FUNCTIONAL PROGRAMMING MT2020

Sheet 2

3.1 Generalising an earlier exercise: for finite types a, b and c there are as many functions of type $a \to b \to c$ as there are are of type $(a, b) \to c$ (because as numbers $(c^b)^a = c^{b \times a}$).

This correspondence, and the similar one for infinte types, is demonstrated by the (predefined) functions

```
curry   :: ((a,b) -> c) -> (a -> b -> c)
uncurry :: (a -> b -> c) -> ((a,b) -> c)
```

for which both $curry \cdot uncurry$ and $uncurry \cdot curry$ are identity functions (of the appropriate type).

There is a unique total function of each of these types. Write out what must be the definitions of these two functions, and prove that they are mutually inverse. (If you type these definitions at an interpreter, remember to change their names to avoid clashing with the Prelude functions.)

3.2 Suppose $h\ x\ y = f\ (g\ x\ y)$. Which of the following are true, which are false, and (in each case) why?

1. $h = f \cdot g$
2. $h\ x = f \cdot g\ x$
3. $h\ x\ y = (f \cdot g)\ x\ y$

3.3 Give most general types for the following, where possible

```
> subst f g x = (f x) (g x)
> fix f = f (fix f)
> twice f = f . f
> selfie f = f f
```

You should try to work out what the most general type is by hand, but you can check that you are right by using an interpreter; and if you are wrong, check that you understand why.

4.1 Show that if f and g are strict, so is the composition $f \cdot g$.
Is the converse true: that if $f \cdot g$ is strict, so must f and g be?

4.2 Which of these equations are badly typed? For the others, what can you say about the type of xs, and whether and when the equation holds?

$a)\,[\,] : xs = xs$	$e)\,xs : [\,] = [xs]$	$i)\,[[\,]] +\!\!+ xs = xs$
$b)\,[[\,]] +\!\!+ [xs] = [[\,], xs]$	$f)\,[\,] : xs = [[\,], xs]$	$j)\,xs : xs = [xs, xs]$
$c)\,[[\,]] +\!\!+ xs = [xs]$	$g)\,[xs] +\!\!+ [\,] = [xs]$	$k)\,xs : [\,] = xs$
$d)\,xs : [xs] = [xs, xs]$	$h)\,[[\,]] +\!\!+ xs = [[\,], xs]$	$l)\,[xs] +\!\!+ [xs] = [xs, xs]$

4.3 Suppose that the class of ordered types is declared by something like

```
class Eq a => Ord a where
  (<), (<=), (>), (>=) :: a -> a -> Bool
  x <  y = not (x >= y)
  x >  y = not (x <= y)
  x >= y = x == y || x > y
```

(It includes a couple of other things, and these are not quite the default definitions.) Lists are lexicographically ordered, like the words of a dictionary. Write an instance declaration for `Ord [a]`.

4.4 Time (as Richard Bird might say) for a song:

```
One man went to mow
Went to mow a meadow
One man and his dog
Went to mow a meadow

Two men went to mow
Went to mow a meadow
Two men, one man and his dog
Went to mow a meadow

Three men went to mow
Went to mow a meadow
Three men, two men, one man and his dog
Went to mow a meadow
```

Write a Haskell function $song :: Int \to String$ so that $song\ n$ is the song when there are n men (and a dog). Assume $n \leqslant 10$. To print the song, type for example: `putStr (song 5)`. You may want to start from

```
> song 0  = ""
> song n  = song (n-1) ++ "\n" ++ verse n
> verse n = line1 n ++ line ++ line3 n ++ line
```

Geraint Jones, 2020