Hydras & Co.: Formalized mathematics in Coq for inspiration and entertainment

Pierre Castéran, Jérémy Damour, Karl Palmskog, Clément Pit-Claudel, Théo Zimmermann

Hydras & Co.: Formalized mathematics in Coq for inspiration and entertainment

Pierre Castéran[1], Jérémy Damour[2], Karl Palmskog[3], Clément Pit-Claudel[4], and Théo Zimmermann[5]

[1] Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400 Talence, France
[2] Univ. de Paris, F-75013 Paris, France
[3] KTH Royal Institute of Technology, Stockholm, Sweden
[4] MIT CSAIL, Cambridge, Massachusetts, USA
[5] Inria, Univ. de Paris, CNRS, IRIF, UMR 8243, F-75013 Paris, France

Abstract

Hydras & Co. is a collaborative library of discrete mathematics for the Coq proof assistant, developed as part of the Coq-community organization on GitHub. The Coq code is accompanied by an electronic book, generated with the help of the Alectryon literate proving tool. We present the evolution of the mathematical contents of the library since former presentations at JFLA meetings. Then, we describe how the structure of the project is determined by two requirements which must be continuously satisfied. First, the Coq code needs to be compatible with its ever-evolving dependencies (the Coq proof assistant and several Coq packages both from inside and outside Coq-community) and reverse dependencies (Coq-community projects that depend on it). Second, the book needs to be consistent with the Coq code, which undergoes frequent changes to improve structure and include new material. We believe Hydras & Co. demonstrates that books on formalized mathematics are not limited to providing exposition of theories and reasoning techniques—they can also provide inspiration and entertainment that transcend educational goals.

1 Introduction

1.1 Background

Libraries of formalized mathematics based on proof assistants, such as Coq, Lean, and Isabelle/HOL, are continually growing in size and scope. For example, Mathlib for Lean and the core set of projects in the Mathematical Components family for Coq both amount to hundreds of thousands of lines of code (LOC) and tens of thousands of definitions and theorems [36, 35]. However, after a key mathematical definition or result is added to a library, it must be *documented and maintained* [51, 41].

Maintenance includes not only adaptation to changes in new proof assistant versions, but also reorganization to accommodate new contributions. Documentation is usually two-fold: *source code comments* that describe specific definitions and results in-line and *books* that carefully introduce the library and its idioms [32, 2]. The former tends to be terse and dense and serves experienced users, while the latter is more long-winded and exhaustive and serves beginners. As a library changes and expands, all its documentation needs to be made consistent and complete. Authors use many techniques, including literate programming [30] and custom tools, to pretty-print and check source code snippets and generate proof assistant output.

While books that document proof assistant libraries are valuable to beginners, the purpose of recently published books is mainly instrumental, i.e., to teach a certain topic or technique. We believe that books and libraries can instead become *ends in themselves*—not just sources of exposition and learning, but also sources of inspiration and entertainment.

1.2 Vision

The Hydras & Co. project, part of Coq-community on GitHub [14], aims to be an experimental platform for the collaborative development of documented libraries of formal proofs. Coq-community is a community organization that the first and last authors founded in 2018 with two goals in mind: providing a solution for the long-term maintenance of interesting open source Coq packages, and working collaboratively on documentation projects. The Hydras & Co. project demonstrates that these two goals are not independent: interesting Coq packages can become the basis for new documentation. This umbrella project now includes evolved versions of the former Cantor and Additions libraries [12, 9] (under the new names of Hydra-battles and Addition-chains), the Ackermann sub-library extracted from Russel O'Connor's Goedel library [38, 37], and a preliminary bridge to the Gaia library by José Grimm [25, 22]. By following this approach of commenting interesting Coq packages, we provide new documentation content that contributes to the diversity of the thriving Coq ecosystem.

We call on the Coq users in the JFLA community and beyond to come and join us in this effort, by bringing new interesting projects which are worth presenting to Coq learners, *a.k.a.* Coq users, and guiding them in their exploration. We also always have project ideas to extend further our explorations and anyone is welcome to join the team by sending small or larger contributions through pull requests. The current state of the project is already the result of such evolutions after several of us contributed project solutions and new proposals to the initial version of the first author.

Futhermore, contrary to traditionally published books, the "book" that forms part of this project is intended to be forever evolving. As new Coq formalization patterns and proof techniques appear, the book can be adapted to demonstrate their use (in case they fit well with our applications). By using modern maintenance techniques such as continuous integration and deployment, we can ensure that this documentation stays up to date with the latest Coq releases. With Alectryon [41, 40], we ensure that code and documentation are always in sync.

1.3 Hydra games

We chose to build our library and book on two simple themes which allow many variations: computing powers in a monoid, and Kirby and Paris' hydra battles. In the interest of space, we will only present the second theme in this paper.

Hydra games (also known as *Hydra battles*) appear in an article published in 1982 by two mathematicians, Laurie Kirby and Jeff Paris: *Accessible Independence Results for Peano Arithmetic* [29]. This article describes a game between two players: Hercules and a hydra. A short description of the game can be found in [4, 29, 11]. One can also play with Andrej Bauer's simulator [3]. In a few words:

- A hydra is a finite tree, traditionally presented with the root at the bottom, the leaves of which are called *heads* (Figure 1).
- At every round, Hercules chops off one head of the hydra. If the head is at a distance greater than 1 from the root, then some sub-tree h of the hydra is copied a certain amount n of times. The number n of copies and the sub-tree h may depend on the considered variant of the game and the time elapsed since the beginning of the fight. Figure 1 shows an example with $n = 2$.

