packages are now under the control of their own authors (who manage their evolution).

However, useful packages can stop being maintained by their authors. This is a phenomenon which happens in every package ecosystem, but it is particularly relevant in the Coq ecosystem, since many packages were initially produced as paper artifacts or as part of PhD theses. When this happens, interested users can pick up their maintenance by adopting them in Coq-community (an organization that was created in 2018 for this purpose and which now hosts nearly 60 projects maintained by over 30 people).

Finally, Coq developers test their changes against an extended test suite consisting of many (actively maintained) external projects. When they decide to merge a change that breaks a project in this suite, they write compatibility fixes for this project and submit it to the project's maintainers. This means that, in practice, Coq developers still participate to the long-term maintenance of some important Coq packages. However, they are only responsible for producing compatibility fixes, but the roadmap and evolution of the packages and their release schedule still remain under the control of their original authors (or current maintainers).

In this context, the role of the Platform is not only to provide an easy solution to install Coq with external packages, but also to define a set of packages for which their maintainers have agreed to abide by certain rules (such as timely release of compatible versions, and care in not making too many breaking changes). The implicit coordination signals transmitted by Platform releases help the Coq ecosystem catch up to new Coq versions more quickly than previously, when relatively few third-party packages were available in the months following a Coq release [8].

As Coq projects evolve, they may become incompatible with other (dependent) projects, e.g., due to architectural changes that cannot be accommodated with reasonable effort. Since maintainers of such projects have committed to Platform inclusion for the long term, they have an incentive to find solutions to such compatibility issues together with Platform maintainers, e.g., through the introduction of new packages alongside previous packages that facilitates slow-paced user migration.

7 Comparisons

7.1 Ecosystem-specific "platforms" and distributions

Naturally, the Coq Platform is quite comparable with many similar "platforms" and distributions which it takes inspiration from (e.g., the Haskell Platform, the Scala Platform, TeXLive, etc.). These distributions are all centered around the concept of making it easier to access third-party packages and providing a "batteries included" experience to beginners.

However, while some such distributions, like TeXLive, have been largely successful, some others have not: the Haskell Platform was deprecated in 2022. Many technical and social factors can enter in consideration and result in eventual failure or success, and the reasons why the Haskell Platform was deprecated are not documented. Nevertheless, one can notice that the Haskell ecosystem has a similar (and very much alive) solution to provide well-defined sets of packages at compatible versions: Stackage. Similarly, to the Coq Platform, Stackage is based upon a social contract [15].

7.2 Linux and other Linux-like distributions

On the opposite side of the spectrum, these distributions provide consistent set of packages for important software pieces, including proof assistants such as

Coq, and sometimes also libraries. However, in rapidly evolving ecosystems, it quickly becomes impossible to provide a large set of compatible libraries without coordination with their maintainers. When coordination mechanisms are put in place (such as with the Coq Platform, or with Stackage), this can provide ways for distribution maintainers to provide a larger set of packages, by relying on the documented sets of compatible versions. This is why Linux distributions virtually always provide a package for TeXLive. Similarly, we could expect that (if there was enough interest for Coq libraries), distribution maintainers could provide packages matching the content of the Coq Platform. We aim to provide documentation to facilitate this endeavour.

7.3 Isabelle and the Archive of Formal Proofs

The Isabelle generic proof assistant, and in particular its Isabelle/HOL instantiation, are the basis for the Archive of Formal Proofs (AFP) [7].

Updates to AFP entries are primarily done by Isabelle developers when code breaks due to Isabelle evolution, rather than by the authors of the entries, as in the Platform. While Coq developers are participants in Platform maintenance, they normally only update projects that are in Coq's Continuous Integration (CI) suite. Not all Platform projects are in Coq's CI, and Coq's CI contains many non-Platform projects.

The focus in the Platform is on including generally useful libraries, plugins, and tools—not novel research artifacts. Coq research artifacts can be submitted as packages to the Coq opam archive. In contrast to AFP, the Coq opam archive only performs minimal reviewing and updating of submitted packages.

Each Platform release defines several Coq versions, and each Coq version may have several different package collections ("picks"). In contrast, the AFP works with a single Isabelle version at any given time.

7.4 Lean and Mathlib

The Lean prover [11,12] has similar foundations to Coq, but its development and ecosystem is differently organized. In contrast to Coq, which aims to provide upgrade paths between versions, Lean 3 and Lean 4 are largely incompatible. Lean has a single significant large library, mathlib [17], that contains an extensive collection of formalized mathematics and is actively developed in a single repository on GitHub [10].

