The Coq Proof Assistant
Introduction

Albert-Ludwigs-Universität Freiburg

2014-05-08
Which semester?

Experience:
- Logic courses, Th. comp. science
- Verification, Hoare Calculus
- Functional Programming
- Formal Systems

Coq:
- Proof Assistant
- Programming language
- (show live)
Software Foundations (Benjamin Pierce et al.)

- Self study course
- Chapters: Commented source code with exercises
- [http://www.cis.upenn.edu/~bcpierce/sf/](http://www.cis.upenn.edu/~bcpierce/sf/)
  Version 2013-07-18

- 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/coq-practicum/2014](http://proglang.informatik.uni-freiburg.de/teaching/coq-practicum/2014)
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
  - Each 25% of final grade
  - 2 weeks time to submit
Departement of Programming Languages
Building 079, Rooms 00-013 and 00-014

- Prof. Dr. Peter Thiemann
- Luminous Fennell: fennell@informatik.uni-freiburg.de
http://coq.inria.fr/

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

goals* 36% (14,0) -N- (Coq Goals Undo-Tree)--1:55PM
Informal

“Clearly, zero is the smallest natural number!”

Formal (Coq)

Inductive nat : Set :=
| 0 : 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 (S n1) n2.

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

\[ e ::= k \mid \text{True} \mid \text{False} \mid x \mid e + e \mid e - e \]
\[ s ::= x := e \mid s; s \mid \text{IF } e \text{ THEN } s \text{ ELSE } s \mid \text{WHILE } e \text{ DO } s \]

Lambda Calculus

\[ e ::= k \mid \text{True} \mid \text{False} \mid x \mid \text{IF } e \text{ THEN } e \text{ ELSE } e \]
\[ \mid \lambda x. e \mid e \ e \]

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