Idris Exercises: Part 2

Edwin Brady

Kats Idris Workshop, 21st May 2016

Questions

1. Implement the following function, which states that if you add the same value onto the front of equal lists, the resulting lists are also equal:

   ```
   same_cons : {xs : List a} -> {ys : List a} ->
               xs = ys -> x :: xs = x :: ys
   ```

   Since this function represents an equality proof, it is sufficient to know that your definition type checks.

2. Define a type `ThreeEq` which expresses that three values must be equal.
   (*Hint:* `ThreeEq` should have the type `a -> b -> c -> Type`)

3. Implement the following function:

   ```
   allSameS : (x, y, z : Nat) -> ThreeEq x y z -> ThreeEq (S x) (S y) (S z)
   ```

4. Implement the following function which checks whether two lists are equal and returns a proof if so:

   ```
   checkEqList : DecEq a => (xs : List a) -> (ys : List a) -> Maybe (xs = ys)
   ```

   Can you improve the definition using `Dec` instead of `Maybe`?

5. The following definition of a reverse function on vectors doesn't type check:

   ```
   my_reverse : Vect n elem -> Vect n elem
   my_reverse [] = []
   my_reverse (x :: xs) = my_reverse xs ++ [x]
   ```

   How can you correct it using `rewrite`?
   (*Hint:* Look up the library function `plusCommutative`)

6. The following definition of a reverse function is more efficient, but incomplete:

   ```
   my_reverse : Vect n a -> Vect n a
   my_reverse xs = reverse' [] xs
     where reverse' : Vect n a -> Vect m a -> Vect (n+m) a
           reverse' acc [] = ?reverseProof_nil acc
           reverse' acc (x :: xs)
                           = ?reverseProof_xs (reverse' (x::acc) xs)
   ```

   Complete this definition by implementing the holes `reverseProof_nil` and `reverseProof_xs`.