A Nix toolbox for reproducible Coq environments, Continuous Integration and artifact reuse

Cyril Cohen, Théo Zimmermann

A Nix toolbox for reproducible Coq environments, Continuous Integration and artifact reuse

Cyril Cohen[1] and Théo Zimmermann[2]

[1] Université Côte d'Azur, Inria, France
[2] Université de Paris, Inria, France

When using or contributing to a Coq project (especially one with third party dependencies), it can be difficult and annoying to install the exact set of dependencies at the versions the maintainers recommend. It is more convenient when projects provide an easy way to set everything up to be ready to code, even more so if the dependencies can be quickly fetched in pre-compiled form. Nonetheless, it is also important to preserve the flexibility for project maintainers to test their projects against alternative versions (releases and development) of the dependencies, both while developing locally and in a distant Continuous Integration service (CI). Finally, being able to test that candidate changes do not break reverse dependencies (i.e., packages that depend on the project) is essential to bring confidence to project maintainers.

We have designed the Coq Nix toolbox which relies on Nix [3] to allow for better sharing of configurations and pre-compiled packages across several projects. This toolbox makes it easy to setup a project and generates CI configurations to test multiple versions of the dependencies and compatibility with reverse dependencies. It enables the use of a single command `nix-shell` to get the same working environment for every developer. This also saves compilation time since developers and users may download packages that have already been compiled in CI.

1 A short introduction to Nix and Nix Coq packages

Nix is a purely functional language and a software suite geared towards package management. Although this package manager is at the basis of the NixOS distribution, it can be used on any Linux distribution, on macOS, and even on Windows thanks to WSL.

Derivations in Nix. The main container types in Nix are *attribute sets*, which are simply finite maps from strings to any object. A *derivation* is a distinguished attribute set containing data and instructions on how to produce some output (often binaries, libraries or documentation), which is called the *realization* of the derivation. NixPkgs [5] is one of the largest known package repositories: it gathers everything to make a Linux distribution, but also thousands of packages useful for developers in many programming languages. It is implemented as a big Nix program which produces an attribute set of (attribute sets of ...) derivations.

While Nix allows installing multiple package versions, unlike Docker, the files, binaries, etc. are accessible directly from the root filesystem, there is no notion of container or image, and often no need to build manually, since a single call to `nix-shell` in an appropriate context will handle everything automatically.

Generating and overriding a Coq library derivation. We build on the work of many Nix users, who created and maintained an attribute set called `coqPackages` in NixPkgs. This attribute set contains many Coq third party library derivations (and includes Coq itself). It is at the center of our toolbox.

We redesigned `coqPackages` to streamline the creation of derivations of Coq third party libraries and to make sure it was easy to alter.[1] Indeed, we now provide a function `mkCoqDerivation` which builds a derivation out of a minimal input, defaulting to some standard practices in the Coq ecosystem (e.g. dependency on Coq, using Dune or GnuMake, etc). We also included a lightweight overriding mechanism which reuses the metadata from the derivation (URL or version control system, owner, repository, etc.) to alter source fetching using a single argument (usually a string denoting a version number, a branch or a pull request). Contrarily to Opam packages, there is no version resolution mechanism, a given derivation specifies exact versions for all its dependencies, and overriding allows testing alternative versions.

[1] We thank Vincent Laporte for providing feedback and reviewing this refactoring.

2 The Coq Nix toolbox

We describe here our design of the Coq Nix toolbox [1]. It is a Nix program, which, given a project `A`, reads a set of configuration files with a predetermined location from the root `A-root` of the project (`A-root/.nix/config.nix`, `A-root/.nix/*overlays/`, etc), produces or alters several derivations from `coqPackages` according to the configuration files, and provides various commandline utilities.

After setup (cf. "Initialization and update"), running the command `nix-shell` provides a shell where all the dependencies of your project are available, along with the toolbox commandline utilities.

Producing the project derivation. The main feature of the toolbox is to modify or create (if necessary) the derivation of `A` to use the local directory `A-root` as a source. All existing reverse dependencies of `A` inside the local instance of NixPkgs will then refer to the development version (from `A-root`).

Lightweight modification of dependencies and reverse dependencies using bundles. Changing the version of a related package—e.g. a direct or reverse dependency—can usually be done in the configuration files by modifying a single field in `A-root/.nix/config.nix`. The file organizes packages by *bundles*. A bundle is an attribute set where each field describes a set of package versions that must be compatible with each other. The user can then call `nix-shell --argstr bundle name` to select which bundle to use.

Ready-to-publish NixPkgs overlays. Sometimes, giving a version number is not enough. Indeed, if compilation instructions change in an unforeseen way, it may be necessary to change the Nix code that creates a derivation in NixPkgs. We provide the directory `A-root/.nix/*overlays/` so that users can write their own derivations locally, in the exact form that should be used for later publication to NixPkgs. This is the recommended method if your package is not yet in NixPkgs upon initialization.

Configuring CI for the project and reverse dependencies. From the information contained in NixPkgs and each bundle declared in the configuration file, the commandline tool `genNixActions` computes the dependency graph of packages within `coqPackages`, filters out unrelated packages, and turns it into a GitHub Action workflow. This way we get CI tests for the specific set of versions each bundle declares, and in particular the project package is tested against each bundle of dependencies.

Caching and artifact reuse The CI may optionally refer to one or several Cachix [4] repositories to store and pull compiled binaries and `.vo` files. If enabled, it means a derivation is never realized twice. And if users declare the appropriate caches locally, they can reuse (i.e., download) transparently any of the realizations compiled by the CI, without recompiling it. We also provide an experimental `cachedMake` commandline utility which attempts to download precompiled `.vo` to speed up local incremental builds.

Initialization and update The toolbox can be run directly from `https://coq.inria.fr/nix/toolbox` (`nix-shell` can run Nix files stored in arbitrary locations). One can then use the commands provided by the toolbox to generate the configuration files, update the toolbox version as well as the NixPkgs version. We also plan to make it possible to setup the toolbox through Coq-Community Templates [2].

References

[1] Cyril Cohen and Théo Zimmermann. Coq Nix Toolbox. https://github.com/coq-community/coq-nix-toolbox.

[2] The Coq Community. Coq-Community Templates, 2018–2021. https://github.com/coq-community/templates.

[3] Eelco Dolstra and Andres Löh. NixOS: a purely functional Linux distribution. In James Hook and Peter Thiemann, editors, *Proceeding of the 13th ACM SIGPLAN international conference on Functional programming, ICFP 2008, Victoria, BC, Canada, September 20-28, 2008*, pages 367–378. ACM, 2008.

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

[5] The NixOS maintainers. Nix Package collection, 2003–2021. https://github.com/nixos/nixpkgs.