Porting Coq Scripts to the Mathematical Components Library Version 2

May 9, 2023

The goal of this document is to explain how to port developments using MathComp to MathComp 2.

The Mathematical Components library (hereafter, MathComp) provides a number of mathematical structures organized as hierarchies. Hierarchy Builder (hereafter, HB) is an extension of the Coq proof assistant to ease the development of hierarchies of structures [4]. MathComp 2 is the result of the port of MathComp to HB [1].

For the sake of concreteness, we illustrate the port of CompDecModal [5] in Sect. 3. Before that we review basic usage of HB in Sect. 1 and review the documentation tools available for porting in Sect. 2.

Contents

1 Quick Reminder About HB

The goal of this section is to briefly explain the three main commands introduced by HB: `HB.mixin`, `HB.structure`, and `HB.instance`. The knowledgeable reader can safely skip this section.

Let us explain in a generic way the most basic scenario. Here is the pattern to declare a structure `Struct` that sits sit at the bottom of a hierarchy. The interface of the structure goes into a mixin:

```
HB.mixin Record isStruct params carrier := {
  ... properties about the carrier ...
}
```

The structure itself is declared like a sigma-type:

```
#[short(type=structType)]
HB.structure Definition Struct := {carrier of isStruct carrier}
```

HB is using COQ attributes to declare the type corresponding to a structure.

Here is the pattern to declare a new structure `NewStruct` that extends the existing structure `Struct`; note the `of` syntax.

```
HB.mixin Record NewStruct_from_Struct params carrier
    of Struct params carrier := {
  ... more properties about the carrier ...
}
```

In the case of the extended structure, the sigma-type makes appear the dependency to the parent structure; note the `&` syntax.

```
#[short(type=newStructType)]
HB.structure NewStruct params :=
  {carrier of NewStruct_from_Struct parames carrier
           & Struct params carrier}.
```

This process results in the creation of the types `structType` and `newStructType` such that elements of the latter are also understood to be elements of the former.

Finally, the declaration of a mixin `Struct` is accompanied by the creation of a constructor `Struct.Build` which is used to instantiate a structure using the command:

```
HB.instance Definition _ := Struct.Build params.
```

The command `HB.instance` should trigger the printing of several lines of information output such as `module_type__canonical__s`
The absence of this output often indicates failure of the `HB.instance` command.

For more information about HB see:

- the original paper for an extensive introduction to HB commands [4],
- the HB development for documentation and examples [7] (start with the README),
- various papers for more applications [1] [2, Sect. 3] [3, Sect. 4].

2 Tools to Port MathComp Applications

2.1 Documentation

The following pieces of documentation are useful during the process of porting a MATHCOMP application to MATHCOMP 2:

- The changelog is the primary source of information. See `CHANGELOG.md`.
- Additionally, structures are documented in the headers of the source code according to the following format:

  ```
  (***************************************************************************)
  (*                            Centered Title                              *)
  (*                                                                         *)
  (* Some introductory text: what is this file about, instructions to use this *)
  (* file, etc.                                                              *)
  (*                                                                         *)
  (* Reference: bib entry if any                                             *)
  (*                                                                         *)
  (* * Section Name                                                          *)
  (*   definition == prose explanation of the definition and its parameters    *)
  (*     notation == prose explanation, scope information should appear nearby *)
  ```

```
(*   structType == name of structures should make clear the corresponding    *)
(*                 HB structure with the following sentence:                 *)
(*                 "The HB class is Xyz."                                    *)
(*     shortcut := a shortcut can be explained with (pseudo-)code instead of *)
(*                 prose                                                     *)
(*                                                                           *)
(* Acknowledgments: people                                                   *)
(*****************************************************************************)
```

See for example the `eqType` structure defined in the file `ssreflect/eqtype.v`. See this wiki entry for more information about the documentation of scripts.

- Optionally, the user can double-check the naming of identifiers and lemmas with the naming conventions explained in `CONTRIBUTING.md`.