Kirby and Paris proved the following theorems, applying combinatorial results about ordinal numbers by Jussi Ketonen and Robert Solovay [28].

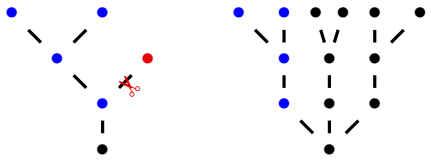

Figure 1: Two successive states of a hydra in a battle. Hercules chopped off the rightmost head of the hydra (red), and the whole left tree except the root node (blue) was copied twice.

Theorem 1. *In the Hydra game, Hercules eventually wins, whichever the strategy of both players: choice of a head to chop off, choice of the number of copies.*

Theorem 2. *Theorem 1 cannot be proved in Peano Arithmetic.*

The contrast between the simplicity of the statements above and the complexity of their proofs convinced us that it is a good theme for a commented library [26] of formal proofs written for the Coq proof assistant [19].

Complex formalizations and proofs are explained in the book. Whenever various reasonable choices exist, we try to present and compare the alternatives. For instance, Figures 7 and 8 show two radically different proofs of the equality $\omega + 42 + \omega^2 = \omega^2$. The first one is a simple proof by computation, the second one shows how this equality is a consequence of the axioms of the set-theoretic model by Kurt Schütte [43].

This work is also an opportunity to provide concrete examples of formalization and proof techniques: operational type classes, functions defined by equations, dependently typed functions, etc. It may be also used as a library on ordinal numbers, for instance for proving termination properties. Prior stages of this project have already been presented at JFLA [10, 11]. In this paper, we present recent evolutions of the library: new results, interaction with the Coq-community project [14], and documentation generated with Alectryon [41, 40].

2 Recent developments

The 2018 article [11] presented a formal proof of the following theorem:

Theorem 3. *Let μ be any ordinal strictly less than ϵ_0. There is no function mapping hydras to the segment $[0, \mu)$ that could be used as a measure in proving termination for* any *hydra battle.*

Our proof was based on the construction of a battle between Hercules and the hydra where the number of copies at each round was given by the elimination of an existential quantifier. So, it was mandatory to consider the class of *all* hydra battles, or, equivalently, the class of battles where the hydra choses arbitrarily its number n of copies at every round of the battle.

Unfortunately, the examples most commonly shown in the literature (see for instance [29, 4, 3]) assume that the hydra grows k copies at step k of the game, which is incompatible with our proof. We prove now that Theorem 3 still holds with these typical battles. Since we are looking for a minoration of the length of such battles, we can work with the following hypotheses and invariants, without any loss of generality.

- The game starts at an initial step $k = i$, where i is any natural number.
- Hercules always chops off the rightmost head of the hydra.

- The hydra is always the tree representation of some ordinal strictly below ϵ_0 in Cantor normal form (thus, the rightmost branch is also one of the shortest). For instance, Figure 1 shows the hydras respectively associated with ω^{ω^2+1} and $\omega^{\omega^2} \times 3$.

In mathematical terms, if at step $k \geq i$, the hydra is associated with the ordinal α, at step $k+1$ it is associated with $\{\alpha\}(k+1)$, *i.e.* the $(k+1)$th element of the canonical sequence of α [28]. In the following, the expression "the hydra α" is an abbreviation of "the hydra associated with the ordinal α".

Our new proof of Theorem 3 is based on a systematic study of strictly decreasing sequences of ordinals below ϵ_0, borrowed from Ketonen and Solovay [28], and the formalization of which is described in detail in chapters 5 and 6 of [13].

Besides Theorem 3, we study also the number of steps of a battle: Let $\alpha < \epsilon_0$ be an ordinal. We prove that the number of steps of the battle starting with α at step i is greater or equal than $H'_\alpha(i) - i$, where H' is a slight variant of the Hardy hierarchy of rapidly growing functions [53, 28, 42, 52]. The function H'_α is defined by transfinite recursion over α on Figures 2 and 3.

$$H'_0(i) = i \tag{1}$$
$$H'_\alpha(i) = H'_{(\{\alpha\}(i+1))}(i) \quad \textit{if } \alpha \textit{ is a limit ordinal} \tag{2}$$
$$H'_\alpha(i) = H'_\beta(i+1) \quad \textit{if } \alpha = \beta + 1 \tag{3}$$

Figure 2: The H' rapidly growing hierarchy of arithmetical functions

```
Equations H'_ (alpha: E0) (i:nat) :  nat  by wf  alpha Lt :=
  H'_ alpha  i with E0_eq_dec alpha Zero :=
    { | left _zero =>  i ;
      | right _nonzero
          with Utils.dec (Limitb alpha) :=
          { | left _limit =>  H'_ (Canon alpha (S i))  i ;
            | right _successor =>  H'_ (Pred alpha) (S i)}}.
```

Figure 3: H' definition using the Equations plug-in.

