DEPARTMENT EEMCS
Date: June 5, 2015

Test: Programming Paradigms — Functional Programming

June 12, 2015
13:45 – 16:45

Remarks:
- During this test you may use the syllabus: *Functional Programming – an overview*, nothing else.
- You may only use predefined Haskell functions and operators from the packages *Prelude*, *Data.List*, *Data.Char*.
- **Mention the type for *every* function that you define.**
- Judgement: there are three exercises, which all weigh equally in the total score.
- Style and elegancy also play a role in the judgement, e.g., do not use unnecessary helper functions, counters, etcetera.
- Good luck!

Opgave 1.

a. A *polynome of the order* n is a function of the form:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

Define a function *polynome* that calculates a polynome of the order n for the list of co-efficients $[a_0, a_1, \ldots, a_n]$. Define the function *polynome* in three different ways: with recursion, with higher order functions, and with list comprehension.

b. Write a function *coins* which determines all different ways in which a certain amount can be made from coins with the values: 1, 2, 5, 10, 20, 50 cents. For example, an amount of 4 cents can be made as follows (each number indicates the value of a coin):

[1,1,1,1], [1,1,2], [2,2]

Thus, your function should calculate a list of lists such that the total of each individual list is the given amount. Every combination should be given only once, the order in which the coins are mentioned is irrelevant (thus the lists [1,1,2], [1,2,1], [2,1,1] all represent the same combination of coins).

c. (1) Write a function *add35* which calculates the sum of all numbers which are smaller than a given number n, and which are a multitude of 3 and 5. If such a number is both a multitude of 3 *and* 5, it should only be counted once.

For example, for n=20 the list of all multitudes (smaller than n) of 3 and/or 5, together with the sum, are:

$$3 + 5 + 6 + 9 + 10 + 12 + 15 + 18 = 78$$

(2) Generalise your definition such that it works for an arbitrary set ks of numbers the multitudes of which (smaller than a given number n) should be included. Thus, for ks=[3, 5, 7] and n=20 the result should be:

$$3 + 5 + 6 + 7 + 9 + 10 + 12 + 14 + 15 + 18 = 99$$

Opgave 2.

a. Define a type *Tree* for *binary trees* with an *Int* at every internal node, but no information at the leaves.

b. A tree is *balanced* if the difference in length of the shortest and the longest branch (a branch is a path from the root to a leaf) is not more than 1. Write a function *isBalanced* which checks whether a tree of type *Tree* is balanced.

c. Two trees are *isomorphic* if they have the same *structure*, though the numbers at corresponding nodes need not be the same.

(1) Write a function *isomorphic* which checks whether two trees of type *Tree* are isomorphic.

(2) Write a function *isomorphicPlus* which checks whether two trees of type *tree* are isomorphic, *and* which checks whether corresponding numbers are in a given reation r to each other (for example, whether each number in tree t_1 is twice as big as its corresponding number in tree t_2).

d. A tree of type *Tree* is *sorted* if (1) all numbers in the tree are different, and (2) for *every* node p in the tree it holds that all numbers in the left subtree at node p are smaller than the number at node p, and all numbers in the right subtree at p are bigger than the number at p. Write a function *sorted* which checks whether a tree is sorted.

e. Write a function *addListToTree* which inserts a list of numbers to a sorted tree such that the resulting tree is sorted again. If a number from the list is already in the tree, this number should be skipped.

Opgave 3.

a. A *graph* is a set of nodes together with a set of edges. Assume that there is at most *one* edge between any pair of nodes, that the edges are *un*directed, and there are *no* weights on the edges.

Define a type *Graph* for such graphs. You may choose how to represent nodes, as long as that type belongs to the class *Eq*.

b. A *path* is a sequence of edges such that the end-node of an edge in the sequence is the same as the start-node of the next edge in the sequence. Write a function *pathExists* which checks whether there exists a path from node a to node b, i.e., node a is the start-node of the first edge in the path, end node b is the end-node of the last edge.

c. A graph is *connected* when for every pair of nodes a and b in the graph there exists a path between a and b. Write a function *connected* which checks whether a graph is connected.

d. Suppose g is a connected graph. Write a function *minimalCut* which finds the minimal number of edges such that after removing these edges from graph g, the graph will be disconnected.