Semantics, WS 2011-2012: Solution for Assignment 5

Prof. Dr. Gert Smolka, Dr. Chad Brown

Remark: You may use any of the tactics we used in class including *econstructor*, *congruence*, *firstorder* and *auto*. In addition, the tactic *eassumption* is helpful when the claim has an evar, but otherwise matches an assumption.

Exercise 5.1 Formulate the following equivalences as goals in Coq and prove them.

a) $c;\mathsf{skip} \;\cong\; c$

b) $\mathsf{if\,false\,then}\; c_1 \;\mathsf{else}\; c_2 \cong c_2$

c) $\mathsf{while\,false\,do}\; c \;\cong\; \mathsf{skip}$

d) $\mathsf{while}\; b \;\mathsf{do}\; c \;\cong\; \mathsf{if}\; b \;\mathsf{then}\; c;\mathsf{while}\; b \;\mathsf{do}\; c \;\mathsf{else\,skip}$

Solution to Exercise 5.1

```
Goal forall c,
cequiv (c ; SKIP) c.

Proof. split ; intros st st' A.
inv A. inv H4. assumption.
econstructor. now apply A. now constructor. Qed.

Goal forall c1 c2 b,
(forall st, beval st b = false) ->
cequiv (IFB b THEN c1 ELSE c2 FI) c2.

Proof. intros c1 c2 b A. split ; intros st st' B.
inv B. generalize (A st). congruence. assumption.
apply E_IfFalse ; now auto. Qed.

Goal forall b c,
(forall st, beval st b = false) ->
cequiv (WHILE b DO c END) SKIP.

Proof. intros b c A. split ; intros st st' B.
inv B. now constructor. generalize (A st). congruence.
inv B. apply E_WhileEnd. auto. Qed.
```

```
Goal forall b c,
cequiv (WHILE b DO c END) (IFB b THEN (c ; WHILE b DO c END) ELSE SKIP FI).

Proof. split ; intros st st' A.
inv A.
   apply E_IfFalse. assumption. now constructor.
   apply E_IfTrue. assumption. econstructor ; eassumption.
inv A.
   inv H5. eapply E_WhileLoop ; eassumption.
   inv H5. apply E_WhileEnd ; assumption. Qed.
```

Exercise 5.2 Use Coq to prove that the approximation relation $\lesssim$ is reflexive and transitive.

Solution to Exercise 5.2

```
Goal forall c,
cimpl c c.

Proof. firstorder. Qed.

Goal forall c1 c2 c3,
cimpl c1 c2 -> cimpl c2 c3 -> cimpl c1 c3.

Proof. firstorder. Qed.
```

Exercise 5.3 Use Coq to prove that program equivalence $\cong$ is reflexive, symmetric and transitive.

Solution to Exercise 5.3

```
Goal forall c,
cequiv c c.

Proof. firstorder. Qed.

Goal forall c1 c2,
cequiv c1 c2 -> cequiv c2 c1.

Proof. firstorder. Qed.

Goal forall c1 c2 c3,
cequiv c1 c2 -> cequiv c2 c3 -> cequiv c1 c3.

Proof. firstorder. Qed.
```

Exercise 5.4 Use Coq to prove that if $c_1 \lesssim c_1'$ and $c_2 \lesssim c_2'$, then $c_1; c_2 \lesssim c_1'; c_2'$.

Solution to Exercise 5.4

```
Goal forall c1 c1' c2 c2',
cimpl c1 c1' -> cimpl c2 c2' -> cimpl (c1;c2) (c1';c2').

Proof. intros c1 c1' c2 c2' A B st st' C. inv C.
apply E_Seq with (st':=st'0) ; auto. Qed.
```

Exercise 5.5 Use Coq to prove that if $c_1 \lesssim c_1'$ and $c_2 \lesssim c_2'$, then $\text{if } b \text{ then } c_1 \text{ else } c_2 \lesssim \text{if } b \text{ then } c_1' \text{ else } c_2'$.

Solution to Exercise 5.5

```
Goal forall b c1 c1' c2 c2',
cimpl c1 c1' -> cimpl c2 c2' ->
cimpl (IFB b THEN c1 ELSE c2 FI) (IFB b THEN c1' ELSE c2' FI).

Proof. intros b c1 c1' c2 c2' A B st st' D. inv D.
apply E_IfTrue ; auto.
apply E_IfFalse ; auto. Qed.
```

Exercise 5.6 Assume we know the relational semantics is functional.

```
Lemma ceval_functional c st st1 st2 :
c / st || st1 -> c / st || st2 -> st1 = st2.
```

a) Prove if $\text{skip} \lesssim c$, then $\text{skip} \cong c$.

b) Prove if $c \lesssim c'$ and c terminates on all states, then $c \cong c'$.

