Coq によるプログラムの検証入門

集中講義 千葉大学大学院

アフェルト レナルド

産業技術総合研究所

2017 年 01 月 25 日

目 次

1　依存型を用いた自然数の引き算

授業の pred 関数のように, 引き算を実装する. 具体的に, m と n は自然数とする. その場合, $m - n$ は $m \geq n$ が成り立つときにしか定義されていない.

自然数の引き算を行う完全な関数の例:

```
Fixpoint tminus (n m : nat) : nat :=
  match n with
  | O ⇒ O
  | S n' ⇒ match m with
              | O ⇒ n
              | S m' ⇒ tminus n' m'
              end
  end.
```

```
Compute tminus 5 3.
Compute tminus 5 6.
```

次の型を持つ引き算関数を実装する:

```
Require Import Arith.
```

```
Fixpoint pminus (n m : nat) : m ≤ n → nat.
Abort.
```

提案した関数をテストする.

```
Lemma three_le_five : 3 ≤ 5.
Proof.
```

```
auto.
Qed.

Compute pminus 5 3 three_le_five.
```

次の型を持つ引き算の関数を実装する. ただし, 対話的な方法を使う. (時間があれば, 直接に記述してみる.)

```
Fixpoint iminus (n m : nat) : m ≤ n → { k : nat | k + m = n }.
Abort.
```

2 ホーア論理を用いて, 階乗のプログラムを検証する

授業で紹介したホーア論理を用いて, 階乗のプログラムを検証する. 仕様として, Coq の Factorial 標準ライブラリの fact 関数を利用する.

授業で見たホーア論理を用いて, 次の証明を完成させる.

```
Require Import Omega.
Require Import Factorial.

Lemma facto_fact x X ret : x ≠ ret →
  hoare
    (fun s ⇒ eval (exp_var x) s = X ∧
              eval (exp_var ret) s = 1)
    (facto x ret)
    (fun s ⇒ eval (exp_var ret) s = fact X).
Proof.
intros xret.
set (P' := fun s : state ⇒
  eval (exp_var ret) s × fact (eval (exp_var x) s) = fact X
).
set (Q' := fun s : state ⇒ eval (exp_var ret) s ×
  fact (eval (exp_var x) s) = fact X ∧
   ¬ (beval (neg (equa (exp_var x) (cst O))) s)).
apply (hoare_conseq P' Q').
- unfold imp.
  intros s.
  unfold P'.
  simpl.
  destruct 1.
  rewrite H.
  rewrite H0.
  omega.
- unfold imp, Q'.
  intros s H.
  destruct H as [H1 H2].
```

```
  rewrite ← H1.
  assert (H : (eval (exp_var x) s) = 0).
    simpl in H2.
    simpl.
    omega.
  rewrite H.
  simpl.
  omega.
- apply hoare_while.
  set (R := fun s : state ⇒ eval (exp_var ret) s ×
    fact (eval (exp_var x) s - 1) = fact X ∧
    0 ≤ eval (exp_var x) s - 1).
  apply (hoare_seq R).
+
+
```

```
set (P' := fun s : state ⇒ eval (exp_var ret) s ×
  fact (eval (exp_var x) s) = fact X).
set (Q' := fun s : state ⇒ eval (exp_var ret) s ×
  fact (eval (exp_var x) s) = fact X ∧
   ¬ (beval (neg (equa (exp_var x) (cst O))) s)).
apply (hoare_conseq P' Q').
```