Using H's equations as rewrite rules, we can study a concrete example. We take the hydra of figure 4 and $i = 0$ as initial configuration. By a sequence of rewritings and inductions, we prove that the number of steps of the considered battle is greater or equal than 2^{2^N}, where $N = 2^{70} - 1$. By comparison with the diagonalized Ackermann function $\lambda i.\, A(i,i)$, we prove also that, for $\alpha \geq \omega^\omega$, the function computing the length of the battle starting with the hydra α and the initial step i is not primitive recursive.

Proof engineering improvements. The original release of the Cantor library preceded support for type classes in Coq [48]. Changes over time before the move of Hydras & Co. to Coq-community have since then introduced type classes for some concepts in the Hydra-battles library related to ordinals, e.g., to provide a uniform way to compare ordinals within different representation systems. Most type classes initially used a *semi bundled* approach [35], but we

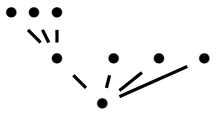

Figure 4: The hydra associated with the ordinal $\omega^3 + 3$

have recently *unbundled* classes to enable more sharing during resolution. For instance, we use the (single-field) `Decision` type class [49], that describes deciding a proposition, in parameters to other classes rather than in their fields. Due to the modest depth of our class hierarchy, we believe unbundling will not lead to scalability issues. Orthogonally to type classes, we introduced automation for definitions using the Paramcoq [27] and Equations [47] plug-ins, as illustrated in Figure 3.

3 Integration with Coq-community

3.1 Background

Coq-community is an informal organization run by volunteer Coq users on GitHub that aims to maintain interesting open source Coq projects for the long term and facilitate collaboration among Coq users on documentation, tooling, etc. Coq-community was created in 2018, inspired by the Elm Community organization [54]. Such "Community Package Maintenance Organizations" exist in many software ecosystems, as they avoid the common problem of an important package becoming unmaintained after its author has moved on to other projects or has disappeared [55].

In the case of Coq, this problem is likely even more prevalent than in other ecosystems, as many packages are created by graduate students or researchers for a specific paper and not planned to be maintained for the long term by authors. However, authors are generally open to having someone else who expresses interest in their work take over package maintenance.

Coq-community makes it easy to change maintainers by defining a process for transferring or forking an unmaintained package, tooling for setting up good maintenance practices (such as continuous integration), and by making it possible for someone to take over a package without making a long-term commitment (as maintainers who drop out can easily be replaced by some other volunteers).

As of 2021, Coq-community hosts over 50 projects maintained by over 30 volunteers. The hosted projects come from a variety of origins. Some had been maintained in the past by the Coq development team on behalf of the authors, but this meant that only minimal changes required to make the project build with new Coq versions were performed. Some were still maintained by their original authors, but were transferred to enable other users to help out with maintenance and facilitate adoption of best practices on, e.g., continuous integration. Others were simply unmaintained and were revived after their transfer to Coq-community.

Given the objectives of Coq-community, we believe it makes sense to propose a transfer anytime we encounter an interesting Coq project that is insufficiently maintained. After a transfer, the Coq-community maintainers are explicitly allowed to perform large changes and refactorings. This means, for instance, that we can consolidate packages by merging them, or split up a single package into several packages.

3.2 Integration of primitive recursive functions

In order to prove formally that the length of the kind of hydra battles we consider is not given by any primitive recursive function, we chose to use a formalization of primitive recursive functions that was originally part of Russell O'Connor's formal proof of Gödel's first incompleteness theorem [38, 37]. For this purpose, and above all in consideration of the scientific interest of this contribution, we decided to host and maintain O'Connor's work in Coq-community.

Since computability is a key topic in computer science teaching and O'Connor's library is a nice illustration of dependently typed programming, we decided to devote a full chapter (chapter 9 of [13]) to the formalization of primitive recursive functions, with comments on the definitions and proofs and (counter-)examples and exercises. As part of the writing process, we made the formalization into a new sub-library of the Hydra-battles library, dubbed Ackermann.

3.3 Towards a bridge to Gaia

The Gaia project by José Grimm aimed to formalize mathematics in Coq in the style of Nicolas Bourbaki. The formalization of the first book in the Elements of Mathematics series by Bourbaki, on the theory of sets, was initially described in a technical report in July 2009 [20]. The set-theoretic axioms and basic definitions in Gaia were derived from an earlier development by Carlos Simpson [46, 45]. Grimm then wrote (and continually updated) technical reports describing the formalization of Bourbaki's two subsequent books [21, 24] and additional topics in number theory [22, 23], before he passed away in 2019.

In 2020, members of Coq-community transferred the Gaia source code to GitHub and adapted it for recent releases of the Mathematical Components library, which Gaia heavily relies on. Anonymous volunteers ("collaborators of Nicolas Bourbaki") then finished the only in-progress proof left by Grimm. At around 155,000 LOC, Gaia is currently one of the largest maintained open source Coq projects [25].

Gaia contains definitions of ordinals in Cantor and Veblen normal form [22], adapted from the historical Cantor contribution [12]. The data types for ordinals are essentially defined the same way as in Hydras & Co., but they are not identical inside Coq, e.g., due to residing in different modules. There are also minor differences in how ordinal arithmetic is implemented, due to the different evolutionary paths taken since divergence from the ancestor library.

