Generating Mathematical Structure Hierarchies using Coq-ELPI

Cyril Cohen *(Inria)*, Kazuhiko Sakaguchi, Enrico Tassi

FoMM, Pittsburgh, USA
January 6th, 2020

Structures in Mathematics

- A **carrier** in Set / Type,
- A set of **constants** in the carrier, and **operations**,
- Proofs of the **axioms** of the structure

Structures in Mathematics

- A **carrier** in Set / Type,
- A set of **constants** in the carrier, and **operations**,
- Proofs of the **axioms** of the structure

```
Record is_ring A := mk_ring {
   zero : A; add : A -> A -> A; opp : A -> A;
   one : A;  mul : A -> A -> A;
   addrA : associative add;
   addrC : commutative add;
   add0r : left_id zero add;
   addNr : left_inverse zero opp add;
   mulrA : associative mul;
   mul1r : left_id one mul;
   mulr1 : right_id one mul;
   mulrDl : left_distributive mul add;
   mulrDr : right_distributive mul add;
 }.
```

Structures in formalization

Purpose:

- factor theorems across instances, using the **theory** of each structure,
- **automatically find** which structures hold on which types.

Structures in formalization

Purpose:

- factor theorems across instances, using the **theory** of each structure,
- **automatically find** which structures hold on which types.

Requirements:

- declare a **new instance**,

Structures in formalization

Purpose:

- factor theorems across instances, using the **theory** of each structure,
- **automatically find** which structures hold on which types.

Requirements:

- declare a **new instance**,
- declare a **new structure**
  - above, below or in the middle
  - handle diamonds (e.g. monoid, group, commutative or not),
  - by amending existing code, or not,

Structures in formalization

Purpose:

- factor theorems across instances, using the **theory** of each structure,
- **automatically find** which structures hold on which types.

Requirements:

- declare a **new instance**,
- declare a **new structure**
  - above, below or in the middle
  - handle diamonds (e.g. monoid, group, commutative or not),
  - by amending existing code, or not,
- **predictability** of inferred instance,

Structures in formalization

Purpose:

- factor theorems across instances, using the **theory** of each structure,
- **automatically find** which structures hold on which types.

Requirements:

- declare a **new instance**,
- declare a **new structure**
  - above, below or in the middle
  - handle diamonds (e.g. monoid, group, commutative or not),
  - by amending existing code, or not,
- **predictability** of inferred instance,
- **robustness** of user code with regard to *new declarations*.x

Structures relating to each other

Examples:

- Monoid $\leftarrow$ Group $\leftarrow$ Ring $\leftarrow$ Field $\leftarrow$ ...
- Euclidean Spaces $\rightarrow$ Normed Spaces $\rightarrow$ Complete Space $\rightarrow$ Metric Spaces $\rightarrow$ Topological Spaces $\rightarrow$ ...

Structures relating to each other

Examples:

- Monoid $\leftarrow$ Group $\leftarrow$ Ring $\leftarrow$ Field $\leftarrow$ ...
- Euclidean Spaces $\rightarrow$ Normed Spaces $\rightarrow$ Complete Space $\rightarrow$ Metric Spaces $\rightarrow$ Topological Spaces $\rightarrow$ ...

Going through arrows must be automated.

Structures relating to each other

Examples:

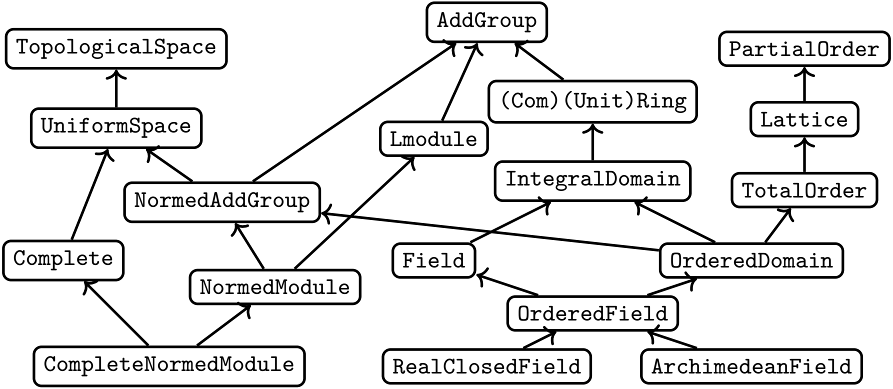
- Monoid $\leftarrow$ Group $\leftarrow$ Ring $\leftarrow$ Field $\leftarrow$ …
- Euclidean Spaces $\rightarrow$ Normed Spaces $\rightarrow$ Complete Space $\rightarrow$ Metric Spaces $\rightarrow$ Topological Spaces $\rightarrow$ …

Going through arrows must be automated.

