Type Classes for Mathematics

Bas Spitters Eelis van der Weegen

Radboud University Nijmegen
Formath EU project

ITP 2010

Interfaces for mathematical structures:

- Algebraic hierarchy (groups, rings, fields, ...)
- Relations, orders, ...
- Categories, functors, ...
- Algebras over equational theories
- Numbers ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, ...)

Need solid representations of these.

Representing interfaces in Coq

Engineering challenges:

- Structure inference
- Multiple inheritance/sharing
- Convenient algebraic manipulation (e.g. rewriting)
- Idiomatic use of notations

Solutions in Coq

Existing solutions:

- Dependent records
- Packed classes (Ssreflect)
- Modules

All of these have problems.

New solution: Use type classes!
Coq has *first class* type classes.

Solutions in Coq

Existing solutions:

- Dependent records
- Packed classes (Ssreflect)
- Modules

All of these have problems.

New solution: Use type classes!
Coq has *first class* type classes.

Bundling

Core principle in our approach:

> Represent algebraic structures as predicates,
>
> ... over fully *unbundled* components.

Fully unbundled:

```
Definition reflexive {A: Type} (R: relation A): Prop
  := Π a, R a a.
```

- Very flexible *in theory*
- Inconvenient *in practice* (without type classes!):
  - Nothing to bind notations to
  - Declaring/passing inconvenient
  - No structure inference
- Hence: existing solutions choose to bundle.

Fully unbundled:

```
Definition reflexive {A: Type} (R: relation A): Prop
  := Π a, R a a.
```

- Very flexible *in theory*
- Inconvenient *in practice* (without type classes!):
  - Nothing to bind notations to
  - Declaring/passing inconvenient
  - No structure inference
- Hence: existing solutions choose to bundle.

Bundling is bad

Fully unbundled (the other end of the spectrum):

```
Record ReflexiveRelation: Type :=
  { A: Type;   R: relation A;   proof: Π a, R a a }.
```

Addresses *some* of the problems:

- Structure inference
- Notations
- Declaring/passing

But also introduces new ones:

- Prevents sharing
- Multiple inheritance (diamond problem)
- Long projection paths

Bundling is bad

Fully unbundled (the other end of the spectrum):

```
Record ReflexiveRelation: Type :=
  { A: Type;   R: relation A;   proof: Π a, R a a }.
```

Addresses *some* of the problems:

- Structure inference
- Notations
- Declaring/passing

But also introduces new ones:

- Prevents sharing
- Multiple inheritance (diamond problem)
- Long projection paths

Solving problems with type classes

Revised SemiGroup:

```
Class Equiv (A: Type) := equiv: relation A.
Class SemiGroupOp (A: Type) := sg_op: A → A → A.

Infix "=" := equiv.
Infix "∗" := sg_op.

Class SemiGroup
      (G: Type) {e: Equiv G} {op: SemiGroupOp G}: Prop :=
  { sg_setoid:> Equivalence e
  ; sg_ass:> Associative op
  ; sg_proper:> Proper (e ⇒ e ⇒ e) op }.
```

Algebra

We have formalized:

- Algebraic hierarchy
- Universal algebra
- Category theory

Included: unification-based quoting

Conclusions

Predicate type classes for mathematics:

- Works well in practice
- Match mathematical practice
- Compatible with efficient computation
- Plan: use as basis for computational analysis (Formath)

Sources/papers:

Google keywords: `coq math classes`