A hierarchy of ordered types in Mathematical Components

Xavier Allamigeon[1], Cyril Cohen[2], Kazuhiko Sakaguchi[3], and Pierre-Yves Strub[4]

[1] Inria and CMAP, CNRS, École Polytechnique, Institut Polytechnique de Paris, France
[2] Inria, Université Côte d'Azur, France
[3] University of Tsukuba, Japan
[4] LIX, CNRS, École Polytechnique, Institut Polytechnique de Paris, France
xavier.allamigeon@inria.fr, Cyril.Cohen@inria.fr,
sakaguchi@logic.cs.tsukuba.ac.jp, pierre-yves@strub.nu

Overview Up to version 1.10.0, the Mathematical Components (MathComp) library [6] lacked ordered structures. Because of its applications (see below), this library must be integrated with the rest of MathComp structures and provide partially and totally ordered types, meet-semilattices, non-distributive and distributive lattices respectively called `porderType`, `orderType`, `meetSemilatticeType`, `latticeType`, and `distrLatticeType` in Figure 1. Some types must have several ordering: e.g., products have point-wise and lexicographic orderings, and any ordered type $(T, \leq)$ has its dual $(T, \geq)$. We develop such a library using the packed classes methodology [4], which we extend with a display mechanism, in order to address the latter requirement.

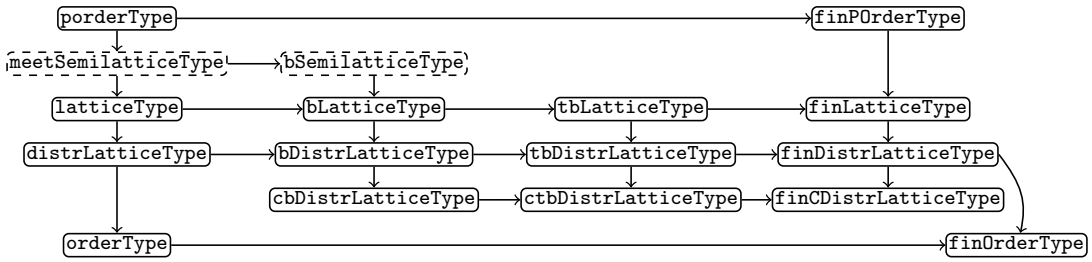

Figure 1: Structures with solid borders are in MathComp 1.11, others are the work-in-progress. `fin`, `b`, `t`, and `c`, respectively denote finiteness, the existence of bottom, top, and complement.

Displays With canonical structures a given carrier may have at most one instance of each structure. The MathComp way to deal with this shortcoming is to create a canonical instance on an alias. However, operators of two structures on the same carrier are printed in the same way, which is confusing. To solve this problem, we introduce the notion of *display*, which is an extra parameter to a structure, and which is only used for pretty-printing purposes. They are opaque elements of type `unit`, exactly like *keys* in MathComp. The user can then tune their notations to print operators in a different way depending on the value of the key.

For example, the structure `porderType` has signature `unit → Type`, and the operator `le` has type `∀ (disp : unit) (T : porderType disp), rel T`. If `P` is a preorder with display `d`, the alias `dual P := P` is canonically a preorder with display `dual_display d`, and we have the notations

```
Notation "x  ≤  y" := (@le _                x y) : order_scope.  (* default display *)
Notation "x ≤^d y" := (@le (dual_display _) x y) : order_scope.  (* default dual display *)
```

where `x ≤^d y` is convertible to `y ≤ x` and `dual_display : unit → unit` is an opaque function.

Thus, `(≤ : rel nat)` and `(≤ : rel (dual nat))` are respectively interpreted as the "less than or equal to" relation of `nat` and its dual, and printed as $\leq$ and `≤^d`. This interpretation is done by looking up the table of canonical instances indexed by head symbols of carriers such as `nat`, `dual`, and other types and aliases. As of now, displays are printing-only: input must be done through explicit type annotations. Future plans include accepting them as input too, but the infrastructure is so heavy that it would require a modification of the automated hierarchy builder tool [3].