The monorepository approach used for mathlib comes with several advantages, not least of which are avoiding version management. Coq projects must continually decide whether they should do a release compatible with a new Coq version, while mathlib is continually compatible with the latest version of Lean 3. The switch of mathlib from Lean 3 to Lean 4 is expected to happen atomically, rather than using versioning as for Coq libraries.

Thanks to the cross-platform binary format for elaborated terms used by Lean, mathlib can be straightforwardly distributed both in source and binary

form, which generally leads to a faster setup than for comparable Coq libraries. However, we believe the operating-system-specific binary distributions of the Platform have comparable ease of use.

8 Conclusion

In this paper, we presented the history and current state of the Coq Platform, the official distribution of the Coq proof assistant, and its aim to improve Coq-based artifact reproducibility.

In recent years, venues such as POPL, IJCAR, FLOC and ETAPS have seen an increasing number of submissions supported by Coq artifacts. In specialized conferences such as CPP or ITP, nearly all submissions come with a proof assistant artifact as their main contribution, many of which use Coq.

We believe that current standard for evaluating such artifacts are too low even at CPP or ITP. In particular, there is no strict requirement for the artifact to be reproduced by the reviewers. In fact, the program committee chairs are usually more than happy if one reviewer (out of typically three) manages to do it, since recreating an environment where the artifact can actually be inspected can be extremely difficult and time consuming.

The Platform will hopefully make the task of evaluating Coq artifacts easier and less time consuming, not only in specialized venues, but also in broader ones where program committee members may have less Coq experience.

Acknowledgements We are grateful to Michael Soegtrop, all members of the Coq Team, and the developers of all Platform packages.

References

1. Appel, A.W.: Coq's vibrant ecosystem for verification engineering (invited talk). In: International Conference on Certified Programs and Proofs. pp. 2–11 (2022). https://doi.org/10.1145/3497775.3503951
2. Coq Platform Team: Coq Platform 2022.01.0 (2022), https://github.com/coq/platform/releases/tag/2022.01.0
3. Coq Team: Coq GitHub repository (2022), https://github.com/coq/coq
4. Coq Team: Coq opam archive (2022), https://github.com/coq/opam-coq-archive
5. Coq Team: Coq Package Index (2022), https://coq.inria.fr/opam/www/
6. Coq Team: Coq Website (2022), https://coq.inria.fr/download
7. Eberl, M., Klein, G., Lochbihler, A., Nipkow, T., Paulson, L., Thiemann, R.: Archive of Formal Proofs (2022), https://www.isa-afp.org
8. Kohlhase, M., Rabe, F.: Experiences from exporting major proof assistant libraries. J. Autom. Reasoning **65**, 1265–1298 (2021). https://doi.org/10.1007/s10817-021-09604-0
9. Leroy, X.: Formal verification of a realistic compiler. Commun. ACM **52**(7), 107–115 (2009). https://doi.org/10.1145/1538788.1538814
10. Mathlib Team: Lean mathlib (2022), https://github.com/leanprover-community/mathlib

11. de Moura, L., Kong, S., Avigad, J., van Doorn, F., von Raumer, J.: The Lean theorem prover (system description). In: Automated Deduction. pp. 378–388 (2015). https://doi.org/10.1007/978-3-319-21401-6_26
12. de Moura, L., Ullrich, S.: The Lean 4 theorem prover and programming language. In: Automated Deduction. pp. 625–635 (2021). https://doi.org/10.1007/978-3-030-79876-5_37
13. OCaml Team: OCaml Package Manager (2022), https://opam.ocaml.org
14. Soegtrop, M.: Coq Platform Charter (2019), https://github.com/coq/platform/blob/90cf0da1a2e65adbe6885735b2708d61f87d5799/charter.md
15. Stackage Team: Stackage Maintainers Agreement (2021), https://github.com/commercialhaskell/stackage/blob/master/MAINTAINERS.md
16. Tassi, E.: Coq and Coq Platform Release Cycle (2021), https://github.com/coq/ceps/blob/master/text/052-platform-release-cycle.md
17. The mathlib Community: The Lean Mathematical Library. In: International Conference on Certified Programs and Proofs. pp. 367–381 (2020). https://doi.org/10.1145/3372885.3373824
18. Zimmermann, T.: Challenges in the collaborative evolution of a proof language and its ecosystem. Ph.D. thesis, Université de Paris (2019), https://hal.inria.fr/tel-02451322