Software Foundations
Introduction

Albert-Ludwigs-Universität Freiburg

Luminous Fennell

2012-10-24

Upfront Notes

- Which semester?
- Experience:
  - Logic courses, Th. comp. science
  - Verification, Hoare Calculus
  - Functional Programming
  - Formal Systems
- Coq:
  - Proof Assistant
  - Programming language
  - (show live)

Modus Operandi

Software Foundations (Benjamin Pierce et al.)

- Self study course
- Chapters: Commented source code with exercises
- `http://www.cis.upenn.edu/~bcpierce/sf/`
  Version 2012-7-25

- Work the chapters at home
- Meeting once a week for questions/discussion
- Exercises may be submitted
- Course Homepage:
  http://proglang.informatik.uni-freiburg.de/teaching/softwarefoundations/2012ws/

Exercises

- Chapter Exercises
  - Edited versions on course website

    `(* EXPECTED *)` Exercise is **strongly** recommended

    `(* NO SOLUTION *)` Solution on demand
  - Sample solution 1-2 weeks later
- Graded Exercises
  - **4 graded exercises**, distributed throughout the semester (2 before, 2 after christmas)
  - Each 25% of final grade
  - 2 weeks time to submit

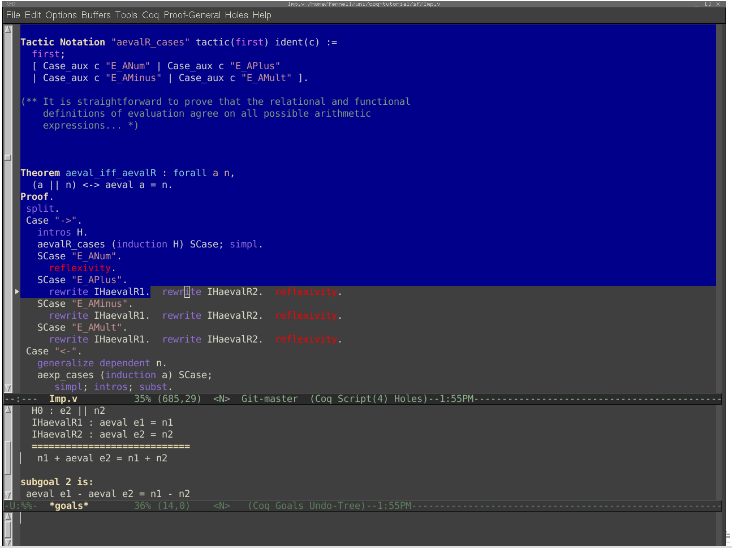
Contact

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

- Prof. Dr. Peter Thiemann
- Luminous Fennell:
  `fennell@informatik.uni-freiburg.de`
- (Manuel Geffken, Robert Jakob, Matthias Keil)

Coq

http://coq.inria.fr/



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).
```

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

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

Formalization of Programming Languages

While Programs

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

Lambda Calculus

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

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

Implementation of Certified Functional Programs

. . . if you're interested

```
Define sortedlist := { l : natlist | sorted l }.
insertSorted : nat -> sortedlist -> sortedlist := ...

myFancySort : forall l : natlist,
              { l' : natlist | l' = simpleSort l } := ...
```