As an initial step in bridging Gaia and Hydras & Co., we were able to establish a correspondence between both implementations of ordinals through rewriting lemmas. For instance, we proved that multiplication of ordinals in Cantor normal form in Hydras & Co. refine Gaia's corresponding multiplication operation, and then imported Gaia's proof of associativity of the multiplication almost for free.

The initial draft bridge code mostly uses the SSReflect proof language and idioms from the Mathematical Components library. We made this design decision since we believe it is less challenging to reason about Hydra-battles code using SSReflect and MathComp than to reason about Gaia without SSReflect.

3.4 Package Genealogy, Dependencies, and Organization

Both due to its prior stages [10, 11] and the recent integrations described just above, the current Hydras & Co. Coq code has a complex inheritance. Figure 5 illustrates the relationships between Hydras & Co. packages and show their ancestry from historical Coq contributions. To indicate the scope and relative sizes of packages, Table 1 breaks down lines of code for each package in

the Hydras & Co. galaxy, as reported by the `coqwc` tool, and lists their repository names in Coq-community and identifiers in the Coq opam package index [18].

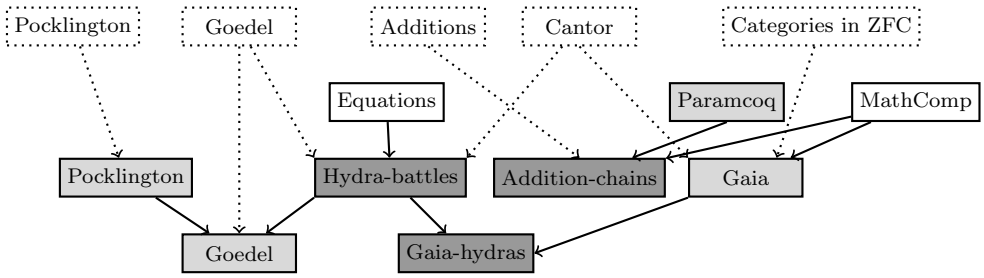

Figure 5: Genealogy and dependencies for Hydras & Co. packages. Dotted boxes represent historical Coq contributions, while regular boxes represent maintained Coq packages. Dark gray packages are maintained in the Hydras & Co. GitHub repository, while light gray packages are maintained in other Coq-community repositories. Dotted lines represent Coq code ancestry, while regular lines represent direct code dependencies.

Before the Ackermann sub-library was moved from the Goedel package repository to the Hydras & Co. repository, the Goedel package had around 7,000 specification LOC and 38,000 proof LOC. Figure 5 and Table 1 thus reveal that the seemingly large Coq formalization of Gödel's incompleteness theorem can be viewed as a modestly-sized proof on top of general libraries for, on the one hand, first-order logic, primitive recursive functions, and Peano Arithmetic (Ackermann), and on the other hand, prime numbers and their properties (Pocklington).

Package name	Repository	Index identifier	Version	Spec LOC	Proof LOC
Pocklington	`pocklington`	`coq-pocklington`	`8.12.0`	825	3,798
Goedel	`goedel`	`coq-goedel`	`8.13.0`	2,799	10,762
Gaia	`gaia`	`coq-gaia`	`1.12`	28,850	124,839
Hydra-battles	`hydra-battles`	`coq-hydra-battles`	`0.5`	14,368	48,385
Addition-chains	`hydra-battles`	`coq-addition-chains`	`0.5`	1,961	2,210
Gaia-hydras	`hydra-battles`	`coq-gaia-hydras`	`0.5`	81	177

Table 1: Current numbers of lines of code for packages in the Hydras & Co. galaxy.

In traditional software development, it is common for packages to continually depend on a large number of packages. In contrast, projects in the Coq ecosystem are often subject to "dependency aversion", where maintainers eschew depending on useful packages because it may lead to work in adapting to upstream changes. With Hydras & Co., we hope to demonstrate that projects with complex dependencies are feasible to manage using recent advances in build management and infrastructure.

4 Modernizing the build infrastructure

4.1 Documentation with Alectryon

The Hydras & Co. book is written in LaTeX, but it makes very frequent references (about once per page, 274 snippets over 281 pages) to parts of the Coq development, showing definitions (Fig. 3), computation results (Fig. 6), lemmas (Fig. 7) and parts of proof scripts (Fig. 8). The order in which these references appear in the book is independent of the structure of our libraries,

so we chose to maintain the book as a standalone document, separate from the Coq source code. This contrasts with the Coqdoc approach, where explanatory prose is embedded within Coq source files (a detailed discussion of different approaches to documenting Coq developments can be found in [41]).

Originally, we copied snippets from Coq sources into LaTeX manually, and recorded and inserted the corresponding outputs manually as well. This approach is common, but brittle: changes to Coq definitions or lemmas had to be reflected in the book's sources, and we found multiple instances where the book and the Coq development had diverged.

We solved this maintenance issue by moving to Alectryon, a tool that automatically records Coq proofs [40]. Instead of copy-pasting fragments into LaTeX, we now embed small LaTeX files automatically generated by Alectryon from our Coq development. Our build system guarantees that these LaTeX snippets are always up-to-date and consistent with the code (and, by comparing these snippets across releases of Coq or versions of Hydras & Co., we can easily spot unexpected changes).

