A Case for Multi-Switch Constraints in OPAM

Fabrice Le Fessant
INRIA & OCamlPro
`fabrice.le_fessant@inria.fr`

Abstract

Package managers usually only deal with packages and their versions, and the constraints on their dependencies towards other packages' versions. Among package managers, `opam` is probably the first one to introduce the notion of *switch*, i.e. the ability to manage different directories, where different sets of packages with different versions are installed, as each directory is treated as an independant universe when solving dependency constraints. In this talk, we will support a case to be able, in `opam`, to manage different switches in the same universe, allowing to express dependency constraints that cross switch boundaries.

1 Introduction

Package managers usually only deal with packages and their versions, and the constraints on their dependencies towards other packages' versions. Among package managers, `opam` is probably the first one to introduce the notion of *switch*, i.e. the ability to manage different directories, where different sets of packages with different versions are installed, as each directory is treated as an independant universe when solving dependency constraints. These switches have been introduced in `opam` because packages compiled with different versions of OCaml cannot be used together. Indeed, most OCaml users need to use several versions of OCaml at the same time, either because they want to be able to compile their software on these different OCaml versions, or because they want to try some specific features, either available as extensions of former OCaml versions, or only available in the most recent official versions.

`opam` has been designed to make it easy for users to switch between OCaml versions, and additionnal tools (such as `ocp-manager`) can be used for even more flexibility. Until now, however, switches are treated as completely different universes, when examining available packages and solving dependency constraints for packages, i.e. packages in a switch cannot depend on the presence of another package in a different switch.

We think it is a good time for `opam` to remove this limitation, and to allow package developers to write dependency constraints of a package in a switch on the packages that are installed in other switches. With such constraints, any `opam` operation could trigger installation or removal of packages in several switches. In the next sections, we show how such a mechanism could be easily implemented, and several applications, that such a change would make possible.

2 Implementation

We propose to extend the language to describe the dependency constraints of a package (the `depends` field of the `opam` file) with *switch prefixes* on package names, such as `switch-prefix:package-name`. A switch prefix is a symbolic name, resolved independantly inside every switch to an existing switch value. By default, all packages are prefixed with the switch prefix `host`, resolved to the current switch. Switch descriptions (compilers) can specify the switch prefixes that can be used within them, in which case the value of these switch prefixes are either also specified in the compiler description, or has to be specified by the user. If a switch prefix value is finally not defined, or does not exist, it defaults to the `missing` switch, and the prefixed package will not be available.

When calling its CUDF solver, `opam` will generate a universe (the list of all available packages with their dependencies) that will contain the transitive closure of the current switch, and all the switches that are reachable from it, through inter-switch package dependencies. Conversely, `opam` will interprete the solver reply by removing and installing the packages in the different switches. We think the modifications required to support switch prefixes on packages are conceptually minimal, and will be easy to implement in the current version of `opam`.

Of course, to be useful, variables in `opam` files would also have to allow prefixing, so that build commands can use the knowledge of how prefix switches have been resolved in the switch where the package is being installed. Finally, inter-switch dependencies would of course be read-only, in the sense that packages would be allowed to read and execute files from their dependencies in other switches, but would not be allowed to modify the content of these other switches during their installation.

3 Applications

We think the addition of inter-switch dependencies is an interesting idea, both theoretically (it provides a clear semantics for `opam` switches) and practically, as several useful applications can be implemented on top of it. In this section, we provide several examples.

3.1 Cross-compilation of OCaml packages

Inter-switch dependencies could be use to provide a simple environment for cross-compilation of OCaml packages. Indeed, a difficulty of cross-compiling OCaml is that a lot of packages depend on code-generation tools and preprocessors, to be built and executed for the `build` platform, and libraries, to be built for the `host` platform (the target).

Using inter-switch dependencies, dependencies to packages containing code-generation tools and preprocessors would be prefixed with a `build` switch prefix, whereas dependencies to libraries would be prefixed with the default `host` switch prefix. In a switch without cross-compilation, the `build` switch prefix would resolved to the current switch, whereas in a cross-compilation switch (for example, `4.02.0-linux-mingw64`) the `build` switch prefix would resolve to the local switch without cross-compilation (`4.02.0` for example).

As a theoretical example, installing a cross-compilable `sexplib` in a cross-compilation switch would trigger the compilation of `sexplib-runtime` in the cross-compilation switch, and `camlp4`, `sexplib-runtime` and `sexplib-syntax` in the local switch.

3.2 Management of External Dependencies

One of the current problems of `opam` is the impossibility to include external dependencies (dependencies to system libraries, for example), either to decide which packages can be installed, or to trigger the installation of these external dependencies.

Using inter-switch dependencies, we propose to create a specific switch, called `system`, containing packages corresponding to external dependencies. On different platforms, this switch would be provided by different repositories than the rest of OCaml `opam` packages. For example, there would be a repository for Debian/Ubuntu external dependency. In such a repository, each switch description would correspond to a different system (a specific Linux distribution), and the corresponding packages would call, when installed, the system specific command to install the system specific package corresponding to the external dependency.

3.3 Per-switch Repositories

In the current version of `opam`, it is not possible to add a remote repository to a specific switch. Repositories have to be shared by all switches, an unfortunate limitation when, for example, you want to beta-test the packages provided by a specific repository, without compromising all your switches.

Using switch prefixes, per-switch repositories would be easy to implement: when reading the packages of a per-switch repository, all package names can be prefixed with the switch name in the global universe transmitted to the CUDF solver.

3.4 Multi-Switch Packages

For some packages, being independant from a particular switch could be very useful. For example, a package providing only the `ocp-indent` command could install the tool in a directory shared by all switches, as the command does not depend on anything in the switch at runtime. Currently, installing such a package is not optimal, as installing the package in the current switch could trigger the compilation of many dependencies, which have already been installed in another switch. A solution to benefit from the capacity of the CUDF solver to minimize the number of operations would be to send to the solver a universe containing all the packages in all the current switches, and requesting the installation of a meta-package, depending on the disjunction of the package prefixed by every existing switch. Then, the solver would reply with the shortest list of operations to install the package in only one of the switches, which is enough to provide the tool to all the switches.

3.5 All-switches Commands

It is sometimes useful to provide commands that will work on all the existing switches. An example would be `opam upgrade-all`, which would trigger the upgrade of all the switches at once. Another example would be `opam install-all some-package`, for a package that is often used in all developments of a particular user. Compared to calling the same command on all switches, the main interest would be to enable parallel installation of all the packages in the different switches.

3.6 Application-Specific and Language-Specific Switches

The `Coq` community has started to work on how to use `opam` as a package manager for Coq contributions. A specific problem is that a user might want to install different versions of Coq in different switches, but sharing the same version of OCaml. Another example would be to use `opam` to package contributions for other languages than OCaml. A user might then want to install a package that is implemented in different languages, OCaml, Python and Javascript for example.

In both these examples, the problem could be solved by using inter-switch dependencies. In the Coq example, Coq packages would depend on a generic `ocaml` switch prefix, that is instantiated by the particular switch (corresponding to a particular OCaml version) with which Coq is built. Moreover, listing the packages available in the Coq switch would only show the Coq packages, as OCaml packages on which they depend would be in another switch.

4 Discussion

We propose to present the addition of inter-switch dependencies in this talk, and discuss some of the new possibilities that would be offered by such a feature in `opam`. We think it is easy to implement, without changing much the current architecture of `opam`, and that, once present, many new applications would appear, that have not even been forseen in this paper.