Chapter 3 Exercises

Types and Clauses

Connor Baker

Compiled on December 21, 2019 at 9:03pm

3.11 Exercises

1. What are the types of the following values?
   - `['a','b','c'] :: ?`
     - `[Char]`
   - `('a','b','c'):: ?`
     - `(Char, Char, Char)`
   - `[(False,'O'),(True,'1')] :: ?`
     - `[(Bool, Char)]`
   - `([False,True],['0','1']):: ?`
     - `([Bool], [Char])`
2. Write down definitions that have the following types; it does not matter what the definitions do as long as they are type correct.
   - `bools :: [Bool]`
     - `bools = [False, True]`
   - `nums :: [[Int]]`
     - `nums = [[0,1],[2,3]]`
   - `add :: Int -> Int -> Int -> Int`
     - `add a b c d = a + b + c + d`
   - `copy :: a -> (a,a)`
     - `copy x = (x,x)`
   - `apply :: (a -> b)-> a -> b`
     - Note that this is equivalent to `apply :: (a -> b)-> (a -> b)`, so we can use `apply f x = f x`
3. What are the types of the following functions?
   - `second xs = head (tail xs)`
     - `second :: [a] -> a`
   - `swap (x,y)= (y,x)`?
     - `swap :: (a,b)-> (b,a)`
   - `pair x y = (x,y)`?
     - `pair :: a -> b -> (a,b)`
   - `double x = x * 2`?
     - `double :: Num a => a -> a`
   - `palindrome xs = reverse xs == xs`?

    - `palindrome :: [a] -> Bool`
- `twice f x = f (f x)`?
    - `twice :: (a -> a)-> (a -> a)`

4. Check your answers to the proceeding questions using GHCi.

   Omitted.

5. Why is it not feasible in general for function types to be instances of the `Eq` class? When is it feasible? Hint: two functions of the same type are equal if they always return equal results for equal arguments.

   Being an instance of the `Eq` class means that the type is comparable. With integers, comparability is no problem – we are taught the strict total ordering of $<$ on the set of the integers with the number line from an early age. However, to discern whether two functions are equal, one must check whether they define the same mappings. This requires operating on the entirety of the domain of both functions, which is computationally expensive.

   One can raise additional questions:

   - How would one define a function as less than another function?
   - What if the functions don't have the same domains, but they have the same ranges?
   - What if there is a transformation on the domain that makes them equivalent?
     - Should we then also consider every possible transformation?
   - For these reasons and more, it is typically not possible to define an ordering on functions (I use weak language here on purpose – I do not have a great deal of math under my belt).