Importing snippets into a LaTeX document was not one of the original use cases of Alectryon. Firstly, the original Alectryon did not have support for exporting to LaTeX. Secondly, it was geared towards documenting individual source files, where code and prose are interleaved within the same file. In this style, the code is documented in the same order as it is compiled. We extended Alectryon to support our needs by implementing a LaTeX backend, and by programming it to generate individual snippet files, one per `snippet` comment block. The later part was straightforward: all it took was to build a custom Alectryon *driver*, a small (about 100 lines) Python program that leverages most of the Alectryon toolchain but defines a custom frontend that understands our snippet annotations (and otherwise exposes the exact same command line as Alectryon).

Once the infrastructure was in place, the transition happened gradually, over a few weeks: for each snippet of Coq code that was in the book, we had to take the following steps:

1. Mark the snippet in the Coq sources (we use special comments `(* begin snippet` *name* `*) ... (* end snippet` *name* `*)`)
2. Configure output display, using special Alectryon annotations to configure what should be shown (only the inputs, inputs and outputs, some steps of the proof but not all, some proof states at key moments in a proof, etc.)
3. Replace the copy-pasted inputs and output in LaTeX with an `input` command.

```
Eval cbv  zeta beta delta [pow_17]  in  pow_17.
```
```
= fun x : A =>
  x * x * (x * x) * (x * x * (x * x)) *
  (x * x * (x * x) * (x * x * (x * x))) * x
: A -> A
```

Figure 6: Automatically capturing the output of computations

Applications in other proof assistants. The Alectryon toolchain is intended to eventually support the languages of many proof assistants, and already has preliminary support for Lean 3. Our Alectryon extensions for snippet generation are in principle applicable to any language

```
Example Ex42: omega + 42 + omega^2 = omega^2.
Proof.
  rewrite <-  Comparable.compare_eq_iff.
  ──────────────────────────────────────
  compare (omega + 42 + phi0 2) (phi0 2) = Eq
  reflexivity.
Qed.
```

Figure 7: A simple proof by computation

supported by Alectryon. However, snippet delimiters are language specific, and our tooling currently assumes Coq-compatible delimiters; we expect to lift this restriction in the near future. We view Hydras & Co. as an incubator for experimental Alectryon features that, when deemed stable and useful, can be disseminated to a wider audience including users of other proof assistants.

4.2 Technologies supporting the monorepo structure

A monorepo (shorthand for monolithic repository) is a version control repository containing multiple independent or related packages. Monorepos have gained increasing popularity following experiences in large companies such as Google, but are also used at a smaller scale for managing open source projects. They are known to simplify the management of dependencies, making cross-packages changes, and refactorings [8].

One of the few early uses of monorepos in the Coq ecosystem was for the Mathematical Components library [32], whose maintainers developed a custom build infrastructure to check multiple packages on each commit. In light of that recent tooling improvements have made monorepo use easier, we chose to use an explicit monorepo structure for Hydras & Co. in Coq-community. Other Coq-community projects have since adopted a similar structure.

In the context of Hydras & Co., we rely on the following tools.

The Dune build system [50]. Dune was originally designed to build OCaml projects, but was recently extended to support Coq. Dune allows building packages contained in a single source tree separately, which is essential to be able to publish the different libraries contained in a single repository as separate (opam) packages to the Coq package index.

However, Dune is currently inconvenient for local IDE-based Coq code development and does not yet support building Coqdoc files. Therefore, we still support building the whole project using `make` via the `coq_makefile` tool bundled with Coq, and we rely on this build system in our documentation generation pipeline. We expect to migrate fully to Dune as soon as these limitations are lifted.

Docker-Coq-Action [34]. This GitHub Action provides a very simple way of setting up Continuous Integration (CI) for a Coq project with an opam file. At the current time, we rely on it, together with the mathcomp Docker images [33], to test our two main libraries, Hydra-battles and Addition-chains, with multiple versions of Coq.

The Coq Nix Toolbox [17]. This toolbox, based on the Nix package manager, provides an alternative way of setting up CI for a Coq project (also relying on GitHub Actions). We use

Let us prove again the equality $\omega + 42 + \omega^2 = \omega^2$. We recall that ω^2 is an abbreviation of $\phi_0(2)$, *i.e* the third additive principal ordinal, and that `F` is a notation for the coercion which maps natural numbers to ordinals.

```
Example Ex42: omega + 42 + omega^2 = omega^2.
```

Our proof is very different from the computational proof of Figure 7. By definition of additive principal ordinals, it suffices to prove the inequality $\omega + 42 < \phi_0(2)$.

```
  assert (HAP:= AP_phi0 2).
  elim  HAP; intros _ H0; apply H0; clear H0.
```

```
HAP: In AP (phi0 (F 2))
──────────────────────────
omega + F 42 < phi0 (F 2)
```

Since the set AP of additive principals is closed under addition (by Lemma *AP_plus_closed*), it suffices to prove the inequalities $\omega < \omega^2$ and $42 < \omega^2$.

```
  Check AP_plus_closed.
```

```
AP_plus_closed
     : forall alpha beta gamma : Ord,
       In AP alpha ->
       beta < alpha ->
       gamma < alpha -> beta + gamma < alpha
```

```
  assert (Hlt: omega < omega^2) by
      (rewrite omega_eqn; apply phi0_mono, finite_mono;
        auto with arith).
```

```
HAP: In AP (phi0 (F 2))
Hlt: omega < phi0 (F 2)
──────────────────────────
omega + F 42 < phi0 (F 2)
```

```
  apply AP_plus_closed; trivial.
```

```
──────────────────────────
F 42 < phi0 (F 2)
```

```
  (* ... *)
```

Figure 8: A proof interleaved with text (from the book)

this Nix-based CI for several things.

