# Porting Coq Scripts to the Mathematical Components Library Version 2

Reynald Affeldt, Yves Bertot, Cyril Cohen, Pierre Roux, Kazuhiko Sakaguchi, Enrico Tassi

# Porting Coq Scripts to the Mathematical Components Library Version 2

Reynald Affeldt[2], Yves Bertot[1,4], Cyril Cohen[1,4], Pierre Roux[3], Kazuhiko Sakaguchi[1], and Enrico Tassi[1,4]

[1]Inria, France
[2]National Institute of Advanced Industrial Science and Technology (AIST), Japan
[3]ONERA / DTIS, Université de Toulouse, France
[4]Université Côte d'Azur, France

**Abstract**

The Mathematical Components library (hereafter, MathComp) [8] provides, among others, 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 [9] is the result of the port of MathComp to HB [1].

This document is a technical report whose goal is to explain how to port MathComp developments to MathComp 2. It has been written by the participants of the MathComp Documentation Sprint that happened from [2023-05-03] to [2023-05-10].

## Contents



## 1 Target Audience of this Document

Based on our experience porting several developments to MathComp 2, we can distinguish three categories of users:

1. MathComp users who have not been using the `Canonical` command should not see much difference compared to the past version upgrades of MathComp. Some identifiers that are now useless have been removed but this is documented in the changelog. For example `bool_eqType` might need to be replaced by `bool : eqType` or just `bool`. Also, the behavior of some rewritings might have changed, requiring explicit patterns. Typically, it might happen that rewriting with associativity lemmas requires the user to indicate whether it is supposed to happen on the left or the right-hand side of an equivalence relation, so that `rewrite addrA` might need to be rewritten `rewrite [in LHS]addrA` for example. See Sect. 4.4 for a concrete example. For such users, reading this tutorial through the end might not be necessary.

2. The target readers are primarily MathComp users who have been instantiating structures using the `Canonical` command.

3. As for the few users who have been developing their own hierarchies of structures, this tutorial might be of little help and they rather need to refer to:

   - 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],
   - already ported developments such as odd-order, multinomials, etc.

For the sake of concreteness, we illustrate the port of CompDecModal [5] in Sect. 4. Before that we review the basics of HB in Sect. 2 and review the documentation tools available for porting in Sect. 3.

## 2 Quick Reminder about the HB Vocabulary

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 consider the most basic scenario in generic terms. Here is the pattern to declare a structure `Struct` that sits 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}
```

Note that 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__struct_Struct is defined
```

The absence of this output often indicates failure of the `HB.instance` command.

# 3 Tools to Port MathComp Applications

## 3.1 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` [9].
- Additionally, structures are documented in the headers of Coq scripts 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` [9].

## 3.2 HB Commands Useful to Explore an Existing Hierarchy

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

### 3.2.1 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 *factory*: 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`.

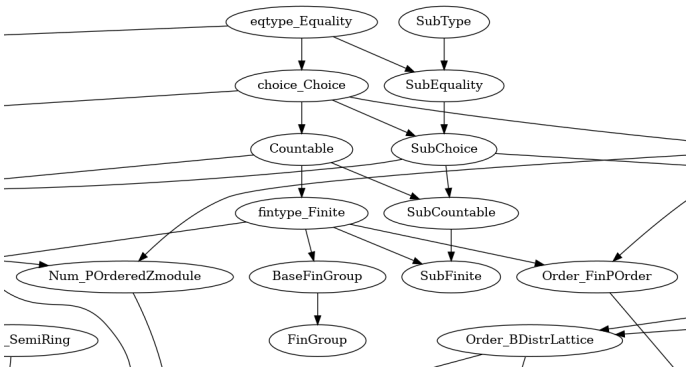


Figure 1: The vicinity of the structure `eqType` in MathComp

### 3.2.2 Information about Constructors with `HB.howto` and `HB.about`

To discover constructors that 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
```

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`.

### 3.2.3 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.

# 4 Porting a MathComp Development 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 Coq scripts, 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

[1] We have also observed at the time of this writing that the output of the `HB.instance` command may not be visible by default with VSCoq.

MathComp whose port involves fixing the instantiation of basic structures that most developments using MathComp are likely to use.

In the following, the offending commands appear in a gray areas

```
command incompatible with MathComp 2
```

and their fixes are singly famed:

```
MathComp 2 fix for the command above
```

## 4.1 Import the HB Library

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

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

## 4.2 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. 2) 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.

Finally, the last `Canonical` command causes a deprecation warning that needs to be addressed:

