Homework 2 Solution

CS 440: Programming Languages and Translators, Fall 2019

Problems

1. (Function \( f \) = \text{length} \ (\text{filter} \ g \ x) \)
   a. \( f :: (a \rightarrow \text{Bool}) \rightarrow [a] \rightarrow \text{Int} \)
   b. It returns the number of \( x \) elements that pass test \( g \).
   c. \( f = \lambda g \rightarrow \text{length} \ . \ (\text{filter} \ g) \)

2. (Function \( f :: (a \rightarrow a \rightarrow a) \rightarrow a \rightarrow a \rightarrow a \))
   a. If \( f \ h \ x \ y = h \ x \ y \), then \( f \ (*) \ 2 \ 3 \) has no syntax errors and yields 6.
   b. Define \( g \ (h, (x, y)) = h \ (x, y) \), then \( g :: ((a, a) \rightarrow a, (a, a)) \rightarrow a \) and
      \( g \ (\text{uncurry} \ (\ast), (2,3)) = 6 \).

3. If \( f \ x \ y \ z = x :: ([y] : [z]) \), then \( f :: [a] \rightarrow a \rightarrow [a] \rightarrow [[a]] \)

4. (\( f \ x \ n = \text{take} \ n \ x == \text{take} \ n \ (\text{drop} \ n \ x) \).
   a. \( f \) tests the first \( n \) members of \( x \) against the second \( n \) members and returns true if they match. It returns false if \( x \) has < 2\( n \) elements, and it ignores any elements after the first 2\( n \).
   b. \( f \ x \ 5 \) should be true for any \( x \) of the form \( y \ ++ \ y \ ++ \ldots \) where \( y \) is of length 5. For example, if \( x = [1,2,3,4,5,1,2,3,4,5,9,9,9,9,9,9] \), we will get true.

5. (Redefine \( \text{foldl} \) on lists)
   a. (With a conditional expression)
      \[
      \text{foldl1} \ f \ a \ x
      = \text{if} \ x == [\] \ \text{then} \ a \ \text{else} \ \text{foldl1} \ f \ (f \ a \ (\text{head} \ x)) \ (\text{tail} \ x)
      \]
   b. (Function definition by cases)
      \[
      \text{foldl2} \ _ \ a \ [] = a
      \text{foldl2} \ f \ a \ (h : t) = \text{foldl2} \ f \ (f \ a \ h) \ t
      \]
   c. (Using a case expression)
      \[
      \text{foldl3} \ f \ a \ x = \text{case} \ x \ \text{of} \ [\] \rightarrow a
                                  h : t \rightarrow \text{foldl3} \ f \ (f \ a \ h) \ t
      \]
6. \( \text{(twice \( x \) = does some value occur twice in \( x \))?} \)

a. \( \text{(Code problems)} \) Let me number the lines of code to make referencing them easier:

1. \( \text{twice :: \([ \text{a}] \rightarrow \text{Bool} \)} \)
2. \( \text{twice :: \([ \text{a}] \rightarrow \text{Bool} \)} \)
3. \( \text{twice \([ \text{} \) = False} \)
4. \( \text{twice \([,] \) = False} \)
5. \( \text{twice \([x,x] \) = True} \)
6. \( \text{twice \([,] \) = False} \)
7. \( \text{twice \((h1 : h2 : t) = (h1 == h2 | | twice h1 t) \)} \)
8. \( \text{twice \(( _ ++ [x] ++ _ ++ [y] ++ _ ) = x \text{ == y} \)} \)

So for errors,

- Line 1: Since we're going to be comparing the elements of the list, we need type \( \text{a} \) (in \( [ \text{a}] \rightarrow \text{Bool} \)) to support equality testing. Replace the line by
  \( \text{twice :: Eq \text{a} => \([ \text{a}] \rightarrow \text{Bool} \)} \)

- Line 5: Can't use \( x \) twice in the pattern \([x,x]\). One fix: Replace the line by
  \( \text{twice \{x,y\} = (x == y) \)} \)

- Line 6: Is now obsolete. Presumably the original idea was that the \([x,x]\) pattern from line 4 was supposed to check for two matching values so line 5 was supposed to handle two non-matching values, but that case is now handled by line 4.