1. To test the build of the project (as a single unit) with `coq_makefile` (whereas the Docker-based CI relies on the opam packages, which use Dune to build the libraries contained in the project).

2. To generate the documentation of the project (book in PDF format and Coqdoc HTML pages) by relying on the output of the `coq_makefile` build.

Title	Year	Category
Coq'Art [6]	2004	Traditional
Software Foundations [39]	2007	Executable
Certified Programming with Dependent Types [15]	2011	Executable
Program Logics for Certified Compilers [1]	2014	Traditional
Programs and Proofs [44]	2014	Executable
Formal Reasoning About Programs [16]	2017	Traditional
Computer Arithmetic and Formal Proofs [7]	2017	Traditional
Mathematical Components [32]	2018	Traditional

Table 2: Coq book categorization.

3. To test the bridge to the Gaia library, because the Nix-based CI supports out-of-the-box caching of build dependencies. Given that Gaia takes more than 5 minutes to build, this build is only done once, then reused at each new run of the CI.

4. To test compatibility with the Goedel library, which was made to depend on the Hydra-battles library since the Ackermann sub-library was moved from the former to the latter. For this, we rely on the fact that the Coq Nix Toolbox has native support for generating a CI configuration that includes reverse dependency compatibility testing.

The artifacts of the Nix builds are stored in the Coq-community binary cache on Cachix [31]. This means that a given version never has to be built twice and Continuous Integration can be almost instantaneous when no change has been made (e.g., after merging a pull request).

By relying on two different technologies for CI, we are able to fit a larger range of use cases, while also providing more assurance that the project does build correctly in a variety of configurations. Given that the two technologies are well maintained (within Coq-community), relying on both does not incur a significant cost, compared to the benefits they provide.

5 Comparison of Hydras & Co. with other Coq books

We believe that Coq books can broadly be divided into two categories:

- *Executable textbooks* that are in effect large, well-commented Coq programs from which other representations of the material (HTML, PDF) are derived using tools.

- *Traditional textbooks* generated from documentation languages such as LaTeX that are accompanied by standalone Coq formalizations and/or code snippets.

Table 2 shows a category breakdown for what we believe are the most prominent Coq books in English. Note that several traditional books, such as Coq'Art, used custom tools during their writing process to keep code snippets synchronized with Coq. However, once a book is published, the accompanying code generally takes on a life of its own [5] and may come to substantially diverge from the book.

We believe the Hydras & Co. electronic book has found an attractive set of tradeoffs between the properties of executable books and traditional books. Specifically, in an executable book, the proof assistant language places limits on the structure. For example, Coq may not permit repeating a definition from another file verbatim, since the definition uses variables that are not present in the current context. Thanks to our Alectryon based toolchain, the Hydras & Co. book can include "live" code fragments at any point in the text. And in contrast to traditional books, we validate the consistency of code fragments during every build of the electronic book.

On the one hand, we follow executable books in providing continuous delivery of new revisions at a high pace. But on the other hand, we also aim to make regular timestamped versions compatible with specific versions of Coq, akin to new editions of traditional books.[1]

The flip side of our approach is that we separate the Coq sources from the LaTeX sources that describe the code, and our use of snippet delimiters in the Coq source files may lower source readability. For books and libraries with small scope and specialized audiences, distributing knowledge representations between sources and documentation formats the way we do in Hydras & Co. may introduce overhead without providing complementary benefits. More specifically, we currently distribute the book as a PDF generated by LaTeX that links to corresponding HTML documentation of the Coq sources generated by Coqdoc. In contrast, executable books such as Software Foundations can provide a single uniform HTML representation of both book contents and code. We hope that future toolchain improvements can address all these issues.

6 Conclusion and perspectives

Hydras & Co. wants to provide a connection between scientific literature (e.g., [29, 28, 43]) and proof assistant technology. For the mathematician, it can give a concrete view of the mathematical content, not only through full proofs, but also through illuminating computations: examples, functions associated with constructive proofs, etc. For the Coq learner, it provides a consistent set of examples, allowing to present and compare various formalization and proving techniques. It is also a medium sized library (more than 50,000 LOC), dependent on various tools and libraries of the Coq ecosystem, which may be also used to experiment with new maintenance techniques of the code and its documentation. To facilitate easy dissemination and reuse, all code and documentation is available under the permissive MIT license.

We plan to extend Hydras & Co. in two main directions. Firstly, we plan to bridge the combinatorial results of Hydras & Co. and the set-theoretic content of Gaia, enabling the transfer of many interesting theorems between the two packages. Secondly, we aim to write a formal proof in Coq of the original statement of Theorem 2, using O'Connor's formalization of Peano Arithmetic [37]. Moreover, Hydras & Co. is not limited to the study of ordinal numbers and their applications. We are also developing a package about efficient exponentiation algorithms, and aim to eventually include new topics. We invite new collaborators to join us in our efforts.

