Porting the Mathematical Components library to Hierarchy Builder

Reynald Affeldt[3], Xavier Allamigeon[4], Yves Bertot[1], Quentin Canu[4], Cyril Cohen[1], Pierre Roux[6], Kazuhiko Sakaguchi[2], Enrico Tassi[1], Laurent Théry[1], and Anton Trunov[5]

[1] Université Côte d'Azur, Inria, France
[2] University of Tsukuba, Japan
[3] National Institute of Advanced Industrial Science and Technology (AIST), Japan
[4] Inria, CMAP, CNRS, Ecole Polytechnique, Institut Polytechnique de Paris, France
[5] Zilliqa Research
[6] ONERA / DTIS, Université de Toulouse, France

We report on the porting of the Mathematical Components library (hereafter MC) to the Hierarchy Builder [2] tool (hereafter HB).

MC is an extensive and coherent repository of formalized mathematical theories. At the time of writing about 40 opam packages depend on some components from this library and these packages are not exclusively focusing on mathematics (e.g., QuickChick, Disel, ...). To our knowledge about 100 papers about formal proofs build on top of MC. The library is made of 91 files for a total 124 000 lines of code.

The key to keep the library growing in a rational way is that it revolves around a hierarchy of interfaces which organizes operations and properties. Interfaces come with theories which apply, automatically, to all the objects which are registered as validating the interface. These interfaces are implemented following the packed classes discipline which works well in practice but has never been easy to master. Pull Requests extending or modifying the hierarchy turned out to be problematic, hard to review and integrate due to their length and complexity. Only a few were merged, the others were put on hold waiting for the library to be ported to HB.

HB is a set of high level vernacular commands implemented in Coq-Elpi, an extension language for Coq developed by Tassi. HB takes care of automatically synthesizing all the error prone boilerplate required for packed classes to work. Users just input record declarations, standing for the interfaces, and proofs, for the instances. They are relieved from most of the gibberish of modules, sections, coercions, canonical structures, implicit arguments, phantom abbreviations, ...

In the week from the 6th to the 9th of April 2021 the authors of this abstract gathered (virtually) for a sprint on porting the MC library to HB. The whole library finally compiled on April 20. The porting consisted of declaring all the interfaces and their instances using the new commands, *without changing proofs* in any substantial way. At the time of writing the patch (math-comp#733) amounts to about 5K lines being edited (the declaration of the hierarchy and the declarations of canonical instances), and 5K lines being simply *removed*.

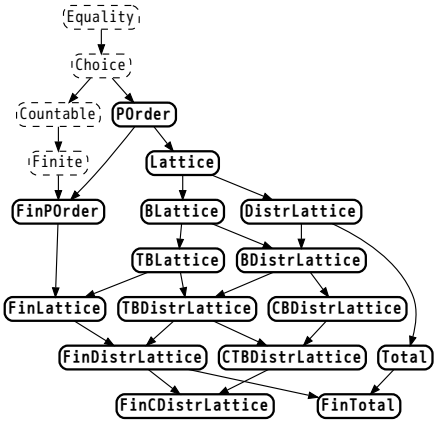

Figure 1: Hierarchy in `order.v`

Documentation HB is a young tool lacking documentation on how to use it proficiently. Some of this knowledge emerged during the sprint and is being archived in the wiki of the project and we plan to share it with the Coq community at the workshop.

The documentation of the MC library itself is also being reworked, updating the file headers with the new interfaces the user is supposed to use. In order to ease the documentation effort we extended HB with two commands, one to draw the hierarchy of interfaces, see Figure 1 for the excerpt for the order component, and another one similar to `About` to access metadata specific to the hierarchy, like the known instances of an interface and where they are declared.

Performance When definitions pile up, unfolding them can lead to terms which are too large to handle even for a computer, and Type Theory provides no lightweight way to enforce abstraction barriers. In order to prevent performance issues the MC library "locks" some definitions, hiding them behind module signatures. Since HB uses a less "efficient" representation of structures (more regular, not hand-optimized) we had to lock a handful more definitions. Given that the MC library was already built with abstraction barriers in mind – once a new concept is defined and all its theory is proved, the client is never supposed to unfold the concept – the changes needed to adapt proofs (unlocking concepts) were minimal. We extended HB with a lock command which takes care of the rather verbose declarations of modules and their signatures.

Lightweight factories HB provides a built-in concept of *factory*, a virtual interface which is compiled to real interfaces automatically. However, sometimes, defining a factory is unnecessarily heavy and we prefer to rely on either one or a combination of the following techniques: we use a lemma which produces a factory from some (smaller) input; we declare a type alias (identity definition) which already validates some interfaces and we copy all its instances to a new type via an automatically generated tool. For example `Countable.copy T (can_type fgK)` equips a type `T` with `Countable`, `Choice` and `Equality` instances proviso a proof `(fgK : cancel f g)` where `cancel f g` expresses the fact that `(f : T -> T')` and `g` are inverse functions, and where `T'` is already a `Countable` type.

Limitations MC defined a normed $\mathbb{Z}$-module as a $\mathbb{Z}$-module equipped with a norm with values in a numeric domain, and a numeric domain as a $\mathbb{Z}$-module on itself; this apparent cyclicity which was encodable by hand [1] is not supported by HB. Since these structures are only really needed in MC-analysis, we managed to port MC by oversimplifying them. We plan to extend HB to support this kind of self-reference thus enabling the porting of MC-analysis. There are also hierarchies of morphisms and subobjects in MC, while we could have ported the former, the latter follows a design pattern that is not yet supported by HB.

The coding sprint One of the challenges of the sprint was to contribute simultaneously to several MC files and eventually update HB with fixes for blocking bugs. We used a locking mechanism based on Github's check list on the pull request, whose advancement status was also reflected on the file graph of the library. Moreover we systematically used a Zulip stream and a Jitsi video chat per file, allowing the developers to work in small groups, share their knowledge and progress and quickly reach out for help. Using Nix to describe the development environment helped keeping the continuous integration in sync with the environment used by the developers. Still, we could have benefited from more tool support. In particular, an UI that would allow executing in parallel two versions of a Coq script would have been a plus in this porting activity.

Impact on MC users and perspectives The switch to HB will lead to MC version 2.0. Updating to it will cause breakage but we believe the benefits outweigh the porting burden, if only because we will then be able to rework the hierarchy *without breaking client code*, since factories can serve as backward compatible interfaces. Moreover it will trigger a number of immediate improvements to the library from the addition of new, long awaited, structures like semilattices and semirings to the integration of existing user developments (e.g., dioid). Still, since MC is widely used, we plan to continue maintaining the 1.x branch even after version 2.0 is released (possibly backporting updates which don't require changing the hierarchy).

References

[1] Reynald Affeldt, Cyril Cohen, Marie Kerjean, Assia Mahboubi, Damien Rouhling, and Kazuhiko Sakaguchi. Competing inheritance paths in dependent type theory: a case study in functional analysis. In *IJCAR 2020 - International Joint Conference on Automated Reasoning*, pages 1–19, Paris, France, June 2020.

[2] Cyril Cohen, Kazuhiko Sakaguchi, and Enrico Tassi. Hierarchy Builder: Algebraic hierarchies Made Easy in Coq with Elpi (System Description). In *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, volume 167 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:21, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.