Definitionally involutive duals We present a new instance of forgetful inheritance [1], that is, definitionally involutive dual ordering, a work-in-progress feature of MathComp. Since the fundamental idea of forgetful inheritance is implementing inheritance by inclusion rather than construction, we redefine the mixin record of `porderType` as a primitive record that includes all the dual axioms as in Figure 2. Consequently, the construction of dual can be done by transposing axioms to their duals, and iterating it twice becomes a definitional identity function. But this change does not break user code, because the old mixin record has been redesigned as a factory [1, 3] to implicitly derive an actual mixin instance by coercion.

Since displays are opaque and prevent us to make duals definitionally involutive, we also have to make `dual_display` transparent, while CoqMT [5] may help us to extend conversion with an equation `dual_display (dual_display disp) = disp`. Doing so allows, for example, to ease the reduction of the definition and properties of coatoms of a lattice to the ones of atoms in the dual lattice. Indeed, when doing so, it is not rare that one has to deal with duals of duals, and in that case, has to use type casts for converting back elements in the dual of the dual of a lattice to that lattice. Now that the dualization of lattices is definitionally involutive, all these type casts can be removed.

```
Record POrder.mixin_of := Mixin {
  le : rel T; lt : rel T;
  lt_def : ∀x y, lt x y = (y ≠ x) && le x y;

  refl   : reflexive     le;
  anti   : antisymmetric le;

  trans  : transitive    le; }.
```

```
Set Primitive Projections.
Record POrder.mixin_of := Mixin {
  le : rel T; lt : rel T;
  lt_def  : ∀x y, lt x y = (y ≠ x) && le x y;
  lt_def' : ∀x y, lt y x = (y ≠ x) && le y x;
  refl    : reflexive     le;
  anti    : antisymmetric le;
  anti'   : antisymmetric (fun x y⇒ le y x);
  trans   : transitive    le;
  trans'  : transitive    (fun x y⇒ le y x); }.
```

Figure 2: The `porderType` mixins defined for `(T : eqType)` without (left) and with (right) definitionally involutive duals. The latter one includes all the dual axioms (highlighted by red) that replaces `le` and `lt` with their transpositions in their counterparts (highlighted by blue).

Applications In MathComp Analysis [1], we manipulate several ordered types such as numeric domains, extended reals, and nonnegative subsets of numeric domains. We have successfully defined and reasoned about those instances using displays. Also, the present work help us to factor out the notion of absolute values and norms as normed Abelian groups as in [1, Sect. 4.2].

In Coq-Polyhedra [2], the introduction of meet semilattices has allowed to simplify the order related properties of polyhedra (as subsets of $\mathbb{R}^n$) in a significant way. Moreover, faces of polyhedra have a structure of finite lattice with many special properties. The fact that the collection of the face lattices of polytopes (i.e. bounded polyhedra) are closed under sublattices play a central role in many proofs by induction. This has motivated the formalization of sublattices, as well as other features such as atomicity and coatomicity. In a future work, dual lattices will be used in order to formalize the polar of polytopes.

References

[1] R. Affeldt, C. Cohen, M. Kerjean, A. Mahboubi, D. Rouhling, and K. Sakaguchi. "Competing inheritance paths in dependent type theory: a case study in functional analysis". accepted in the proceedings of IJCAR '20, available at https://hal.inria.fr/hal-02463336. 2020.

[2] X. Allamigeon, R. D. Katz, and P.-Y. Strub. "The Face Lattice of Polyhedra". accepted in the proceedings of IJCAR '20. 2020.

[3] C. Cohen, K. Sakaguchi, and E. Tassi. "Hierarchy Builder: algebraic hierarchies made easy in Coq with Elpi". accepted in the proceedings of FSCD '20, available at https://hal.inria.fr/hal-02478907. 2020.

[4] F. Garillot, G. Gonthier, A. Mahboubi, and L. Rideau. "Packaging Mathematical Structures". In: *TPHOLs '09*. Vol. 5674. LNCS. Springer, 2009, pp. 327–342.

[5] J.-P. Jouannaud and P.-Y. Strub. "Coq without Type Casts: A Complete Proof of Coq Modulo Theory". In: *LPAR-21*. Vol. 46. EPiC Series in Computing. EasyChair, 2017, pp. 474–489.

[6] The Mathematical Components project. *The Mathematical Components repository*. 2020. URL: https://github.com/math-comp/math-comp.