Acknowledgments

We thank the anonymous reviewers for their comments, and we are grateful to the original authors and current maintainers of the Coq packages we use and depend on: José Grimm (Gaia), Russell O'Connor (Goedel), Matthieu Sozeau (Equations), the Mathematical Components team, Marc Lasson and Chantal Keller (Paramcoq), and the authors and maintainers of Coq and its associated tools.

References

[1] Andrew W. Appel. *Program Logics for Certified Compilers*. Cambridge University Press, Cambridge, United Kingdom, 2014. https://www.cs.princeton.edu/~appel/papers/plcc.pdf.

[1] The latest Hydras & Co. release is 0.5, available at https://github.com/coq-community/hydra-battles/releases/tag/v0.5.

[2] Jeremy Avigad, Leonardo de Moura, Soonho Kong, and Sebastian Ullrich. Theorem proving in Lean 4, 2021. https://leanprover.github.io/theorem_proving_in_lean4/.

[3] Andrej Bauer. The hydra game. http://math.andrej.com/2008/02/02/the-hydra-game.

[4] Andrej Bauer. The hydra game source code. https://github.com/andrejbauer/hydra, 2008.

[5] Yves Bertot and Pierre Castéran. Coq'Art examples and exercises. https://github.com/coq-community/coq-art.

[6] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Springer, Berlin, Heidelberg, 2004. https://www.labri.fr/perso/casteran/CoqArt/.

[7] Sylvie Boldo and Guillaume Melquiond. *Computer Arithmetic and Formal Proofs*. ISTE Press - Elsevier, 2017.

[8] Gleison Brito, Ricardo Terra, and Marco Tulio Valente. Monorepos: a multivocal literature review. *CoRR*, 2018. https://arxiv.org/abs/1810.09477.

[9] Pierre Castéran. Additions. User Contributions to the Coq Proof Assistant, 2004. https://github.com/coq-contribs/additions.

[10] Pierre Castéran. Utilisation en Coq de l'opérateur de description. In *Actes des Journées Francophones des Langages Applicatifs*, pages 30–44, 2007. http://jfla.inria.fr/2007/actes/PDF/03_casteran.pdf.

[11] Pierre Castéran. Hydra ludica, une preuve d'impossibilité de prouver simplement. In Sylvie Boldo and Nicolas Magaud, editors, *Journées Francophones des Langages Applicatifs*, pages 91–104, 2018. https://hal.inria.fr/hal-01707376/document.

[12] Pierre Castéran and Évelyne Contejean. On ordinal notations. User Contributions to the Coq Proof Assistant, 2006. https://github.com/coq-contribs/cantor.

[13] Pierre Castéran. Hydras & Co. https://github.com/coq-community/hydra-battles/releases/tag/v0.5, 2021.

[14] The Coq community project. https://github.com/coq-community/.

[15] Adam Chlipala. *Certified Programming with Dependent Types*. MIT Press, 2011. http://adam.chlipala.net/cpdt/.

[16] Adam Chlipala. Formal reasoning about programs, 2017. http://adam.chlipala.net/frap/.

[17] Cyril Cohen and Théo Zimmermann. A Nix toolbox for reproducible Coq environments, Continuous Integration and artifact reuse. The Coq Workshop, July 2021.

[18] Coq Development Team. Coq opam package index. https://coq.inria.fr/opam/www/.

[19] Coq Development Team. The Coq Proof Assistant. https://coq.inria.fr.

[20] José Grimm. Implementation of Bourbaki's Elements of Mathematics in Coq: Part one, theory of sets. Research Report RR-6999, INRIA, 2009. https://hal.inria.fr/inria-00408143.

[21] José Grimm. Implementation of Bourbaki's Elements of Mathematics in Coq: Part two; ordered sets, cardinals, integers. Research Report RR-7150, INRIA, 2009. https://hal.inria.fr/inria-00440786.

[22] José Grimm. Implementation of three types of ordinals in Coq. Research Report RR-8407, INRIA, 2013. https://hal.inria.fr/hal-00911710.

[23] José Grimm. Fibonacci numbers and the Stern-Brocot tree in Coq. Research Report RR-8654, INRIA, 2014. https://hal.inria.fr/hal-01093589.

[24] José Grimm. Implementation of Bourbaki's Elements of Mathematics in Coq: Part three structures. Research Report RR-8997, INRIA, 2016. https://hal.inria.fr/hal-01412037.

[25] José Grimm, Alban Quadrat, and Carlos Simpson. Gaia. https://github.com/coq-community/gaia. A Coq-community project.

[26] Hydra battles. https://github.com/coq-community/hydra-battles. A Coq-community project.

[27] Chantal Keller and Marc Lasson. Parametricity in an Impredicative Sort. In *Computer Science*

Logic, volume 16, pages 381–395. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012. http://drops.dagstuhl.de/opus/volltexte/2012/3685.

[28] Jussi Ketonen and Robert Solovay. Rapidly growing Ramsey functions. *Annals of Mathematics*, 113(2):267–314, 1981. http://www.jstor.org/stable/2006985.