Solution to Exercise 5.6

a)

```
Goal forall c,
cimpl SKIP c -> cequiv SKIP c.

Proof. intros c A. split.
assumption.
intros st st' B. rewrite (ceval_functional c st st' st B).
constructor.
apply A. constructor.
Qed.
```

```
(*** Here are alternative proofs. ***)
Goal forall c, cimpl SKIP c -> cequiv SKIP c.

Proof. intros c A. split.
assumption.
intros st st' B.
cut (st' = st). intros C. subst. constructor.
eapply ceval_functional; eauto. apply A. constructor.
Qed.

Goal forall c, cimpl SKIP c -> cequiv SKIP c.

Proof. intros c A. split.
assumption.
intros st st' B.
assert (SKIP / st || st) by constructor.
assert (st' = st) by (eapply ceval_functional; eauto).
subst. constructor.
Qed.
```

b)

```
Goal forall c c',
cimpl c c' -> (forall st, terminates c st) -> cequiv c c'.
intros c c' A B. split.
assumption.
intros st st' D.
destruct (B st) as [st'' E].
rewrite (ceval_functional c' st st' st'').
assumption.
assumption.
apply A. assumption.
Qed.

(*** Here is an alternative proof. ***)
Goal forall c c',
cimpl c c' -> (forall st, terminates c st) -> cequiv c c'.
intros c c' A B. split.
assumption.
intros st st' D.
destruct (B st) as [st'' E].
assert (st' = st'') by (eapply ceval_functional; eauto).
rewrite H.
exact E.
Qed.
```

Exercise 5.7 Assume we have a type of states, an abstract boolean predicate b on states and an abstract function c on states.

```
Variable state : Type.
Variable b : state -> bool.
Variable c : state -> state.
```

Suppose we define a relation *rel* on states by the following two rules.

$$\frac{b\sigma = \mathit{false}}{\mathit{rel}\ \sigma\ \sigma} \qquad\qquad \frac{b\sigma = \mathit{true} \qquad \mathit{rel}\ (c\ \sigma)\ \sigma'}{\mathit{rel}\ \sigma\ \sigma'}$$

Remark: You should be able to do part (a) of this problem with no trouble. Parts (b) - (d) are more challenging. To do a case analysis on the result of a non-variable term t you may write

```
remember t as x. destruct x.
```

instead of

```
destruct t.
```

a) Define a step function *step:nat −> state −> option state* so that the proposition

   ```
   forall s s', rel s s' <-> exists i, step i s = Some s'.
   ```

   will be provable.

b) Prove

   ```
   Lemma agree s s' :
   rel s s' <-> exists i, step i s = Some s'.
   ```

c) Prove

   ```
   Lemma monotone i s s' :
   step i s = Some s' -> step (S i) s = Some s'.
   ```

d) Prove

   ```
   Lemma functional s s' s'' :
   rel s s' -> rel s s'' -> s'=s''.
   ```

Solution to Exercise 5.7

```
Fixpoint step (i : nat) (s : state) : option state :=
match i with O => None
| S i' => if b s then step i' (c s) else Some s
end.
```

```
Lemma agree s s' :
rel s s' <-> exists i, step i s = Some s'.

Proof. split.
intros A. induction A.
  exists 1. simpl. rewrite H. reflexivity.
  destruct IHA as [i B]. exists (S i). simpl. rewrite H. assumption.
intros [i A]. revert s A. induction i ; intros s A.
  now inversion A. simpl in A.
  remember (b s) as rb. destruct rb.
    apply rel1 ; now auto.
    inversion A ; subst. apply rel0 ; now auto.
Qed.

Lemma monotone i s s' :
step i s = Some s' -> step (S i) s = Some s'.

Proof. revert s s'. induction i ; simpl ; intros s s' A.
now inversion A.
remember (b s) as rb. destruct rb.
  change (step (S i) (c s) = Some s'). now auto.
  assumption.
Qed.

Lemma functional s s' s'' :
rel s s' -> rel s s'' -> s'=s''.

Proof. intros A B. induction A ; inversion B ; auto ; congruence. Qed.
```