2.2 HB Commands Useful to Explore an Existing Hierarchy

Besides the changelog and the headers of source code, the user can use HB commands to explore a hierarchy of mathematical structures.

2.2.1 Information about Structures with `HB.about`

Basic information about structures can be obtained via the command `HB.about` as in:

```
> HB.about eqType.
HB: eqType is a structure (from "./ssreflect/eqtype.v", line 137)
HB: eqType characterizing operations and axioms are:
    - eqP
    - eq_op
HB: eqtype.Equality is a factory for the following mixins:
    - hasDecEq (* new, not from inheritance *)
HB: eqtype.Equality inherits from:
HB: eqtype.Equality is inherited by:
    - SubEquality
    - choice.Choice
...
```

(The output message refers to a *factoriy*: this is a generalization of mixin.)

Graph of an HB Hierarchy It is also possible to explore a HB hierarchy using the command `HB.graph`. Inside a Coq file:

```
HB.graph "hierarchy.dot".
```

From a terminal:

```
tred hierarchy.dot | dot -Tpng > hierarchy.png
```

For example, Fig. 1 displays the immediate vicinity of `eqType`.

2.2.2 Information about Constructors with `HB.howto` and `HB.about`

To discover constructors to build a structure, one can use the command `HB.howto`. For instance

```
> HB.howto eqType.
HB: solutions (use 'HB.about F.Build' to see the arguments of each factory F):
    - hasDecEq
```

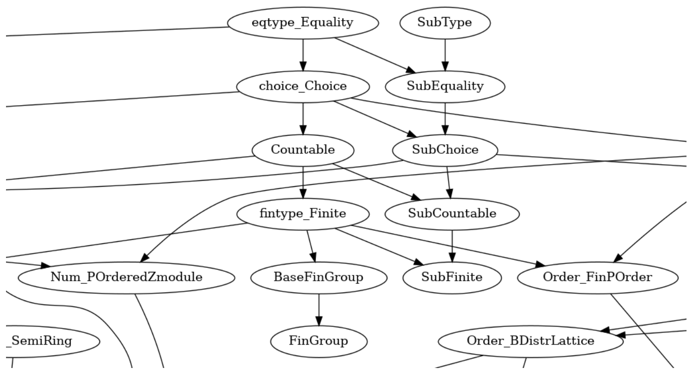

Figure 1: The vicinity of the structure `eqType` in MathComp

tells us that `eqType` instances can be built with `hasDecEq.Build`. (Note that by default `HB.howto` may not return all the available factories; it might be necessary to increase the depth search using a natural number as in `HB.howto xyzType 5`.)

To learn which parameters a `xyz.Build` constructor is expecting, one can use the `HB.about` command:

```
> HB.about hasDecEq.Build.
HB: hasDecEq.Build is a factory constructor
    (from "./ssreflect/eqtype.v", line 135)
HB: hasDecEq.Build requires its subject to be already equipped with:
HB: hasDecEq.Build provides the following mixins:
    - hasDecEq
HB: arguments: hasDecEq.Build T [eq_op] eqP
    - T : Type
    - eq_op : rel T
    - eqP : Equality.axiom eq_op
```

The message indicates that `hasDecEq.Build` is expecting a type `T`, a predicate `eq_op : rel T` (implicit argument, as indicated by the square brackets) and a proof of `Equality.axiom eq_op`. One can thus instantiate an `eqType` on some type `T` with]

```
HB.instance Definition _ := hasDecEq.Build T proof_of_Equality_axiom.
```

or

```
HB.instance Definition _ := @hasDecEq.Build T eq_op proof_of_Equality_axiom.
```

which should output a few lines among which (recall that the absence of this output often indicate an instantiation problem)[1]:

```
module_T__canonical__eqtype_Equality is defined
```

Discover Aliases and feather factories In addition to the structures and constructors listed by `HB.about`, the library defines some aliases (a.k.a. feather factories). These aliases are documented in the header comments. For instance, an `eqType` instance on some type `T` can be derived from some `T'` already equipped with an `eqType` structure, given a function `f : T -> T'` and a proof `injf : injective f`:

```
HB.instance Definition _ := Equality.copy T (inj_type injf).
```

See `eqType.v` for `inj_type`.