[29] Laurie Kirby and Jeff Paris. Accessible independence results for Peano arithmetic. *Bulletin of the London Mathematical Society*, 14(4):285–293, 1982. https://faculty.baruch.cuny.edu/lkirby/accessible_independence_results.pdf.

[30] Donald E. Knuth. Literate Programming. *The Computer Journal*, 27(2):97–111, 01 1984.

[31] Domen Kožar. Cachix, 2018–2021. https://cachix.org.

[32] Assia Mahboubi and Enrico Tassi. Mathematical Components. https://doi.org/10.5281/zenodo.3999478, 2018. With contributions by Yves Bertot and Georges Gonthier.

[33] Erik Martin-Dorel. Docker images of mathcomp, 2018–2021. https://github.com/math-comp/docker-mathcomp.

[34] Erik Martin-Dorel. A gentle introduction to container-based CI for Coq projects. The Coq Workshop, July 2020.

[35] The mathlib Community. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, pages 367–381, New York, NY, USA, 2020. Association for Computing Machinery.

[36] Pengyu Nie, Karl Palmskog, Junyi Jessy Li, and Milos Gligoric. Deep generation of Coq lemma names using elaborated terms. In *International Joint Conference on Automated Reasoning*, pages 97–118, 7 2020. https://arxiv.org/abs/2004.07761.

[37] Russel O'Connor. Goedel. https://github.com/coq-community/goedel. A Coq-community project.

[38] Russell O'Connor. Essential incompleteness of arithmetic verified by Coq. In *International Conference on Theorem Proving in Higher Order Logics*, pages 245–260, Berlin, Heidelberg, 2005. Springer. https://arxiv.org/abs/cs/0505034.

[39] Benjamin Pierce et al. Software Foundations. https://softwarefoundations.cis.upenn.edu.

[40] Clément Pit-Claudel. Alectryon. https://github.com/cpitclaudel/alectryon.

[41] Clément Pit-Claudel. Untangling mechanized proofs. In *International Conference on Software Language Engineering*, pages 155–174, New York, NY, USA, 2020. Association for Computing Machinery. https://dl.acm.org/doi/pdf/10.1145/3426425.3426940.

[42] Hans Jürgen Prömel. Rapidly growing Ramsey functions. In *Ramsey Theory for Discrete Structures*, pages 97–103. Springer, Cham, 2013. https://link.springer.com/chapter/10.1007/978-3-319-01315-2_8.

[43] Kurt Schütte. *Proof Theory*. Springer, 1977. https://link.springer.com/book/10.1007%2F978-3-642-66473-1.

[44] Ilya Sergey. Programs and Proofs: Mechanizing Mathematics with Dependent Types, 2014. https://doi.org/10.5281/zenodo.4996238.

[45] Carlos Simpson. Category theory in ZFC. User Contributions to the Coq Proof Assistant, 2004. https://github.com/coq-contribs/cats-in-zfc.

[46] Carlos Simpson. Set-theoretical mathematics in Coq, 2004. https://arxiv.org/abs/math/0402336.

[47] Matthieu Sozeau and Cyprien Mangin. Equations reloaded: High-level dependently-typed functional programming and proving in Coq. *Proceedings of the ACM on Programming Languages*, 3(ICFP), July 2019. https://hal.inria.fr/hal-01671777.

[48] Matthieu Sozeau and Nicolas Oury. First-class type classes. In *International Conference on Theorem Proving in Higher Order Logics*, pages 278–293, Berlin, Heidelberg, 2008. Springer. https://sozeau.gitlabpages.inria.fr/www/research/publications/First-Class_Type_Classes.pdf.

[49] Bas Spitters and Eelis van der Weegen. Type classes for mathematics in type theory. *Mathematical*

Structures in Computer Science, 21(4):795–825, 2011. https://arxiv.org/abs/1102.1323.

[50] The Dune authors. Dune: A composable build system for OCaml, 2016–2021. https://dune.build.

[51] Floris van Doorn, Gabriel Ebner, and Robert Y. Lewis. Maintaining a library of formal mathematics. In Christoph Benzmüller and Bruce Miller, editors, *Intelligent Computer Mathematics*, pages 251–267, Cham, 2020. Springer International Publishing.

[52] Stan Wainer. A classification of the ordinal recursive functions. *Archiv für mathematische Logik und Grundlagenforschung*, 13(3):136–153, Dec 1970. https://link.springer.com/article/10.1007%2FBF01973619.

[53] Stan Wainer and Wilfried Buchholz. Provably computable functions and the fast growing hierarchy. In Stephen G. Simpson, editor, *Contemporary Mathematics*, volume 65, pages 179–198. American Mathematical Society, Providence, RI, USA, 1987. http://nbn-resolving.de/urn/resolver.pl?urn=nbn:de:bvb:19-epub-3843-7.

[54] Théo Zimmermann. *Challenges in the collaborative evolution of a proof language and its ecosystem*. PhD thesis, Université de Paris, 2019. https://hal.inria.fr/tel-02451322.

[55] Théo Zimmermann and Jean-Rémy Falleri. A grounded theory of community package maintenance organizations-registered report. In *ICSME 2021-37th International Conference on Software Maintenance and Evolution*, 2021.