The Algebraic Library

Assia Mahboubi

14 March 2012

MAP INTERNATIONAL SPRING SCHOOL ON FORMALIZATION OF MATHEMATICS 2012

SOPHIA ANTIPOLIS, FRANCE / 12-16 MARCH

Inria informatics mathematics

ihp Institut Henri Poincaré

MICROSOFT RESEARCH INRIA JOINT CENTRE

Mathematical structures

Consider the three following objects:

1. $\{x \in \mathbb{C} \mid x^n = 1\}$ for a given n, and the product of complex numbers;
2. The set $\{0, \ldots, m-1\}$ of natural numbers, for a given m, and the addition modulo m of natural numbers;
3. Bijective isometries globally preserving the vertexes of a given cube, and their composition.

What do they have in common?

Mathematical structures

If we call $*$ the operation considered in each case:

1. Product of complex numbers
2. Addition modulo m
3. Composition

We can notice that the following properties hold in each case:

- $\forall x\ y\ z, \quad x * (y * z) = (x * y) * z$
- there is a particular element e such that:

$$\forall x, \quad x * e = e * x = x$$

- $\forall x, \exists x' \quad x * x' = x' * x = e$

Mathematical structures

These properties are in fact called the axioms of group.

Their consequences is called (finite) group theory.

We say that our three examples are instances of the finite group structure, and they therefore enjoy all the previous consequences.

Mathematical structures

Codifying this language, ordering its vocabulary and clarifying its syntax is a useful work which is indeed one of the aspects of the axiomatic method [...]. But - and we insist on this point - this is only one of its aspects, and it is certainly the less interesting.

N. Bourbaki "The architecture of mathematics" 1962.

Mathematical structures

Codifying this language, ordering its vocabulary and clarifying its syntax is a useful work which is indeed one of the aspects of the axiomatic method [...]. But - and we insist on this point - this is only one of its aspects, and it is certainly the less interesting.

The essential motivation of the axiomatic method is precisely to define what the logical formalism is alone unable to provide, which is the profound intelligibility of mathematics.

N. Bourbaki "The architecture of mathematics" 1962.

Mathematical structures for formal proofs

Today we will:

- use a computer proof assistant rather than pen and paper;
- implement known mathematical structures, and not discover new ones;
- rely on type-theoretic foundations rather than set-theoretic;
- use techniques from software engineering to organize modular formal developments.

The main motivation is to share efficiently notations and theories, and not to reach the higher level of abstraction.

Lessons from yesterday

Two key ingredients:

- Record types
- Inference via canonical instances

$\Rightarrow$ a generic pattern to organize the formalized content.

Typical scheme

```
Structure my_struct := My_struct{
  dom  : Type;
  c    : dom;
  op   : dom -> dom;
  opP1 : forall x, P x;
  ...}.
```

```
Notation "x * y" := (op x y).
```

```
Section my_struct_theory.

Variable s : my_struct.

Lemma foo : forall x : dom s, Q x.
Proof. ... Qed.

End my_struct_theory.
```

Typical scheme

```
Structure my_struct := My_struct{
  dom  : Type;
  c    : dom;
  op   : dom -> dom;
  opP1 : forall x, P x;
  ...}.
```

```
Notation "x * y" := (op x y).
```

```
Section my_struct_theory.

Variable s : my_struct.

Lemma foo : forall x : dom s, Q x.
Proof. ... Qed.

End my_struct_theory.
```

```
Canonical nat_my_struct := My_struct nat _ _ ....

Theorem very_hard : forall n : nat, R (n * n).
Proof.
...
...
...
...; apply: foo.
...
...
Qed.
```

From a single structure to several ones

The problem we have not addressed yet is the interaction between structures. Issues are:

- inheritance
- sharing
- scalability to many structures and many instances

Naive approaches (like name spaces or simply nested records) do not scale.