```
> 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
```

In fact, going back one step, it can be observed by the output the `HB.instance` commands that the instantiation of the `Countable` structure already triggered the instantiation of the `SubCountable` structure, rendering the last `Canonical` command harmful. It therefore needs to be deleted.

To sum up, here follows the complete fix:

```
1  Section FinSets.
2   Variable T : choiceType.
3   ...
4   HB.instance Definition _ := [isSub for elements by fset_type_rect].
5   HB.instance Definition _ := [Equality of fset_type by <:].
6   Canonical Structure fset_predType := PredType (fun (X : fset_type) x => nosimpl x \in elements X).
7   HB.instance Definition _ := [Choice of fset_type by <:].
8  End FinSets.
9
10 HB.instance Definition _ (T : countType) := [Countable of fset_type T by <:].
```

In fact, we can go one step further. Instead of instantiating the `Equality` structure at line 5 and then the `Choice` structure at line 7, one can start by instantiating the `Choice` structure and get the `Equality` structure automatically, so that a shorter fix would be:

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

HB.instance Definition _ (T : countType) := [Countable of fset_type T by <:].
```

This small example already illustrates the advantage of porting to MathComp 2.

## 4.3 Finitely Iterated Operators

The next series of compilation errors that occur when porting CompDecModal is about finitely iterated operators, which are likely to be used by most MathComp developments:

```
Canonical Structure fsetU_law (T : choiceType) :=
  Monoid.Law (@fsetUA T) (@fset0U T) (@fsetU0 T).
Canonical Structure fsetU_comlaw (T : choiceType) :=
  Monoid.ComLaw (@fsetUC T).
```

The first compilation error indicates a change in the signature of a constructor:

```
> Canonical Structure fsetU_law (T : choiceType) :=
>   Monoid.Law (@fsetUA T) (@fset0U T) (@fsetU0 T).
Error:
```

```
In environment
T : choiceType
The term "fsetUA (T:=T)" has type "forall X Y Z : {fset T}, X `|` (Y `|` Z) = X `|` Y `|` Z"
while it is expected to have type "Type".
```

We can use `HB.about` to discover the relevant structure (see Sect. 3.2.1):

```
> HB.about Monoid.Law.
HB: Monoid.Law.type is a structure (from "./ssreflect/bigop.v", line 415)
...
```

We can now use `HB.howto` to discover how to build this structure (see Sect. 3.2.2):

```
> HB.howto Monoid.Law.type
HB: solutions (use 'HB.about F.Build' to see the arguments of each factory F):
    - Monoid.isLaw
    - SemiGroup.isLaw; Monoid.isMonoidLaw
```

Finally, we can use `HB.about` to learn about the parameters of a constructot (see Sect. 3.2.2):

```
> HB.about Monoid.isLaw.Build
...
HB: arguments: Monoid.isLaw.Build T idm op opA op1m opm1
    - T : Type
    - idm : T
    - op : T -> T -> T
    - opA : associative op
    - op1m : left_id idm op
    - opm1 : right_id idm op
```

We have now enough information to fix the compilation error:

```
HB.instance Definition _ (T : choiceType) :=
  Monoid.isLaw.Build {fset T} fset0 fsetU (@fsetUA T) (@fset0U T) (@fsetU0 T).
```

Note that we have made explicit the *key* (the operator `fsetU`) in the instantiation. Though not strictly necessary, this is good practice to document it on this occasion.

Next compilation error:

```
> Canonical Structure fsetU_comlaw (T : choiceType) :=
>   Monoid.ComLaw (@fsetUC T).
Error:
In environment
T : choiceType
The term "fsetUC (T:=T)" has type "forall X Y : {fset T}, X `|` Y = Y `|` X"
  while it is expected to have type "Type".
```

This is similar to above: use `HB.about` to learn about `Monoid.ComLaw` and use `HB.howto` to inquire about its construction.

```
> HB.howto Monoid.ComLaw.type.
HB: solutions (use 'HB.about F.Build' to see the arguments of each factory F):
    - Monoid.isComLaw
    - SemiGroup.isComLaw; Monoid.isMonoidLaw
    - SemiGroup.isCommutativeLaw; Monoid.isLaw
    - SemiGroup.isLaw; SemiGroup.isCommutativeLaw; Monoid.isMonoidLaw
```

Note that since the key `fsetU` has already been equipped with a `Monoid.Law` structure, we can add this key as an additional parameter to `HB.howto` to restrict the search:

```
> HB.howto fsetU Monoid.ComLaw.type.
HB: solutions (use 'HB.about F.Build' to see the arguments of each factory F):
    - SemiGroup.isCommutativeLaw
```