[1]Beware that at the time of this writing the output may not be visible by default with vscoq.

2.2.3 Information about instances with HB.about

Instances a type is already equipped with can be listed with `HB.about`, for instance:

```
> HB.about bool.
HB: bool is canonically equipped with structures:
    - Order.BDistrLattice
      Order.BLattice
      Order.BPOrder
      (from "./ssreflect/order.v", line 6064)
...
```

lists all the structures `bool` is already equipped with.

3 Porting a Development using MathComp to MathComp 2

The basic strategy to port an existing MathComp development to MathComp 2 is to (1) install MathComp 2, (2) compile the existing source code, and (3) fix the errors one after the other. For the sake of concreteness, we explain the port of CompDecModal [5]. This is a development with a moderate use of MathComp whose port involves fixing the instantiation of basic structures that most developments using MathComp are likely to use.

3.1 Import the HB Library

First thing first, any Coq file using HB must start with:

```
From HB Require Import structures.
```

3.2 Instantiation of Structures with MathComp 2

From the viewpoint of the MathComp user, the main change is the way mathematical structures are now instantiated. Most `Canonical` (or `Canonical Structure`) commands are replaced by `HB.instance` (see Sect. 1) and there are small changes to MathComp notations such `[subType ...]`, etc.

Regarding CompDecModal, the first offending set of commands is the following (file `fset.v`):

```
Section FinSets.
 Variable T : choiceType.
 ...
 Canonical Structure fset_subType := [subType for elements by fset_type_rect].
 Canonical Structure fset_eqType := EqType _ [eqMixin of fset_type by <:].
 Canonical Structure fset_predType := PredType (fun (X : fset_type) x => nosimpl x \in elements X).
 Canonical Structure fset_choiceType := Eval hnf in ChoiceType _ [choiceMixin of fset_type by <:].
End FinSets.

Canonical Structure fset_countType (T : countType) :=
  Eval hnf in CountType _ [countMixin of fset_type T by <:].
Canonical Structure fset_subCountType (T : countType) :=
  Eval hnf in [subCountType of fset_type T].
```

Let us consider compilation errors in order:

```
> Canonical Structure fset_subType := [subType for elements by fset_type_rect].
Error: Syntax error: [reduce] expected after ':=' (in [def_body]).
```

This error is due to a change of notation that is documented in the changelog. Search for the string, say, "`[subType`" in `CHANGELOG.md`:

```
- in `eqtype.v`
  ...
  + notation `[subType for v by rec]`, use `[isSub for v by rec]`
  ...
```

The fix is therefore the following:

```
> HB.instance Definition _ := [isSub for elements by fset_type_rect].
HB_unnamed_factory_3 is defined
fset_fset_type__canonical__eqtype_SubType is defined
```

Note that the instance need not be named and better not be since it is the job of HB to figure out instances automatically. It is important to check that HB displays more than one message as a response to `HB.instance`, otherwise this might indicate a failed instantiation.

Next compilation error:

```
> Canonical Structure fset_eqType := EqType _ [eqMixin of fset_type by <:].
Error: The reference EqType was not found in the current environment.
```

This error is primarily due to the remove of the `EqType` constructor [6, Sect. 2.1]. In fact, most `xyzType` constructors from MathComp should not be necessary anymore. See the changelog. Similarly to the `[subType for _ by _]` notation above, the `[eqMixin of _ by <:]` has changed:

```
- in `eqtype.v`
  ...
  + notation `[eqMixin of T by <:]`, use `[Equality of T by <:]`
  ...
```

The fix is therefore:

```
> HB.instance Definition _ := [Equality of fset_type by <:].
HB_unnamed_factory_8 is defined
eqtype_Equality__to__eqtype_hasDecEq is defined
HB_unnamed_mixin_10 is defined
fset_fset_type__canonical__eqtype_Equality is defined
fset_fset_type__canonical__eqtype_SubEquality is defined
```