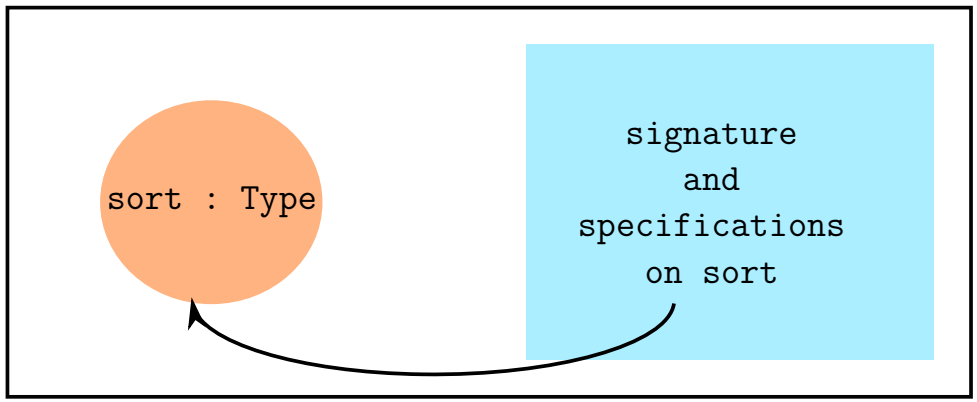
Skeleton of a structure

Records modeling mathematical structures have a special shape:

```
Structure eqType := EqType {
sort : Type;
eq_op : sort -> sort -> bool;
_ : forall x y, reflect (x = y) (eq_op x y)}.
```

- The `sort` type has a special status.
- Other projections are:
  - Constants and operations (signature)
  - Specifications (axioms)

Skeleton of a structure



: type

: class

providing T + T is building an instance T T of type

Design pattern for structures

The design implementation follows this scheme:

- the type of a structure is a two projection record type;
- the first projection gives the carrier type;
- the second projection is itself a (dependent) record type containing the signature and specifications.

The record type describing the signature/specification is called a class.

Example: the eqType structure

```
Module Equality.

Definition axiom T (e : rel T) := forall x y,
    reflect (x = y) (e x y).

Structure class_of (T : Type) :=
  Class {op : rel T; _ : axiom op}.

Structure type :=
  Pack {sort : Type; _ : class_of sort}.

End Equality.
```

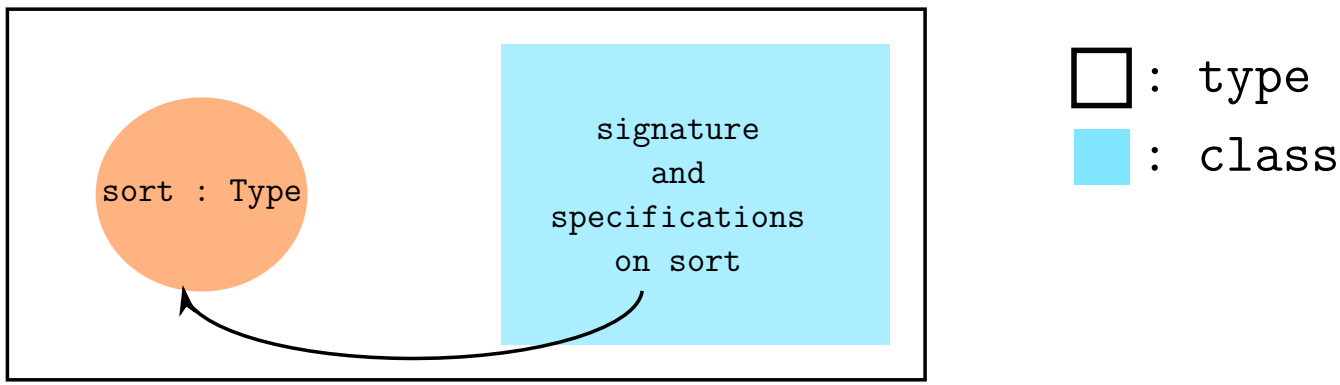
Example: the eqType structure

```
Module Equality.

Definition axiom T (e : rel T) := forall x y,
    reflect (x = y) (e x y).

Structure mixin_of (T : Type) :=
  Mixin {op : rel T; _ : axiom op}.

Notation class_of := mixin_of.

Structure type :=
  Pack {sort; _ : class_of sort}.

End Equality.
```

Example: the eqType structure

Qualified generic names are hidden behind the exported notation we have used so far:

```
Notation eqType := Equality.type.
Notation EqMixin := Equality.Mixin.
Notation EqType T m := Equality.Pack.
```