- Line 7: There's a syntactic error: instead of twice \( h1 \ t \), we need twice \( (h1:t) \) because twice takes one argument, a list. There's also a semantic error: If \( h1 \neq h2 \) and \( h1 \) doesn't occur in \( t \), we should check for \( h2 \) occurring in \( t \). Otherwise a call like twice \([1,2,2]\) won't return true. The line becomes
  \( \text{twice \((h1 : h2 : t) = (h1 == h2 | | twice (h1:t) | | twice (h2:t)) \)} \)

- Line 8: Uses a bad pattern: You can only try to match lists against \([...]\) and \((:); you can't use a general function. The whole line needs to be deleted.

b. Combining the fixes from part (a), we get

\( \text{twice :: Eq \text{a} => \([ \text{a}] \rightarrow \text{Bool} \)} \)
\( \text{twice :: \([ \text{a}] \rightarrow \text{Bool} \)} \)
\( \text{twice \([ \text{} \) = False} \)
\( \text{twice \([,] \) = False} \)
\( \text{twice \([x,y] \) = x \text{ == y} \)
\( \text{twice \((h1 : h2 : t) = (h1 == h2 | | twice (h1:t) | | twice (h2:t)) \)

(c. \( \text{(Extra credit)} \) The next-to-last line \( \text{twice \([x,y] \) = ...} \) is redundant because it's covered by the last line when \( t \) is empty. Now that the last line is the only one that can return true, we can check it first and always return false (on lists of length < 2).
twice :: Eq a => [a] -> Bool
    twice (h1 :: h2 :: t) = (h1 == h2 || twice (h1 : t) || twice (h2 : t))
    twice _ = False

7. \( f \cdot g \cdot x \cdot y = g \cdot x \cdot (y \cdot x) \cdot \)
   a. \[ f \cdot g \cdot x = \lambda y. g \cdot x \cdot (y \cdot x) \]
      \[ f \cdot g = \lambda x. y. g \cdot x \cdot (y \cdot x) \]
      \[ f = \lambda g \cdot x. y. g \cdot x \cdot (y \cdot x) \]
   b. Having \( \text{var} = \text{undefined lambda function} \) means that giving a function a name is just like giving any other kind of expression a name; treating functions like any other kind of value is what first-class functions are about. (The \( f \cdot x = \text{expr} \) syntax is there to make life easier.)

8. Say we have \( f :: \text{type} 1 \rightarrow \text{type} 2 \). For \( f \) to be higher-order, either \( \text{type} 1 \) or \( \text{type} 2 \) (or both) involve functions. In the first case, \( \text{type} 1 \) contains an arrow; in the second, \( \text{type} 2 \) contains an arrow. So \( f \) is higher-order iff it has at least two arrows.
   Examples: \( \text{map} \) and \( \text{filter} \) take functional parameters. Function composition \( (\cdot) \), which is of type \( (\text{a} \rightarrow \text{b}) \rightarrow (\text{b} \rightarrow \text{c}) \rightarrow (\text{a} \rightarrow \text{c}) \), takes functional parameters and produces a functional result.
   Also note that a function with \( \geq 2 \) parameters that is curried will be higher order because partial evaluation gives you a function. E.g., \( (\cdot) \cdot 2 \) is a function, so if you apply it to \( 3 \) using \( (\cdot) \cdot 2 \cdot 3 \), you get \( 6 \). (You can emphasize that \( (\cdot) \cdot 2 \) is a function if you want by writing \( ((\cdot) \cdot 2) \cdot 3 \) or by letting \( \text{var} = (\cdot) \cdot 2 \) and then using \( \text{var} \cdot 3 \).)