Neformalna verifikacija

Bojan Petrović
bojan_petrovic@fastmail.fm

Haklab Beograd

April 2015

Haklab Beograd

Software Foundations

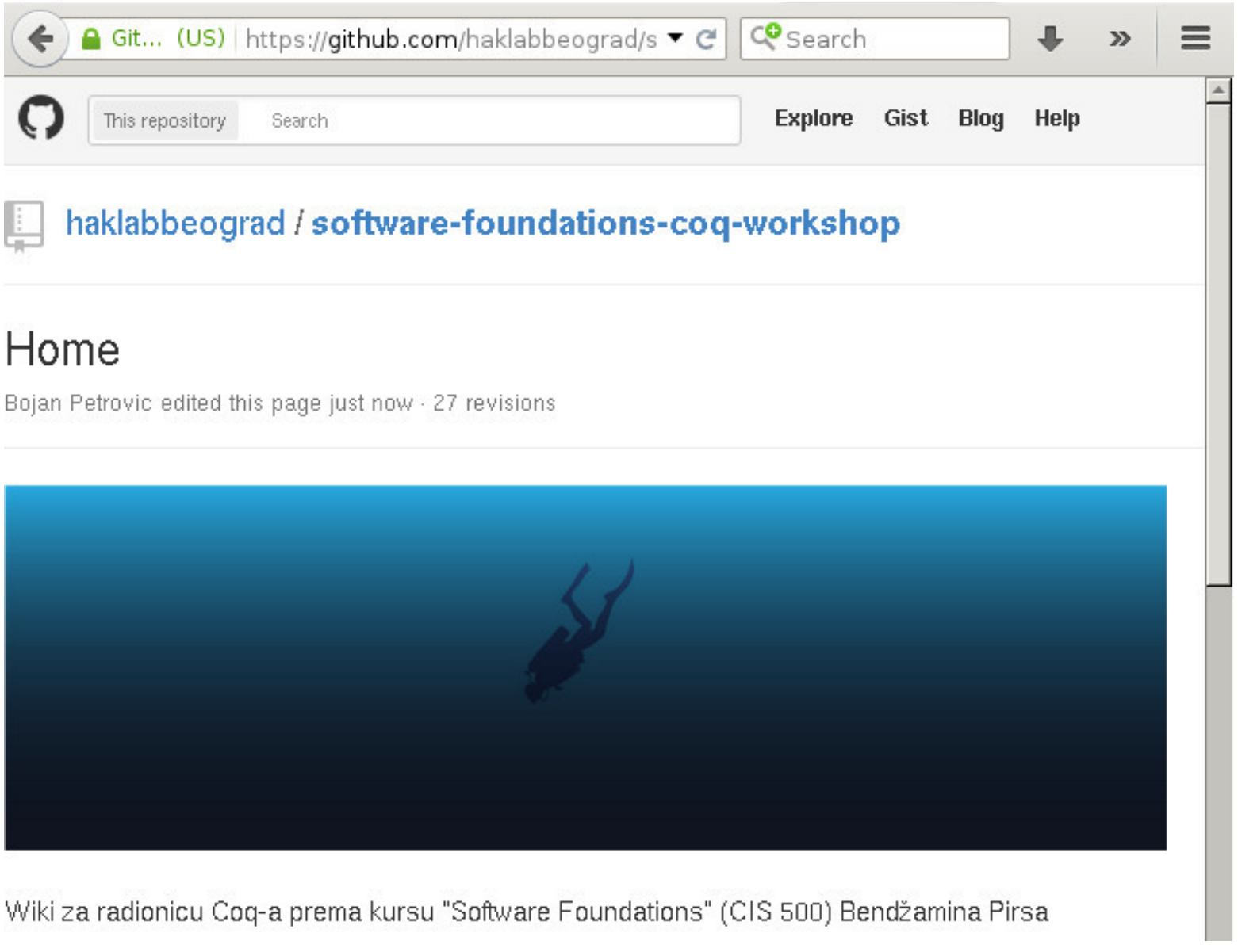
Software Foundations

Glavni tvorac kursa je Bendžamin Pirs sa Univerziteta u Pensilvaniji

Pokrivene oblasti

- Funkcionalno programiranje
- Formalna logika
- Formalna verifikacija softvera
- Operaciona sematika programskih jezika
- Sistemi tipova
- Coq

