Logic and specification in Coq

Yves Bertot

October 2018

Starting from tests

- Tests rely on two components
  - A piece of code to generate test cases
  - A piece of code to verify correct behavior on all cases
- In our approach, we use logic
  - to describe what are all possible inputs
  - to express what is the correct behavior

Example on the filter function

- What do you expect from the filter function?

Filter function specification

- The output of the filter function must only contain values satisfying the boolean property
- All values satisfying the boolean property should be taken

Filter function specification

- The output of the filter function must only contain values satisfying the boolean property
- All values satisfying the boolean property should be taken
- The multiplicity of values is preserved
- The order of values is preserved

Expressing logical property

- Difficulty of Coq: logical values are not boolean
- In a sense, `bool` is restricted to logical statements that can be decided
  - This is sometimes circumvented by axioms
- A new type `Prop` is used for logical propositions
- For `T` a type and `a b : T`, `a = b : Prop`
- `Prop` also has connectives `and` (`/\`), `or` (`\/`), `not` (`~`), implication is written `->`
- Universal quantification is written `forall x : T, P x`
- Existential quantification `exists x : T, P x`
- A boolean value v can be mapped to a proposition by writing v `= true`

Propositions for lists and numbers

- `In x l` is defined by a recursive computation that yields a proposition based on = and \/
- Numbers have <=, <

```
Compute In (2 + 3) (3 :: 5 :: 8 :: nil).
= 3 = 5 \/ 5 = 5 \/ 8 = 5 \/ False : Prop
```

Specifications

```
Definition multiple_of div n :=
   exists k, n = k * div.

Definition prime n :=
  1 < n /\
  forall x, 1 < x < n -> ~multiple_of x n.

forall (A : Type) (f : A -> bool) (x : A) (l : list A),
  In x (filter f l) <-> In x l /\ f x = true
```

specifications based on tests

- Write your function: `f : A -> B`
- Write extra functions to verify that the output is correct, `verif : B -> bool`
- Express a universal statement `forall x :A, verif (f x) = true`
- Being able to prove such a statement is equivalent to exhaustive testing.

Performing proofs

- Interactive loop
- Start with `Lemma` *name* `:` *statement*`.` `Proof.`
- Then apply commands that decompose the goal.
- `intros`, `split`, `exists`, `left`, `right`, `auto`, `rewrite`.
- use `apply` with existing theorems.
- Special case for `In`: `simpl` or `compute`
- When finished, save using `Qed.`

Example on slide

```
Lemma exinlist : In 3 (2 :: 3 :: 5 :: 8 :: nil).
Proof.
compute.
1 subgoal

  ============================
  2 = 3 \/ 3 = 3 \/ 5 = 3 \/ 8 = 3 \/ False
right.
  3 = 3 \/ 5 = 3 \/ 8 = 3 \/ False
left.
 3 = 3
auto.
No more subgoals.
Qed.
```

Difficulties with logical reasoning

- Beginners have troubles with making the difference between various combinations of conjunction ("and") and implication
- Reasoning about negation and false premises is especially difficult
- The logic of Coq is "intuitionistic": proofs most often have to be constructive

Using libraries

- Coq comes with existing libraries
- Existing functions have theorems about them
- Important command : `Search`

```
Search filter.
filter_In:
  forall (A : Type) (f : A -> bool) (x : A) (l : list A),
  In x (filter f l) <-> In x l /\ f x = true
```

Examples of powerful libraries

- Mathematical Components
  - Designed for the 4-color theorem, the odd order theorem (Feit-Thompson)
  - Used for elliptic curves, reasoning about robots, combinatorics, transcendance proofs
- Coquelicot
  - Real analysis: limits, derivatives, integrals, mathematical functions
  - Used for numerical computations (wave function, mathematical constants)
- VST
  - Reasoning about programs in C (in connection with Compcert)
  - Used for pointer data structures, security proofs