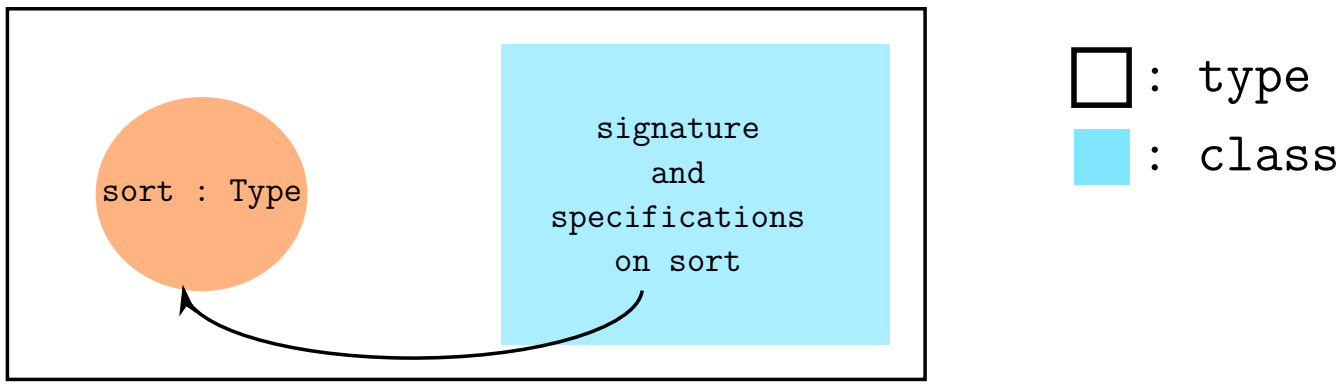


providing `T` + `T` is building an instance `T` `T` of type

Example: the eqType structure

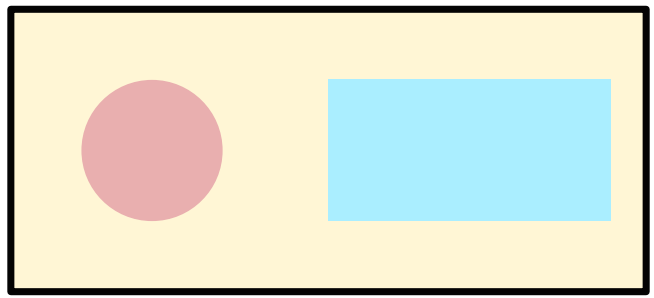
Hence building and instance follows the pattern:

```
Fixpoint eqn (m n : nat) {struct m} : bool := ...
Lemma eqnP : Equality.axiom eqn. Proof. ... Qed.

Canonical nat_eqMixin := EqMixin eqnP.
Canonical nat_eqType : eqType :=
   EqType nat nat_eqMixin.
```

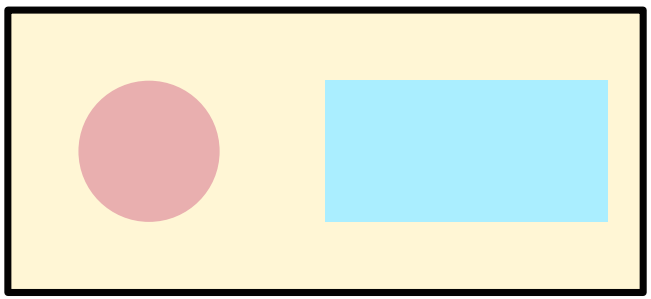


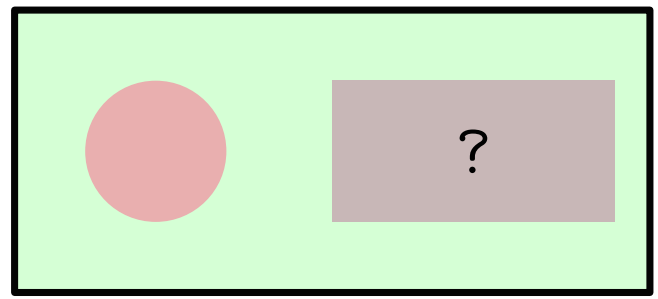
providing T + T is building an instance T T of type