Software Foundations u Haklabu



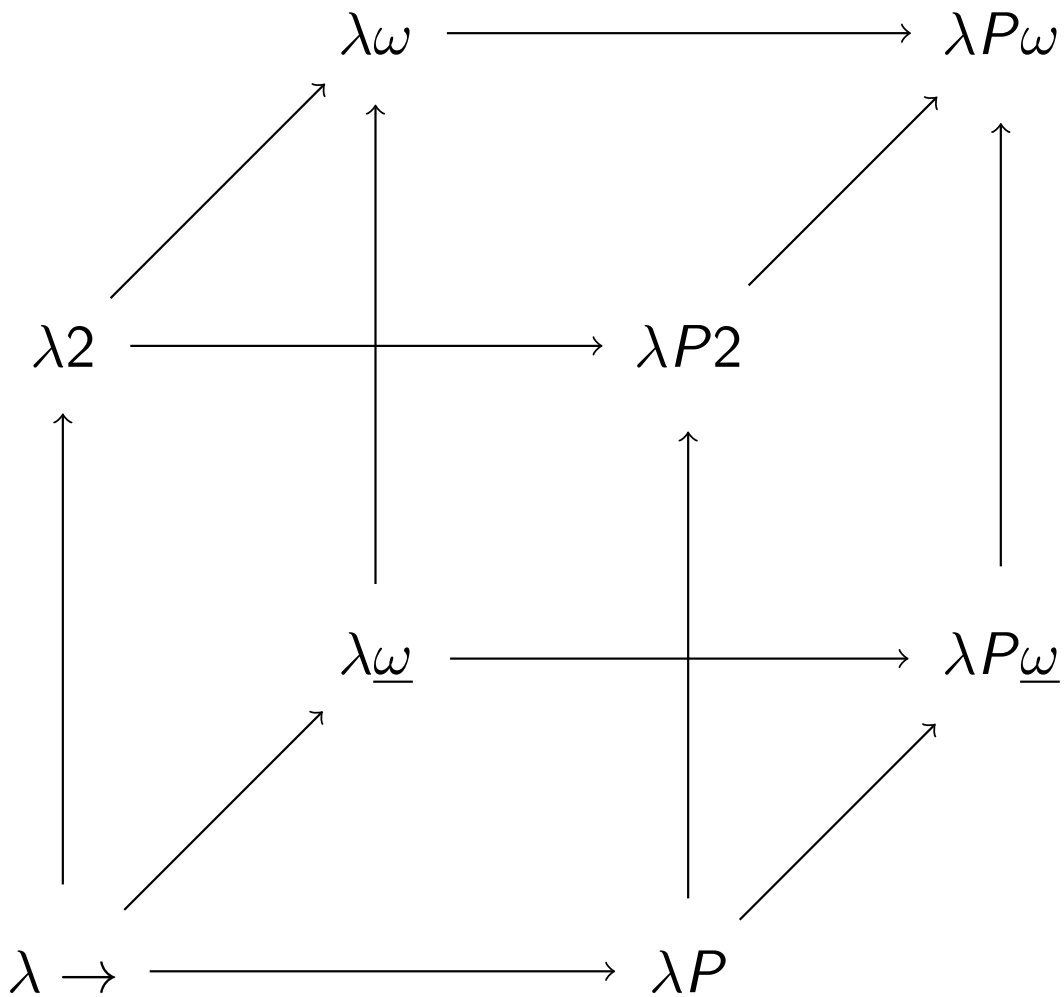
Pedagoški pristup

Grøn, Øyvind; Næss, Arne (1998). *Introduction to general relativity and its mathematics*. HiO-rapport 1998 14. Oslo: Høgskolen i Oslo.

Coq

- Coq je programski jezik i interaktivni dokazivač teorema
- Coq nije Tjuring potpun
- Coq je zasnovan na varijanti lambda računa sa zavisnim tipovima nazvanoj "račun konstrukcija" (Calculus of Constructions)
- Neke od uspešnih primena: CompCert; dokaz teoreme o četiri boje

Barendregtova λ-kocka



man 3 printf

```
PRINTF(3)     Linux Programmer's Manual     PRINTF(3)

NAME
       printf, fprintf, sprintf, snprintf, vprintf,
       vfprintf, vsprintf, vsnprintf - formatted out-
       put conversion

SYNOPSIS
       #include <stdio.h>

       int printf(const char *format, ...);
```

Vektori

```
Check @append.
append: forall (A : Type) (n p : nat),
  t A n -> t A p -> t A (n + p)
```

Ekstrinsična verifikacija

```
Check sort.
sort : list nat -> list nat

Theorem Sorted_sort : forall l, Sorted (sort l).
Theorem Permuted_sort : forall l, Permutation l (sort l).
```

Intrinsična verifikacija

```
Check mergesort.
mergesort
   : forall (A: Type) (l: List A),
     { l' : list A |
     Sorted l' /\ Permutation l' l /\ Stable l' l }
```

Neka od poglavlja

- Functional Programming in Coq
- Logic in Coq
- Simple Imperative Programs
- Program Equivalence
- Hoare Logic
- Hoare Logic as a Logic
- The Simply Typed Lambda-Calculus
- Typing Mutable References
- Subtyping
- Normalization of STLC

Infrastruktura: brojevi

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

Infrastruktura: logika

```
Inductive and (P Q : Prop) : Prop :=
  conj : P -> Q -> and P Q.

Inductive or (P Q : Prop) : Prop :=
 | or_introl : P -> or P Q
 | or_intror : Q -> or P Q.

Inductive False : Prop := .

Inductive ex (X:Type) (P : X->Prop) : Prop :=
   ex_intro : forall (witness:X), P witness -> ex X P.
```

