Exam Functional Programming – December 3rd 2014

Name	
Student number	
I study CS/AI/Other	

- Write **neatly** and carefully. Use a pen (no pencil!) with black or blue ink.
- Write your answers in the answer boxes. If you need more space, use the back side of the sheet and make a reference to it.
- You can score 90 points. You get 10 points for free, yielding a maximum of 100 points in total. Your exam grade is the obtained points divided by 10.
- If you need auxiliary lemmas in a proof, then prove the validity of these lemmas as well.

You may use throughout the entire exam the following functions:

```
[]     ++ ys            =  ys
(x:xs) ++ ys            =  x : (xs++ys)

map f  []               =  []
map f (x:xs)            =  f x : map f xs

foldr f z []            = z
foldr f z (x:xs)        = f x (foldr f z xs)

sum []                  = 0
sum (x:xs)              = x + sum xs

reverse []              = []
reverse (x:xs)          = reverse xs ++ [x]

head (x:xs)             = x
tail (x:xs)             = xs

length []               = 0
length (x:xs)           = 1 + length xs

f . g                   = \x -> f (g x)

zip (x:xs) (y:ys)       = (x,y) : zip xs ys
zip  xs     ys          = []

zipWith _ [] _          = []
zipWith _ _ []          = []
zipWith f (x:xs) (y:ys) = f x y : zipWith f xs ys
```

1. (5× 2=10 points)

(a) What is the type of the following Haskell function `wtel`?

```
wtel [] = []
wtel (x:xs) = if x == [] then wxs else x:wxs
  where wxs = wtel xs
```

(b) What is the type of the following Haskell function `cl`?

```
cl ps = ps ++ [(p,s) | (p,q) <- ps, (r,s) <- ps, q==r]
```

(c) What is the type of the standard Haskell indexing operator `!!` (as an example `[0..10]!!3 = 3`)?

(d) What is the type of the following Haskell function `map2`?

```
map2 f [] [] = []
map2 f (x:xs) (y:ys) = (f x y) : map2 f xs ys
```

(e) What is the type of the following Haskell function `tw`?

```
tw = (\f -> (\x -> (f.f) x))
```

2. (15 points) Consider a positive integer N. We denote its decimal digits by X_0, X_1, ..., X_k. The number N is called a *funny number* if you can select at most three (but at least one) of its digits such that N is a divisor of the number $(X_0 + X_1 + ... + X_k - S)^S$, where S is the sum of the selected digits. As an example, 1458 is a funny number since $((1 + 4 + 5 + 8) - (1 + 5))^{1+5} = 12^6 = 2985984$ is divisible by 1458. Note that we selected the two digits 1 and 5.

   Write a Haskell function `isFunny` (including its type) that takes an integer number as its argument, and returns `True` if and only if this argument is a funny number.

3. (3+3+4=10 points)

- Use a list comprehension to define a function `inverse::[(a, b)]->[(b, a)]` such that `elem (x,y) ps` if and only if `elem (y,x) (inverse ps)`.

- Use a list comprehension to make your own implementation of the standard Haskell function `replicate`. The call `replicate n x` yields a list of length `n` with `x` being the value of every element. So, `replicate 5 'a'` returns `"aaaaa"`.

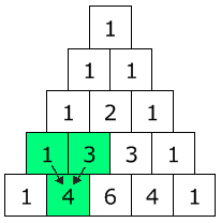
- Define a function `doubleReverse` which takes a list of strings as its argument and reverses each element of the list and then reverses the resulting list. The implementation of `doubleReverse` must use a list comprehension. As an example, `doubleReverse ["hello", "world"] = ["dlrow", "olleh"]`.

4. (3+3+4=10 points)

- The function `powers n` returns the infinite list $[\mathtt{n}^0, \mathtt{n}^1, \mathtt{n}^2, \mathtt{n}^3, ...]$. Give a *recursive* Haskell implementation (including its type) of the function `powers`.

- The sequence a_k is defined as follows: $a_0 = 1$, $a_1 = 2$, $a_k = 3a_{k-1} + 2a_{k-2}$ for integer $k > 1$. Define the infinite list `seqa`, which is the list $[a_0, a_1, a_2, a_3, a_4, ...]$, so `seqa!!k` should yield a_k.

- In the following figure you see the first 5 rows of Pascal's triangle:



To build the triangle, start with the row `[1]` at the top (we call this row 0), then continue placing numbers below it in a triangular pattern. Each row consists of elements that are the sum of the two numbers above it (except for the boundaries, which are all `1`). In the figure, it is highlighted that the 4 in row 4 is obtained by adding the numbers 1 and 3 from row 3.
Give a definition of the infinite list `pascalTriangle ::[[Integer]]`, such that `pascalTriangle!!n` yields the `n`th row of Pascal's triangle (i.e. `pascalTriangle!!4 = [1,4,6,4,1]`).

5. (15 points) The abstract data type (ADT) `Set tp` implements a data type for the storage of *sets* of the type `tp`, where `tp` is of the class `Ord` (i.e. the elements are ordered).

   Implement a `module Set` that exports the ADT `Set`. You can choose a concrete implementation yourself, however this implementation must be hidden from the user of this module.

   The following operations on the data type `Set` must be implemented:

   - `empty` returns an empty set.
   - `isEmpty` returns `True` for an empty set, otherwise `False`.
   - `insert`: returns the set after insertion of an element.
   - `delete`: returns the set after removal of an element.
   - `union`: returns the union of two sets.
   - `intersection`: returns the intersection of two sets.

6. (15 points) Given are the following Haskell definitions of the functions `f` and `g`:

```
f :: Integer -> Integer
f 0 = 0
f 1 = 1
f n = 5*(f (n-1)) - 6*(f (n-2))

g :: Integer -> Integer -> Integer
g n 0 = 1
g n e = n*(g n (e - 1))
```

Prove for all natural numbers `n`: `f n = g 3 n - g 2 n`

7. (15 points) Given are the definitions of the Haskell functions `sum`, and `reverse`:

```
sum :: [Integer] -> Integer
sum [] = 0
sum (x:xs) = (sum xs) + x

reverse :: [a] -> [a]
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]
```

Prove that `sum (reverse xs) = sum xs` for all finite lists `xs`.