Coq Exercises

{jba,jsp,mjf}(at)di.uminho.pt

MAP/i 2007

1 Logical Reasoning

1. Prove the following theorem:

   $$\textbf{Theorem}\ \mathit{ex4} : \forall\ (X : \mathsf{Set})\ (P : X \to \mathsf{Prop}), \sim(\forall\ x, \sim(P\ x)) \to (\mathit{exists}\ x, P\ x).$$

2. Assuming the *Excluded Middle* axiom, prove:

   (a) **Theorem** $\mathit{Pierce} : \forall\ P\ Q, ((P \to Q) \to P) \to P.$

   (b) **Theorem** $\mathit{NNE} : \forall\ P, \sim\sim P \to P.$

2 Reasoning about lists

The following exercises require the library `Lists`. You can load that library executing the command Require Import *Lists*.

1. Consider the following inductive relation:

   Inductive $\mathit{last}\ (A : \mathsf{Set})\ (x : A) : \mathit{list}\ A \to \mathsf{Prop} :=$
   $\mid \mathit{last\_base} : \mathit{last}\ x\ (\mathit{cons}\ x\ \mathit{nil})$
   $\mid \mathit{last\_step} : \forall\ l\ y, \mathit{last}\ x\ l \to \mathit{last}\ x\ (\mathit{cons}\ y\ l).$

   (a) Use `inversion` to prove that $\forall\ x, \sim(\mathit{last}\ x\ \mathit{nil})$.

   (b) (Difficult) Try to avoid using that tactic.

2. Consider the following definition for the `Even` predicate:

   Inductive $\mathit{Even} : \mathit{nat} \to \mathsf{Prop} :=$
   $\mid \mathit{Even\_base} : \mathit{Even}\ 0$
   $\mid \mathit{Even\_step} : \forall\ n, \mathit{Even}\ n \to \mathit{Even}\ (S\ (S\ n)).$

   (a) Define the `Odd` predicate (without mention *Even*).

   (b) (Difficult) Prove that, for every number n, $\mathit{Even}\ n \to \mathit{Odd}\ (S\ n)$. (*Hint: you should strength the property to* $(\mathit{Even}\ n \leftrightarrow \mathit{Odd}\ (S\ n)) \wedge (\mathit{Odd}\ n \leftrightarrow \mathit{Even}\ (S\ n))$)

   (c) Define the function `rev` that reverses a list.

(d) Prove that, for every list l, $rev\ (rev\ l) = l$.

(e) Recall the definition for the function `app` (concatenation of lists). Prove that for every lists l_1 and l_2, $rev\ (app\ l1\ l2) = app\ (rev\ l2)\ (rev\ l1)$.

3. Consider the inductive predicate:

$$\begin{array}{l}\textbf{Inductive}\ InL\ (A : \mathsf{Type})\ (a : A) : list\ A \to \mathsf{Prop} := \\ \quad \mid InHead : \forall\ (xs : list\ A), InL\ a\ (cons\ a\ xs) \\ \quad \mid InTail : \forall\ (x : A)\ (xs : list\ A), InL\ a\ xs \to InL\ a\ (cons\ x\ xs).\end{array}$$

Prove the following properties:

(a) $\forall\ (A : \mathsf{Type})\ (a : A)\ (l1\ l2 : list\ A), InL\ a\ l1 \lor InL\ a\ l2 \to InL\ a\ (app\ l1\ l2)$.

(b) $\forall\ (A : \mathsf{Type})\ (a : A)\ (l1\ l2 : list\ A), InL\ a\ (app\ l1\ l2) \to InL\ a\ l1 \lor InL\ a\ l2$.

4. Define the function `elem` that checks if an element belongs a list of integers (*Hint: use the predefined `Z_eq_dec` which tests for integer equality — for that you should import the `ZArith` module*).

5. Prove that, $\forall\ (a : Z)\ (l1\ l2 : list\ Z), elem\ a\ (app\ l1\ l2) = orb\ (elem\ a\ l1)\ (elem\ a\ l2)$ (the function `orb` is the boolean-or function defined in Library `Bool`).

6. Prove the correctness/completeness of `elem`, i.e. $\forall\ (x : Z)\ (l : list\ Z), InL\ x\ l \leftrightarrow elem\ x\ l = true$.

3 Specifications and Extraction

1. Prove decidability of InL, i.e. $\forall\ (x : Z)\ (l : list\ Z), \{InL\ x\ l\} + \{\sim InL\ x\ l\}$.

2. Perform extraction of the previous result. Compare the result with `elem`. Could you have used the correctness of `elem` to prove the decidability of InL.

3. Consider the well-known list functions `app` and `rev`.

   (a) Give a (relational) specification for them;

   (b) Prove the corresponding corretness assertions.

4 Function Encoding

1. Define in Coq the following Haskell function:

$$\begin{array}{l} div :: Int \to Int \to (Int, Int) \\ div\ n\ d = divAux\ n\ n\ d \\ \textbf{where}\ divAux\ 0\ \_\ \_ = (0, 0) \\ \quad divAux\ (x+1)\ n\ d \mid n < d = (0, n) \\ \qquad \mid otherwise = \mathsf{let}\ (q, r) = divAux\ x\ (n - d)\ d \\ \qquad\quad \mathsf{in}\ (q + 1, r) \end{array}$$

2. Check its results for several alguments.

5 Permutations

Consider the following definition for the permutation relation:

Fixpoint *count* $(z : Z)$ $(l : list\ Z)\{\mathsf{struct}\ l\} : nat :=$
  match l with
  $|\ nil \Rightarrow 0$
  $|\ (z' :: l') \Rightarrow$
    match $Z\_eq\_dec\ z\ z'$ with
    $|$ left $\_ \Rightarrow S\ (count\ z\ l')$
    $|$ right $\_ \Rightarrow count\ z\ l'$
    end
  end.

Definition *Perm* $(l1\ l2 : list\ Z)$: Prop :=
  $\forall\ z, count\ z\ l1 = count\ z\ l2.$

1. Prove that *Perm* is an equivalence relation (i.e. it is reflexive, symmetric and transitive).
2. Prove that, $\forall\ x\ y\ l, Perm\ (x :: y :: l)\ (y :: x :: l)$.

6 Merge Sort

1. Define in Coq the following functions:

$$
\begin{aligned}
&merge\ [\,]\ l = l \\
&merge\ l\ [\,] = l \\
&merge\ (x : xs)\ (y : ys) \mid x \leqslant y = x : merge\ xs\ (y : ys) \\
&\qquad\qquad\qquad\qquad\quad\ \mid otherwise = y : merge\ (x : xs)\ ys \\
&\mathsf{split}\ [\,] = ([\,], [\,]) \\
&\mathsf{split}\ (x : xs) = \mathsf{let}\ (a, b) = \mathsf{split}\ xs\ \mathsf{in}\ (x : b, a) \\
&merge\_sort\ [\,] = [\,] \\
&merge\_sort\ [x] = [x] \\
&merge\_sort\ l = \mathsf{let}\ (a, b) = \mathsf{split}\ l \\
&\qquad\qquad\qquad\quad \mathsf{in}\ merge\ (merge\_sort\ a)\ (merge\_sort\ b)
\end{aligned}
$$

(HINT: the `Function` command is a big help here — the `merge` function was defined in the lecture slides.)