Semantics, WS 2011-2012: Solution for Assignment 6

Prof. Dr. Gert Smolka, Dr. Chad Brown

Read the new version of Chapter 4 of the lecture notes.

Exercise 6.1 Prove the following goals once using inversion and a second time without using inversion. Do not use induction.

a) `Goal ~even 1.`

b) `Goal forall n, even (S (S n)) -> even n.`

Solution to Exercise 6.1

```
Goal ~even 1.

Proof. intros A. now inversion A. Qed.

Goal ~even 1.

Proof. intros A. remember 1 as n. destruct A; discriminate. Qed.

Goal forall n, even (S (S n)) -> even n.

Proof. intros n A. inversion A. assumption. Qed.

Goal forall n, even (S (S n)) -> even n.

Proof. intros n A. remember (S (S n)) as m. destruct A; congruence. Qed.
```

Exercise 6.2 Consider the inductive definition of *le* with one proper argument.

```
Inductive le (x:nat) : nat -> Prop :=
| lex : le x x
| leS : forall y, le x y -> le x (S y).
```

Prove the following by induction on *le*.

a) `Lemma le_Sright x y : le x y -> le (S x) (S y).`

b) `Goal forall x, le x 0 -> x = 0.`

Solution to Exercise 6.2

```
Lemma le_Sright x y : le x y -> le (S x) (S y).
intros A. induction A.
now apply lex.
now apply leS.
Qed.

Goal forall x, le x 0 -> x = 0.
Proof. intros x A. remember 0 as y. destruct A; congruence. Qed.
```

Exercise 6.3 Consider the inductive definition of *le'* with two proper arguments.

```
Inductive le' : nat -> nat -> Prop :=
| lex' : forall x, le' x x
| leS' : forall x y, le' x y -> le' x (S y).
```

Prove the following two versions of $Sx \not\leq 0$ formulated using *le* and *le'*.

```
Goal forall x, ~ le (S x) 0.

Goal forall x, ~ le' (S x) 0.
```

Solution to Exercise 6.3

```
Goal forall x, ~ le (S x) 0.
Proof. intros x A. remember 0 as y. destruct A; congruence. Qed.

Goal forall x, ~ le' (S x) 0.
Proof. intros x A. remember (S x) as x'. remember 0 as y. destruct A; congruence. Qed.
```

Exercise 6.4 Consider the following inductively defined proposition.

```
Inductive F : Prop :=
| FI : F -> F.
```

Prove the following goal.

```
Goal F -> False.
```

Make sure you understand the goal you need to prove at each stage of the proof.

Solution to Exercise 6.4

```
Goal F -> False.
Proof. intros A. induction A. now apply IHA. Qed.
```

After *induction A* the goal is

```
A : F
IHA : False
-----------
False
```

Exercise 6.5 Read the development of the abstract Imp language in the Coq file. Make sure you understand the definitions, theorems, and their proofs. Complete the proofs of *Seq_assoc*, *skip_div*, *monotone_while*, *optimization1*, *eval_monotone* and *eval_agrees_divergence*. **Note:** Two new tactics you may find helpful are *exfalso* and *case_eq*. *exfalso* strengthens the goal by changing the claim to *False*. This can be used when the current hypotheses are inconsistent. *case_eq t* can be used to replace the combination of tactics *remember t as x. destruct x.*

Solution to Exercise 6.5

```
Lemma Seq_assoc c1 c2 c3:
ceq (Seq c1 (Seq c2 c3)) (Seq (Seq c1 c2) c3).

Proof. split ; intros s s' A ; inv A.
inv H4. econstructor. now econstructor ; eauto. assumption.
inv H1. econstructor. eassumption. econstructor ; eauto. Qed.

Lemma skip_div : (exists s : state, True) -> ~ ceq skip div.

Proof. intros [s _] A. apply (div_term s). exists s.
apply A. constructor. Qed.

Lemma monotone_while t : monotone (While t).

Proof. intros c c' A s s' B.
remember (While t c) as r ; induction B ; inv Heqr.
apply semWhileFalse ; assumption.
eapply semWhileTrue ; eauto. Qed.
```

```
Theorem optimization1 (f : com -> com) :
(forall c, cap c (f c)) ->
 forall c, cap c (optimize f c).

Proof.
intros A c. induction c ; simpl ; intros s s' B ; apply A ; inv B.
now apply semAct.
now eapply semSeq ; eauto.
now apply semIfTrue ; auto.
now apply semIfFalse ; auto.
now apply semWhileFalse ; assumption.
eapply semWhileTrue ; eauto. revert H5. apply monotone_while ; auto. Qed.

Lemma eval_monotone m n c :
m<=n -> approx (eval m c) (eval n c).

Proof. intros A. induction c ; intros s s' ; simpl.
(* a *) auto.
(* ; *) case_eq (eval m c1 s) ; try congruence.
        intros. now erewrite IHc1 ; eauto.
(* if *) now case_eq (t s) ; auto.
(* while *) apply evalw_monotone ; auto. Qed.

Corollary eval_agrees_divergence c s :
~ terminates c s <-> forall n, eval n c s = None.

Proof. split.
intros A n. case_eq (eval n c s) ; auto. intros s' B. exfalso.
   apply A. exists s'. apply eval_agrees. now firstorder.
intros A [s' B]. apply eval_agrees in B. destruct B as [n C].
assert (eval n c s = None) by auto.
congruence. Qed.
```