ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D. Basin and P. Müller

Formal Methods and Functional Programming

Exercise Sheet 4: Higher-Order Functions and Correctness

Submission deadline: March 18th, 2012

Please start the subject of your e-mail submission with `[FMFP]`, and submit your Haskell code in a file called `sheet4_<your_nethz_username>.(l)hs`

Assignment 1

In order to compute relative to an arbitrary base, we represent numbers by the following datatype:

```
data Number = N Int [Int]
```

A number `N base digits` where `digits` $= [\mathtt{d}_0, \mathtt{d}_1, \ldots, \mathtt{d}_{n-1}]$ is converted to an integer using the function `toInt :: Number -> Int` defined as follows.

$$\mathtt{toInt\ (N\ base\ digits)} = \sum_{i=0}^{n-1} \mathtt{d}_i \cdot \mathtt{base}^i$$

For example, `N 10 [2,3,1]` represents $2 \cdot 10^0 + 3 \cdot 10^1 + 1 \cdot 10^2 = 132$ and `N 2 [0,1,1,1]` represents $0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 = 14$.

Implement the function `toInt` *without using recursion* according to the specification above.

Hint: You may find the functions `zipWith`, `iterate`, `map`, and `sum` useful.

Assignment 2

Consider the following functions.

```
rev :: [a] -> [a]
rev []     = []                    -- rev.1
rev (x:xs) = rev xs ++ [x]         -- rev.2

(++) :: [a] -> [a] -> [a]
(++) []     ys = ys                -- app.1
(++) (x:xs) ys = x : (xs ++ ys)    -- app.2
```

Formally prove that `rev (xs ++ rev ys) = ys ++ rev xs` holds for all `xs,ys::[a]`. Use structural induction over lists, be explicit about the quantification of variables, and justify all your reasoning steps.

Hint: You can use the following lemmas without proof.

Lemma 1: $\forall$`xs::[a]. rev (rev xs) = xs`.

Lemma 2: $\forall$`xs::[a]. xs ++ [] = xs`.

Lemma 3: $\forall$`a,b,c::[a]. (a ++ b) ++ c = a ++ (b ++ c)`.

Assignment 3:

(a) Define a function `split :: Char -> String -> [String]` that splits a string, which consists of substrings separated by a separator, into a list of strings. Examples:

```
split '#' "foo##goo" = ["foo","","goo"]
split '#' "#" = ["",""]
```

(b) Define a function `center :: [String] -> [String]` that centers the strings in the list by appending spaces to the left and right. For the argument list `l`, `center l` should return a list with the original strings enclosed in spaces so that all strings in the resulting list have the same length and the original string is centered in the string with added spaces (i.e. when x spaces are appended to the left and y to the right, then $|x - y| \leq 1$ must hold).

Example:

```
center ["fullness","of","exposition","is","necessary"] =
  [" fullness ",
   "    of    ",
   "exposition",
   "    is    ",
   "necessary "]
```

Hint: You can use the predefined functions `maximum :: Ord a => [a] -> a` and `replicate :: Int -> a -> [a]`, where `maximum l` returns the maximal element in the list `l` and `replicate n x` returns the list of length `n`, where `x` is the value of every element.

Assignment 4

(a) The function `filterMap :: (b -> Bool) -> (a -> b) -> [a] -> [b]` is defined as.

```
filterMap p f = filter p . map f
```

Another possible definition is based on `foldr` assuming that `aux` and `e` are defined appropriately.

```
filterMap' p f = foldr aux e
  where aux = ...
        e   = ...
```

Give definitions for `aux` and `e` such that `filterMap = filterMap'`.

(b) **Headache of the week:** Define the `foldl` function with `foldr`. Only use λ-abstraction, `id`, and `foldr`. Note that `id` is the identity function, i.e., `id x = x`. Your solutions should have the form `myFoldl f v l = foldr .. .`