We can then check the parameters of `SemiGroup.isCommutativeLaw.Build` to come up with the following fix:

```
HB.instance Definition _ (T : choiceType) :=
  SemiGroup.isCommutativeLaw.Build _ fsetU (@fsetUC T).
```

To sum up, we fix the compilation errors about finitely iterated operators using:

```
HB.instance Definition _ (T : choiceType) :=
  SemiGroup.isCommutativeLaw.Build _ fsetU (@fsetUC T).
HB.instance Definition _ (T : choiceType) :=
  Monoid.isLaw.Build {fset T} fset0 fsetU (@fsetUA T) (@fset0U T) (@fsetU0 T).
```

In fact, we could have proceeded with only one instantiation since we can get the `Monoid.Law` structure from the instantiation of the `Monoid.ComLaw` structure, so that a better fix would be:

```
HB.instance Definition _ (T : choiceType) :=
  Monoid.isComLaw.Build _ _ fsetU (@fsetUA T) (@fsetUC T) (@fset0U T).
```

## 4.4 Other Compilation Errors

Since other compilation errors are similar to the one already explained above, we go faster about them.

**Instantiation of an `Equality` Structure** Next failure in the file `K/K_def.v`:

```
Definition form_eqMixin := EqMixin (compareP eq_form_dec).
Canonical Structure form_eqType := Eval hnf in @EqType form form_eqMixin.
```

The failure is primarily caused by the removal of `EqMixin`. The changelog suggests to use the constructor `hasDecEq.Build` whose parameters can be double-checked with `HB.howto` leading to the following fix:

```
HB.instance Definition _ := hasDecEq.Build form (compareP eq_form_dec).
```

**Instantiation of a `Countable` Structure** Next failure in the file `K/K_def.v`:

```
Definition form_countMixin := PcanCountMixin formChoice.pickleP.
Definition form_choiceMixin := CountChoiceMixin form_countMixin.
Canonical Structure form_ChoiceType := Eval hnf in ChoiceType form form_choiceMixin.
Canonical Structure form_CountType := Eval hnf in CountType form form_countMixin.
```

We learn that `PcanCountMixin` is deprecated and that `CountChoiceMixin` is no more. The changelog suggests to use `PCanIsCountable` instead of `PcanCountMixin`. According to `Locate`, `PCanIsCountable` is located in the file `ssreflect/choice.v` and it has the following type:

```
PCanIsCountable :
forall [T : countType] [sT : Type] [f : sT -> T] [f' : T -> option sT], pcancel f f' ->
  isCountable.axioms_ sT
```

The fix in this case is therefore:

```
HB.instance Definition _ : isCountable form := PCanIsCountable formChoice.pickleP.
```

Note that we have added a type information to make the key (here, `form`) explicit as recommended when performing instantiations. This fix is sufficient because it generates the `Choice` instance along with the `Countable` instance.

**More Instantiations of `Equality` Structures** The next compilation errors in the files `Kstar_def.v`, `gen_def.v`, and `CTL_def.v` are handled as already explained above.

**Failing Rewriting** The next compilation "error" is in the file `CTL/demo.v`. It is actually a tactic whose execution has been substantially slowed down by the upgrade to MathComp 2:

```
move => [p' y]. rewrite /MRel /Mstate (negbTE (root_internal _)) [_ && _]/= orbF.
```

The offending rewrite is the one with `orbF`. It used to apply by default to the left-hand side of an `<->` equivalence but now the user is required to indicate the rewrite location with a pattern to recover a reasonable execution time:

```
move => [p' y]. rewrite /MRel /Mstate (negbTE (root_internal _)) [_ && _]/=.
rewrite [in X in X <-> _]orbF.
```

This kind of slow down is however rather exceptional.

There are a few more compilation errors but they are similar to the ones we explained so far.

# 5 Conclusion

This document illustrated the port of a typical MathComp development to MathComp 2. We reviewed the available tools (documentation and HB tactics) and went through a number of concrete sample errors and warnings that we explained and corrected.

The example of porting CompDecModal demonstrated that the use of HB improves the Coq scripts in terms of readability and even allows for improvements. The complete fix can be found online: branch (relevant commit), it required the edition of 10 files, 35 insertions, and 67 deletions, which arguably represents a moderate amount of work.

The migration process to MathComp 2 will surely generate more questions that the community is ready to answer via the math-comp streams on `https://coq.zulipchat.com`.

# 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.

[8] A. Mahboubi and E. Tassi. *Mathematical Components*. Zenodo, Jan 2021.

[9] MathComp 2. Mathematical Components. `https://github.com/math-comp/math-comp/tree/hierarchy-builder`, May 2023. Branch to prepare the forthcoming release of the Mathematical Components library.