Imp

```
c ::= SKIP
    | x ::= a
    | c ;; c
    | WHILE b DO c END
    | IFB b THEN c ELSE c FI
```

Imp: operaciona semantika (deo)

$$\frac{}{\texttt{SKIP}/st \Downarrow st}\ (Skip)$$

$$\frac{aeval\ st\ a_1 = n}{X\ :=\ a_1/st \Downarrow update\ st\ X\ n}\ (Assign)$$

$$\frac{c_1/st \Downarrow st' \quad c_2/st' \Downarrow st''}{c_1\ ;;\ c_2/st \Downarrow st''}\ (Sequence)$$

$$\vdots$$

Horova logika: Notacija

```
Notation "{{ P }} c {{ Q }}" :=
  (hoare_triple P c Q) (at level 90, c at next level)
  : hoare_spec_scope.

Notation "P [ X |-> a ]" :=
  (assn_sub X a P) (at level 10).

Theorem hoare_asgn : forall Q X a,
  {{Q [X |-> a]}} (X ::= a) {{Q}}.
```

Imp/Imp+BREAK

```
Inductive ceval : com -> state -> state -> Prop :=
  | E_Skip :
      forall (st : state),
      SKIP / st || st
  | E_Ass :
      forall (st : state) (a : aexp) (n : nat) (X : id),
      aeval st a = n ->
      (X ::= a) / st || (update st X n)
  | E_Seq :
      forall (c1 c2 : com) (st st' st'' : state),
      (c1 / st || st') ->
      (c2 / st' || st'') -> (c1 ; c2) / st || st''
  | E_IfTrue :
      forall (st st': state) (c1 c2 : com) (b : bexp),
      beval st b = true -> c1 / st || st' ->
      (IFB b THEN c1 ELSE c2 FI) / st || st'
  | E_IfFalse :
      forall (st st': state) (c1 c2 : com) (b : bexp),
      beval st b = false -> c2 / st || st' ->
      (IFB b THEN c1 ELSE c2 FI) / st || st'
  | E_WhileEnd :
      forall (st : state) (c1 : com) (b : bexp),
      beval st b = false ->
      (WHILE b DO c1 END) / st || st
  | E_WhileLoop :
      forall (st st' st'': state) (c1 : com) (b : bexp),
      beval st b = true ->
      c1 / st || st' ->
      (WHILE b DO c1 END) / st' || st'' ->
      (WHILE b DO c1 END) / st || st''
  where "c1 '/' st '||' st'" := (ceval c1 st st').
```

```
Inductive ceval : com -> state -> status -> state -> Prop :=
  | E_Skip :
      forall (st : state),
      SKIP / st || SContinue / st
  | E_Break :
      forall (st : state),
      BREAK / st || SBreak / st
  | E_Ass :
      forall (st : state) (a : aexp) (n : nat) (X : id),
      aeval st a = n ->
      (X ::= a) / st || SContinue / (update st X n)
  | E_Seq_SBreak :
      forall (c1 c2 : com) (st st' : state),
      (c1 / st || SBreak / st') -> (c1 ; c2) / st || SBreak / st'
  | E_Seq_SCont :
      forall (c1 c2 : com) (st st' st'': state) (s : status),
      (c1 / st || SContinue / st') ->
      (c2 / st' || s / st'') -> (c1 ; c2) / st || s / st''
  | E_IfTrue :
      forall (st st': state) (s : status) (c1 c2 : com) (b : bexp),
      beval st b = true -> c1 / st || s / st' ->
      (IFB b THEN c1 ELSE c2 FI) / st || s / st'
  | E_IfFalse :
      forall (st st': state) (s : status) (c1 c2 : com) (b : bexp),
      beval st b = false -> c2 / st || s / st' ->
      (IFB b THEN c1 ELSE c2 FI) / st || s / st'
  | E_WhileEnd :
      forall (st : state) (c1 : com) (b : bexp),
      beval st b = false ->
      (WHILE b DO c1 END) / st || SContinue / st
  | E_WhileLoop_SBreak :
      forall (st st' : state) (c1 : com) (b : bexp),
      beval st b = true ->
      c1 / st || SBreak / st' ->
      (WHILE b DO c1 END) / st || SContinue / st'
  | E_WhileLoop_SCont :
      forall (st st' st'': state) (s:status) (c1 : com) (b : bexp),
      beval st b = true ->
      c1 / st || SContinue / st' ->
      (WHILE b DO c1 END) / st' || SContinue / st'' ->
      (WHILE b DO c1 END) / st || SContinue / st''
  where "c1 '/' st '||' s / st'" := (ceval c1 st s st').
```

Ispravnost kompajlera aritmetičkih izraza

```
Theorem s_compile_correct :
   forall (e : aexp) (st : state),
   s_execute st [] (s_compile e) = [aeval st e].
```

Neki negativni aspekti

- Nedostatak individualnog angažmana
- “Efikasnost” Coq-a