The next two compilation errors are similarly due to the removal of `choiceType` and `CountType`, and to the change of the notations `[choiceMixin of _ by <:]` and `[countMixin of _ by <:]`:

```
> Canonical Structure fset_choiceType := Eval hnf in ChoiceType _ [choiceMixin of fset_type by <:].
Error: The reference ChoiceType was not found in the current environment.
> Canonical Structure fset_countType (T : countType) :=
>   Eval hnf in CountType _ [countMixin of fset_type T by <:].
Error: The reference CountType was not found in the current environment.
```

The fix can again be inferred from the changelog:

```
> HB.instance Definition _ := [Choice of fset_type by <:].
HB_unnamed_factory_11 is defined
choice_Choice__to__choice_hasChoice is defined
HB_unnamed_mixin_14 is defined
fset_fset_type__canonical__choice_Choice is defined
fset_fset_type__canonical__choice_SubChoice is defined
> HB.instance Definition _ (T : countType) := [Countable of fset_type T by <:].
T is declared
```

```
HB_unnamed_factory_30 is defined
choice_Countable__to__choice_hasChoice is defined
choice_Countable__to__eqtype_hasDecEq is defined
choice_Countable__to__choice_Choice_isCountable is defined
HB_unnamed_mixin_34 is defined
fset_fset_type__canonical__choice_Countable is defined
fset_fset_type__canonical__choice_SubCountable is defined
```

Note that, although HB does provide an `##[hnf]` attribute, it should not be necessary in general.

The last message by HB is important because it indicates that the next command

```
> Canonical Structure fset_subCountType (T : countType) :=
>  Eval hnf in [subCountType of fset_type T].
Warning: Notation "[ subCountType of _ ]" is deprecated since mathcomp 2.0.0.
Use SubCountable.clone instead.
[deprecated-notation,deprecated]
fset_subCountType is defined
```

can now be removed.

To sum up, here follows the complete fix:

```
Section FinSets.
 Variable T : choiceType.
 ...
 HB.instance Definition _ := [isSub for elements by fset_type_rect].
 HB.instance Definition _ := [Equality of fset_type by <:].
 Canonical Structure fset_predType := PredType (fun (X : fset_type) x => nosimpl x \in elements X).
 HB.instance Definition _ := [Choice of fset_type by <:].
End FinSets.

HB.instance Definition _ (T : countType) := [Countable of fset_type T by <:].
Canonical Structure fset_subCountType (T : countType) :=
  Eval hnf in [subCountType of fset_type T].
```

References

[1] R. Affeldt, X. Allamigeon, Y. Bertot, Q. Canu, C. Cohen, P. Roux, K. Sakaguchi, E. Tassi, L. Théry, and A. Trunov. Porting the Mathematical Components library to Hierarchy Builder. In *the Coq workshop 2021*, Jul 2021.

[2] R. Affeldt and C. Cohen. Measure construction by extension in dependent type theory with application to integration, 2022.

[3] R. Affeldt, C. Cohen, and A. Saito. Semantics of probabilistic programs using s-finite kernels in coq. In *12th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2023), January 16–17, 2023, Boston, Massachusetts, USA*. ACM Press, Jan 2023.

[4] C. Cohen, K. Sakaguchi, and E. 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), June 29–July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 34:1–34:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[5] C. Doczkal and J. Bard. Completeness and decidability of modal logic calculi. `https://github.com/coq-community/comp-dec-modal`, 2023. Since 2017.

[6] F. Garillot, G. Gonthier, A. Mahboubi, and L. Rideau. Packaging mathematical structures. In *22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2009), Munich, Germany, August 17–20, 2009*, volume 5674 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2009.

[7] Hierarchy-Builder. High level commands to declare a hierarchy based on packed classes. `https://github.com/math-comp/hierarchy-builder`, 2023. Since 2020.