# Lab Course: The Coq Proof Assistant

Luminous Fennell

2015-04-30

# Overview

## Software Foundations (Benjamin Pierce et al.)

- Self study course
- Chapters: Commented source code with exercises
- `http://www.cis.upenn.edu/~bcpierce/sf/`

## Lab

- Work the chapters at home
- Meeting every other week for questions/discussion
- Exercises of chapters may be submitted; review on request
- Course Homepage:
  `http://proglang.informatik.uni-freiburg.de/teaching/coq-practicum/2014`

# Graded Exercises

- 4 graded exercises, distributed throughout the semester
- Each 25% of final grade
- Timetable $\rightarrow$ course homepage

# Contact

Departement of Programming Languages
Building 079, Rooms 00-013 and 00-014

- Prof. Dr. Peter Thiemann
- Luminous Fennell:
  `fennell@informatik.uni-freiburg.de`

# Coq

http://coq.inria.fr/

```
File Edit Options Buffers Tools Coq Proof-General Holes Help

Tactic Notation "aevalR_cases" tactic(first) ident(c) :=
  first;
  [ Case_aux c "E_ANum" | Case_aux c "E_APlus"
  | Case_aux c "E_AMinus" | Case_aux c "E_AMult" ].

(** It is straightforward to prove that the relational and functional
    definitions of evaluation agree on all possible arithmetic
    expressions... *)



Theorem aeval_iff_aevalR : forall a n,
  (a || n) <-> aeval a = n.
Proof.
 split.
 Case "->".
  intros H.
  aevalR_cases (induction H) SCase; simpl.
  SCase "E_ANum".
    reflexivity.
  SCase "E_APlus".
    rewrite IHaevalR1.  rewrite IHaevalR2.  reflexivity.
  SCase "E_AMinus".
    rewrite IHaevalR1.  rewrite IHaevalR2.  reflexivity.
  SCase "E_AMult".
    rewrite IHaevalR1.  rewrite IHaevalR2.  reflexivity.
 Case "<-".
  generalize dependent n.
  aexp_cases (induction a) SCase;
     simpl; intros; subst.
--:---  Imp.v         35% (685,29)  <N>  Git-master  (Coq Script(4) Holes)--1:55PM------
  H0 : e2 || n2
  IHaevalR1 : aeval e1 = n1
  IHaevalR2 : aeval e2 = n2
  ============================
   n1 + aeval e2 = n1 + n2

subgoal 2 is:
 aeval e1 - aeval e2 = n1 - n2
-U:%%-  *goals*       36% (14,0)    <N>    (Coq Goals Undo-Tree)--1:55PM------

-U:%%-  *response*    All (1,0)     <N>    (Coq Response Undo-Tree)--1:55PM------
```

# Stating and Proving formal theorems

## Informal

"Clearly, zero is the smallest natural number!"

## Formal (Coq)

```
Inductive nat : Set :=
| O : nat
| S : nat -> nat.

Inductive le : nat -> nat -> Prop :=
| le_n : forall n : nat, le n n
| le_S : forall n1 n2 : nat,
         le n1 n2 -> le n1 (S n2).
```

# Stating and Proving formal theorems

## Informal

"Clearly, zero is the smallest natural number!"

## Formal (Coq)

```
Theorem le_nat_total: forall n : nat, le 0 n.
Proof. intros n. induction n as [| n'].
(* Case n = 0 *)
apply le_n.
(* Case n = S n' *)
apply le_S. apply IHn'.
Qed.

(* Or with automation *)
Theorem le_nat_total: forall n : nat, le 0 n.
Proof. intros n; induction n as [| n']; auto.
Qed.
```

# Formalization of Programming Languages

## While Programs

$$
\begin{aligned}
e &::= k \mid \texttt{True} \mid \texttt{False} \mid x \mid e + e \mid e - e \\
s &::= x := e \mid s; s \mid \texttt{IF}\ e\ \texttt{THEN}\ s\ \texttt{ELSE}\ s \mid \texttt{WHILE}\ e\ \texttt{DO}\ s
\end{aligned}
$$

## Lambda Calculus

$$
\begin{aligned}
e ::= {} & k \mid \texttt{True} \mid \texttt{False} \mid x \mid \texttt{IF}\ e\ \texttt{THEN}\ e\ \texttt{ELSE}\ e \\
& \mid \lambda x.\ e \mid e\ e
\end{aligned}
$$

## Meta-Theory and Verification

- Precise definition of semantics
- Type systems
- Proving properties about programs (e.g. Correctness)