Slicing

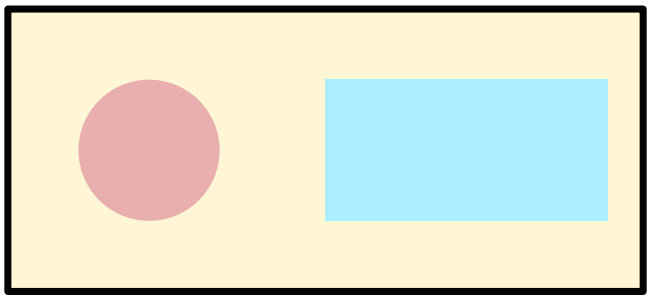


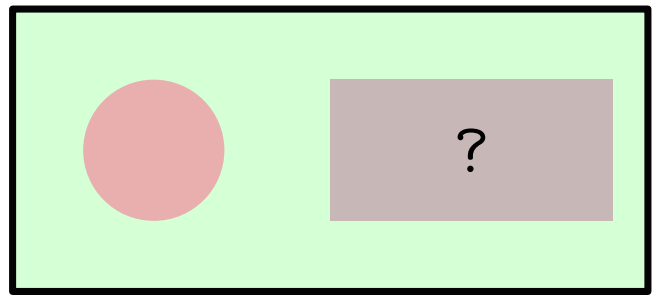
Slicing



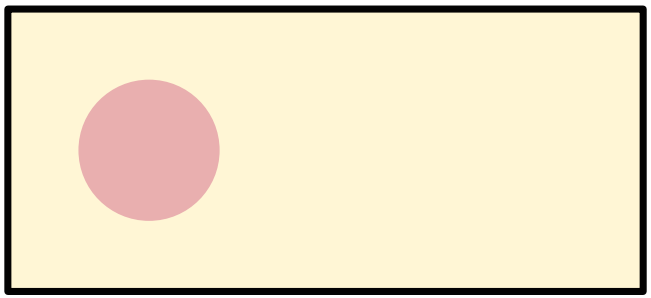


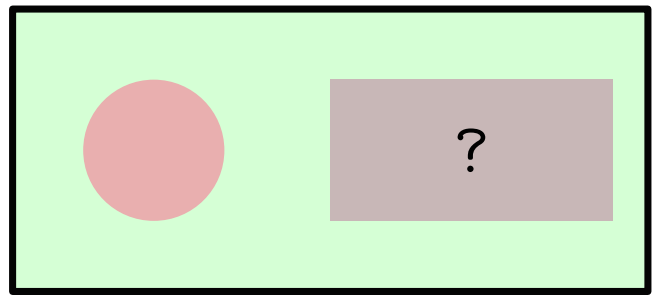
Slicing



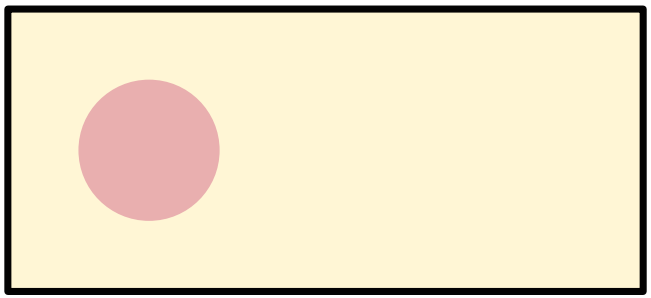


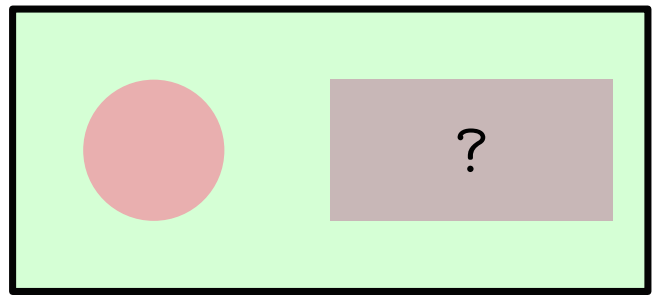
Slicing



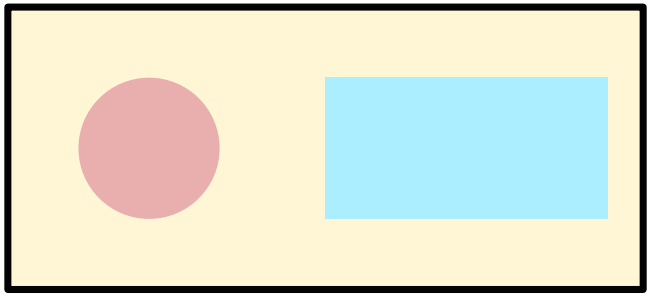


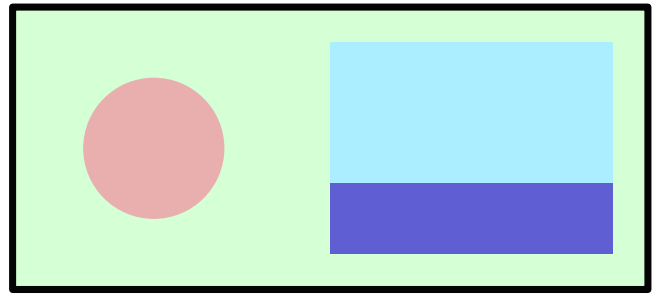
Slicing



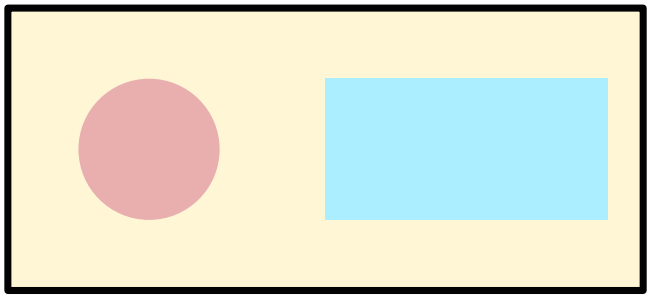


Slicing

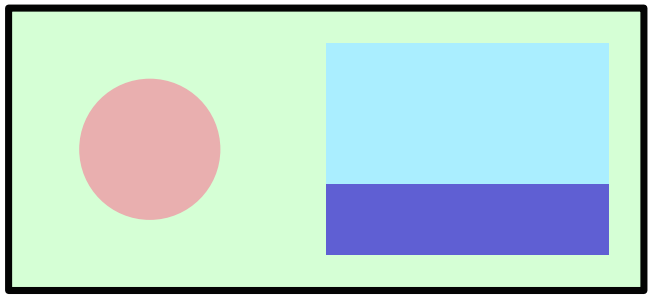




Slicing



`class`



`base`

`mixin`

`class`

Summary

- Each structure is modelled by a two projection record
  - one for the carrier (sort) type
  - one for the (signature/specification) class applied to the sort
- The class is represented by a record type, parametrized by a type.

Summary

- Each enriched structure is defined using an extra record type called a mixin, parametrized by a type.
- This mixin described the extra content one want to add to existing structures
- Each class (except the very first ones) is itself a two projection record
  - a base one, which is the class of a previous structure
  - a mixin one, applied to the base.

Instantiation

Thanks to a careful encoding via canonical structures, the user only needs to:

- Define the operations and prove the desired specifications
- Define the successive slices (mixins) feeding the relevant `Mixin` constructors with the previous material;
- Redundant content (like bases) is inferred automatically.

Example

```
Record polynomial (R : ringType) := Polynomial
   {polyseq :> seq R; _ : last 1 polyseq != 0}.
```

And now:

```
Definition polynomial_eqMixin := ...
Canonical polynomial_eqType :=
  EqType polynomial polynomial_eqMixin.
```

