Lab 10

Functional Programming (ITI0212)

2020.04.22

This week we learned about how to interpret a proposition as a type, and a proof of a proposition as a term of the corresponding type.

To complete this lab you will need the modules `Even` and `LEQ` that we developed interactively during lecture. Recall that in order to access their contents in your lab file, you will need to `import` the respective files, `even.idr` and `leq.idr`, located on the course web site beside this document.

Exponentiating an Even Number

In lecture this week we proved that the sum and product of two even numbers is even. In this section we consider exponentiation. Without thinking too hard, we might believe that for any n, if m is even then the exponential m^n is even too. This is almost true, except in the case that n is 0.

In order to prove that a positive power of an even number is even, we will first define a predicate for positive numbers as a `Nat`-indexed type:

```
data  Positive  :  Nat -> Type  where
  One_positive  :  Positive (S Z)
  S_positive    :  Positive n -> Positive (S n)
```

which we can think of as follows:

$$\begin{tikzcd}
 & \texttt{One\_positive} & \begin{array}{c}\texttt{S\_positive}\\ \texttt{One\_positive}\end{array} & \begin{array}{c}\texttt{S\_positive}\\ \texttt{(S\_positive}\\ \texttt{One\_positive)}\end{array} & \\
\texttt{0} \arrow[r, mapsto, "\texttt{S}"'] & \texttt{1} \arrow[r, mapsto, "\texttt{S}"'] & \texttt{2} \arrow[r, mapsto, "\texttt{S}"'] & \texttt{3} & \cdots
\end{tikzcd}$$

The type `Positive 0` is empty, the type `Positive 1` is a singleton containing `One_positive`, and every type `Positive (S (S n))` is a singleton containing the value of the function `S_positive` applied to the sole inhabitant of `Positive (S n)`.

We can use this type together with `Even` to express the proposition that we wish to prove:

```
{m : Nat} -> Even m -> {n : Nat} -> Positive n -> Even (power m n)
```

where `power` is the exponentiation function on natural numbers from the standard library. This should go smoothly, except for one wrinkle.

Task 1
Because we don't yet know how to tell Idris about equality, we will need the following easy lemma, which you should prove:

```
even_times_one  :  Even n -> Even (n * 1)
```

We are now nearly ready to prove that a positive power of an even number is even. Often the easiest way to prove a property about a recursively defined object is to try to follow its recursive structure in the structure of your proof.

Task 2
Examine the recursive structure of the exponentiation function on natural numbers with `:printdef power`. On which argument(s) is it recursive?

Task 3
Write a proof of the theorem that a positive power of an even number is even by following the recursive structure of the exponentiation function.

```
pow_even_pos  :  Even m -> Positive n -> Even (power m n)
```

Monotonicity of Addition

In lecture we proved that `LEQ` is a preorder relation on the natural numbers. Now we will prove that the addition operation is monotonic with respect to this order.

Task 4
As a warm-up, prove the following facts about adding zero on the right or on the left:

```
zero_plus_right  :  LEQ (m + 0) (m + n)

zero_plus_left   :  LEQ (0 + n) (m + n)
```

As you examine the intermediate proof states, recall that addition of natural numbers is defined recursively on the first argument (`:printdef plus`), so that as far as Idris is concerned `0 + n` and `n` are interchangeable, and likewise, `S m + n` and `S (m + n)` are interchangeable.

Task 5
Now prove that addition is monotonic in its right and left arguments:

```
plus_mono_right  :  LEQ n0 n1 -> LEQ (m + n0) (m + n1)

plus_mono_left   :  LEQ m0 m1 -> LEQ (m0 + n) (m1 + n)
```

Task 6
Finally, prove that addition is monotonic in both arguments simultaneously:

```
plus_mono  :  LEQ m0 m1 -> LEQ n0 n1 -> LEQ (m0 + n0) (m1 + n1)
```

Hint: this should be a one-liner using task 5 and the preorder structure on `LEQ`.