Semantics, WS 2011-2012: Solution for Assignment 2

Prof. Dr. Gert Smolka, Dr. Chad Brown

Note: The test for this assignment will be given in the first 15 minutes of the lecture on Wednesday (since Tuesday is a holiday).

Use Coq's predefined types and functions for booleans, naturals, pairs, lists and options. Predefined objects can be inspected with the command Print. Predefined notation can be inspected with the command Locate. Here are two examples:

```
Locate "*".
Print prod.
```

Exercise 2.1 Define a function that swaps the components of pairs and prove $swap(swap\ p) = p$ for all pairs p.

Solution to Exercise 2.1

```
Definition swap {X Y : Type} (p : prod X Y) : prod Y X :=
match p with (x,y) => (y,x) end.

Goal forall (X Y : Type) (p : X * Y), swap (swap p) = p.
intros X Y p. destruct p. simpl. reflexivity. Qed.
```

Exercise 2.2 Prove $x * y = iter\ x\ (plus\ y)\ O$ for all numbers x and y.

Solution to Exercise 2.2

```
Goal forall m n : nat, m*n = iter m (plus n) 0.
intros m n. induction m ; simpl. reflexivity.
rewrite IHm. reflexivity. Qed.
```

Exercise 2.3 Define an exponentiation function *power* and prove $power\ x\ n = iter\ n\ (mult\ x)\ (S\ O)$ for all $x, n : nat$.

Solution to Exercise 2.3

```
Fixpoint power (x n : nat) : nat :=
match n with
| O => 1
```

```
| S n' => x * power x n'
end.

Goal forall x n : nat,
power x n = iter n (mult x) 1.
intros x n. induction n ; simpl. reflexivity.
rewrite IHn. reflexivity. Qed.
```

Exercise 2.4 Prove the following lemmas.

```
Lemma app_asso (X : Type) (xs ys zs : list X) :
app (app xs ys) zs = app xs (app ys zs).

Lemma length_app (X : Type) (xs ys : list X) :
length (app xs ys) = (length xs) + (length ys).

Lemma rev_app (X : Type) (xs ys : list X) :
rev (app xs ys) = app (rev ys) (rev xs).

Lemma rev_rev (X : Type) (xs : list X) :
rev (rev xs) = xs.
```

Solution to Exercise 2.4

```
Lemma app_asso (X : Type) (xs ys zs : list X) :
app (app xs ys) zs = app xs (app ys zs).
Proof.
induction xs ; simpl. reflexivity. rewrite IHxs. reflexivity.
Qed.

Lemma length_app (X : Type) (xs ys : list X) :
length (app xs ys) = (length xs) + (length ys).
Proof.
induction xs ; simpl. reflexivity. rewrite IHxs. reflexivity.
Qed.

Lemma rev_app (X : Type) (xs ys : list X) :
rev (app xs ys) = app (rev ys) (rev xs).
Proof.
induction xs ; simpl. rewrite app_nil. reflexivity.
rewrite <- app_asso. rewrite IHxs. reflexivity.
Qed.

Lemma rev_rev (X : Type) (xs : list X) :
rev (rev xs) = xs.
Proof.
induction xs ; simpl. reflexivity. rewrite rev_app.
```

```
simpl. rewrite IHxs. reflexivity.
Qed.
```

Exercise 2.5 Here is a tail-recursive function that obtains the length of a list with an accumulator argument.

```
Fixpoint lengthi {X : Type} (xs : list X) (a : nat) :=
match xs with
| nil => a
| cons _ xr => lengthi xr (S a)
end.
```

Proof the following lemmas.

```
Lemma lengthi_length {X : Type} (xs : list X) (a : nat) :
lengthi xs a = (length xs) + a.
```

```
Lemma length_lengthi {X : Type} (xs : list X) :
length xs = lengthi xs O.
```

Solution to Exercise 2.5

```
Lemma lengthi_length {X : Type} (xs : list X) (a : nat) :
lengthi xs a = (length xs) + a.
Proof.
revert a. induction xs ; intros a' ; simpl. reflexivity.
rewrite IHxs. rewrite add_S. reflexivity.
Qed.
```

```
Lemma length_lengthi {X : Type} (xs : list X) :
length xs = lengthi xs O.
Proof.
rewrite lengthi_length. rewrite add_O. reflexivity.
Qed.
```

Exercise 2.6 Define a predecessor function *nat* → *option nat*.

Solution to Exercise 2.6

```
Definition pred (n : nat) : option nat :=
match n with
| O => None
| S n' => Some n'
end.
```

Exercise 2.7 One can define a bijection between *bool* and *fin2*. Show this fact by completing the definitions and proving the lemmas shown below.

```
Definition f (x : bool) : fin 2 :=
Definition g (x : fin 2) : bool :=
Goal forall b : bool, g (f b) = b.
Goal forall x : fin 2, f (g x) = x.
```

Solution to Exercise 2.7

```
Definition f (b : bool) : fin 2 := if b then Some None else None.
Definition g (f : fin 2) : bool := match f with None => false | _ => true end.

Goal forall b, g (f b) = b.
destruct b ; reflexivity. Qed.
Goal forall x, f (g x) = x.
destruct x ; simpl. destruct i.
destruct v. reflexivity. reflexivity. Qed.
```

Exercise 2.8 Prove

```
Goal forall X:Type, forall x y z:X, x = y -> y = z -> x = z.
```

Solution to Exercise 2.8

```
Goal forall X:Type, forall x y z:X, x = y -> y = z -> x = z.
intros X x y z A B. rewrite A. rewrite B. reflexivity.
Qed.
```