Two kinds arrows:

- Extensions: add operations, axioms or combine structures
- Entailment/Induction/Deduction/Generalization.

More examples



"Calculus" structures

"Algebraic" structures

Structure extension

- **Compositional**: no need to start from scratch every time. (E.g. the product of two groups is a group, the product of two rings is a ring),

Structure extension

- **Compositional**: no need to start from scratch every time. (E.g. the product of two groups is a group, the product of two rings is a ring),
- **Noisy**: changes the internal definition of a structure. (E.g. Defining an commutative monoid from a monoid, we get already get one unnecessary axiom),

Structure extension

- **Compositional**: no need to start from scratch every time. (E.g. the product of two groups is a group, the product of two rings is a ring),
- **Noisy**: changes the internal definition of a structure. (E.g. Defining an commutative monoid from a monoid, we get already get one unnecessary axiom),
- **Non-robust**: possible breakage of user code when new intermediate structures are added,

Structure extension

- **Compositional**: no need to start from scratch every time. (E.g. the product of two groups is a group, the product of two rings is a ring),
- **Noisy**: changes the internal definition of a structure. (E.g. Defining an commutative monoid from a monoid, we get already get one unnecessary axiom),
- **Non-robust**: possible breakage of user code when new intermediate structures are added,
- **Not all arrows!**

Structure extension

- **Compositional**: no need to start from scratch every time. (E.g. the product of two groups is a group, the product of two rings is a ring),
- **Noisy**: changes the internal definition of a structure. (E.g. Defining an commutative monoid from a monoid, we get already get one unnecessary axiom),
- **Non-robust**: possible breakage of user code when new intermediate structures are added,
- **Not all arrows! Really?**

Structure entailment

- **More flexible**: no need to cut structures into smaller bits.
- Cover the case of **all arrows**, including extensions.

Structure entailment

- **More flexible**: no need to cut structures into smaller bits.
- Cover the case of **all arrows**, including extensions.

- Major breakage when arbitrary entailment is automatic; e.g. given two normed spaces, and making their Cartesian product, in order to obtain the resulting topology, one can either:
  - first consider the normed space product, then derive the corresponding topological space, or
  - first derive the topological spaces and then consider the topological space product.

Our Design

The best of two the worlds:

- **Extension**, through *mixins* for **internal declaration** and **automatic inference**
- **Entailment**, through *factory* for any other use.
  Factories require mixins and can produces others. (e.g. a full axiomatic can provide all the pieces)

Our Design

The best of two the worlds:

- **Extension**, through *mixins* for **internal declaration** and **automatic inference**
- **Entailment**, through *factory* for any other use.
  Factories require mixins and can produces others. (e.g. a full axiomatic can provide all the pieces)

We generate and generalize the design from *Packaging Mathematical Structures* (Garillot *et al.*) and *Canonical Structures for the working Coq user* (Mahboubi and Tassi).

Our Design

The best of two the worlds:

- **Extension**, through *mixins* for **internal declaration** and **automatic inference**
- **Entailment**, through *factory* for any other use.
  Factories require mixins and can produces others. (e.g. a full axiomatic can provide all the pieces)

We generate and generalize the design from *Packaging Mathematical Structures* (Garillot *et al.*) and *Canonical Structures for the working Coq user* (Mahboubi and Tassi).

We follow a fully bundled approach, where carriers are packaged together with their axiomatic.

Hiearchy builder at work

Five commands:

- `declare_mixin FactoryModuleName TypeName Factories.`
- `declare_factory FactoryModuleName TypeName Factories.`
- `end Functions.`
- `structure ModuleName Factories.`
- `instance Carrier Factories.`

Demo

https://github.com/math-comp/hierarchy-builder

Conclusion

- High-level commands to declare structures and instances, easy to use.
- Predictable outcome of inference,
- Takes into account the evolution of knowledge
  - which is formalized, and
  - which the user has.

  The two knowledge do not need to be correlated.
- Robustness with regard to new declaration *and even changes of internal implementation*.

Conclusion

- High-level commands to declare structures and instances, easy to use.
- Predictable outcome of inference,
- Takes into account the evolution of knowledge
  - which is formalized, and
  - which the user has.

  The two knowledge do not need to be correlated.
- Robustness with regard to new declaration *and even changes of internal implementation*.

Also, Coq-ELPI turned out to be a very comfortable meta-programming language for this.

Future work on Hierarchy Builder

- Adding support for parameters.
- Generating hierarchies of morphisms from structures.
- Generating hierarchies of subobjects from structures.
- Supporting multiple instances on the same carrier.
- Replacing all uses in math-comp and extensions.
- Get better error messages.