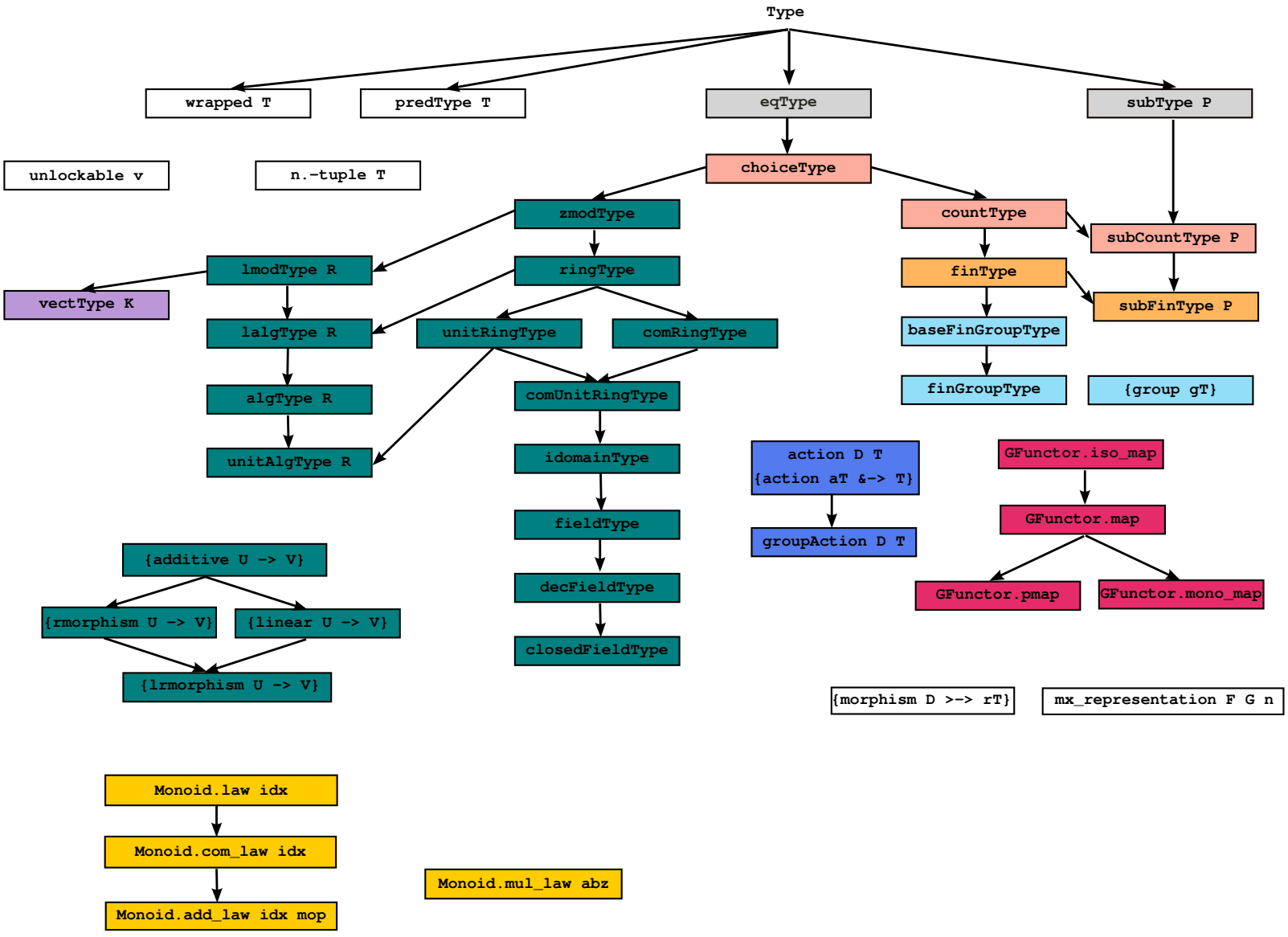
Example

```
Record polynomial (R : ringType) := Polynomial
   {polyseq :> seq R; _ : last 1 polyseq != 0}.
```

After some more work:

```
Definition poly_zmodMixin :=
  ZmodMixin add_polyA add_polyC add_poly0
      add_poly_opp.
Canonical poly_zmodType :=
  ZmodType {poly R} poly_zmodMixin.
```

Description of the existing hierarchy



Content associated with a structure

- Possibly several constructors of mixins
- Notations
- Properties on operations, often expressed by standardized predicates
- Generic lemmas, with a uniform name policy

Summarized on the cheat sheet you will get during the lab session.