Type Classes in CoQ

Elisabet Lobo-Vesga

EAFIT University

28th April, 2014
What is CoQ? ¹

Coq is a proof assistant developed in France since 1989. It is based on an formal language called Calculus of Inductive Constructions (CIC). Coq allows to:

- Define functions or predicates
- State mathematical theorems
- Interactively develop formal proofs of these theorems
- Check these proofs
- Extract certified programs to languages like OCaml or Haskell
- Use a tactic language for letting the user define its own proof methods

¹Coq website http://coq.inria.fr/what-is-coq
The Coq bundle

- Arithmetics in $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$
- Libraries about list, finite sets, finite maps, etc.
- coqtop: interactive mode
- coqide: graphical user interface
- coqdoc and coq-tex: documentation tools
- coqc: the compiler (batch compilation)
- coqchk: stand-alone proof verifier (validation of compiled libraries)
Declarations

A declaration associates a \textit{name} with a \textit{specification}.

- \textbf{Name}: identifier
- \textbf{Specification}: formal expression as logical propositions (\textit{Prop}), mathematical collections (\textit{Set}) and abstract types (\textit{Type})

\begin{verbatim}
name : sort

0    : nat
nat  : Set
Set  : Type
Prop : Type
>    : nat \rightarrow nat \rightarrow Prop
list : Type \rightarrow Type
\end{verbatim}

---

Inductive nat : Set :=
| 0 : nat
| S : nat -> nat.

Definition one := (S 0).
Definition two : nat := S one.
Definition double (m:nat) := plus m m m.
Introduction to Coq \(^2\)

Proofs

Variables A B C : Prop.

Lemma lem :
  (A -> B -> C) -> (A -> B) -> A -> C.

Proof.
intro H.
intros H’ HA.
apply H.
exact HA.
apply H’.
assumption.
Qed.
Haskell Type Classes

Definition

"Typeclasses define a set of functions that can have different implementations depending on the type of data they are given." ³

Parametric polymorphism

“Occurs when a function is defined over a range of types, acting in the same way for each type.”\(^4\)

Ad-Hoc polymorphism (Overloading)

“Occurs when a function is defined over several different types, acting in a different way for each type.”\(^4\)

class Functor f where
  fmap :: (a -> b) -> f a -> f b

data List a = [] | a : [a]
data Maybe a = Nothing | Just a
class Functor f where
    fmap :: (a -> b) -> f a -> f b

instance Functor List where
    fmap _ [] = []
    fmap f (x:xs) = f x : fmap f xs

instance Functor Maybe where
    fmap _ Nothing = Nothing
    fmap f (Just a) = Just (f a)
Type Classes in Coq

Syntax of Class and Instance declarations

**Class** \( \text{Id} (\alpha_1:\tau_1) \cdots (\alpha_n:\tau_n) [:: \text{sort}] := \{ \)

\[
\begin{align*}
&f_1 &: \text{type}_{f_1}; \\
&\vdots \\
&f_m &: \text{type}_{f_m}.
\end{align*}
\]

**Instance** \( \text{ident}:\text{Id} \ term_1 \cdots \ term_n := \{ \)

\[
\begin{align*}
&f_1 &: \text{term}_{f_1}; \\
&\vdots \\
&f_m &: \text{term}_{f_m}.
\end{align*}
\]

Where \( \alpha_i:\tau_i \) are called parameters of the class and \( f_k:\text{type}_k \) are called the methods.

---

Type Classes in Coq

Example of Class and Instance declarations

```coq
Class EqDec (A : Type) := {
  eqb : A → A → bool ;
  eqb_prop:
    ∀ x y, eqb x y = true ⇒ x = y}.

Instance eq_bool : EqDec bool := {
  eqb x y := if x then y else negb y}.

Proof.
intros x y H.
destruct x ; destruct y ;
discriminate || reflexivity.
Qed.
```
Type Classes in Coq

Using Type Classes

Binding classes

**Definition** neqb {A} {eqa : EqDec A} (x y : A) := negb (eqb x y).

Superclasses

```coq
class (Eq a) => Ord a where
  le :: a -> a -> Bool
```

**Class** Ord A {E : EqDec A} := {
  le : A → A → bool}.  

Type Classes in CoQ

Using Type Classes

Substructures

**Definition** neqb {A} {eqa : EqDec A} (x y : A) ::= negb (eqb x y).

Superclasses

```
class (Eq a) => Ord a where
  le :: a -> a -> Bool
```

```
Class Ord A {E : EqDec A} ::= {
  le : A → A → bool
}.```