RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET
INFORMATIK 2
RWTH Aachen · D-52056 Aachen · GERMANY
http://verify.rwth-aachen.de/

LuFG Informatik II

Functional Programming
Exam, February 24, 2010

Prof. Dr. Jürgen Giesl
Carsten Fuhs

First name: ______________________________

Last name: ______________________________

Matr. number: ______________________________

Course of study (please mark exactly one):

∘ Master of SSE

- On every sheet please give your **first name**, **last name**, and **matriculation number**.
- You must solve the exam **without** consulting any **extra documents** (e.g., course notes).
- Make sure your answers are readable. Do not use **red pens or pencils**.
- Please answer the exercises on the **exercise sheets**. If needed, also use the back sides of the exercise sheets.
- Answers on extra sheets can only be accepted if they are clearly marked with your name, your matriculation number, and the **exercise number**.
- **Cross out** text that should not be considered in the evaluation.
- Students that try to cheat **do not pass** the exam.
- At the end of the exam, please return **all sheets together with the exercise sheets**.

	Total number of points	Number of points obtained
Exercise 1	22	
Exercise 2	9	
Exercise 3	6	
Exercise 4	9	
Exercise 5	10	
Total	56	
Grade	-	

First name	Last name	Matriculation number

Exercise 1 (4 + 3 + 4 + 5 + 6 = 22 points)

The following data structure represents polymorphic lists that can contain values of *two* types in arbitrary order:

```
data DuoList a b = C a (DuoList a b) | D b (DuoList a b) | E
```

Consider the following list `zs` of integers and characters:

$$[\,4,\ 'a',\ 'b',\ 6\,]$$

The representation of `zs` as an object of type `DuoList Int Char` in Haskell would be:

```
C 4 (D 'a' (D 'b' (C 6 E)))
```

Implement the following functions in Haskell.

(a) The function `foldDuo` of type

```
(a -> c -> c) -> (b -> c -> c) -> c -> DuoList a b -> c
```

works as follows: `foldDuo f g h xs` replaces all occurrences of the constructor `C` in the list `xs` by `f`, it replaces all occurrences of the constructor `D` in `xs` by `g`, and it replaces all occurrences of the constructor `E` in `xs` by `h`. So for the list `zs` above,

```
foldDuo (*) (\x y -> y) 3 zs
```

should compute

```
(*) 4 ((\x y -> y) 'a' ((\x y -> y) 'b' ((*) 6 3))),
```

which in the end results in `72`. Here, `C` is replaced by `(*)`, `D` is replaced by `(\x y -> y)`, and `E` is replaced by `3`.

First name	Last name	Matriculation number

(b) Use the `foldDuo` function from (a) to implement the `cd` function which has the type `DuoList Int a -> Int` and returns the sum of the *entries* under the data constructor `C` and of the *number of elements* built with the data constructor `D`.

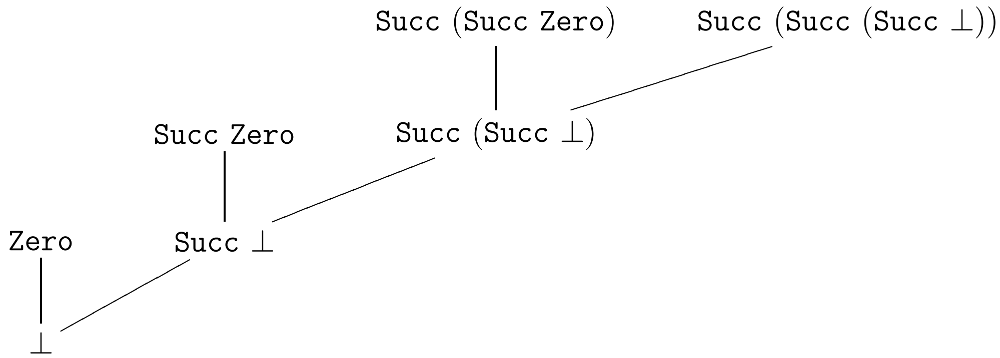
In our example above, the call `cd zs` should have the result `12`. The reason is that `zs` contains the entries `4` and `6` under the constructor `C` and it contains two elements `'a'` and `'b'` built with the data constructor `D`.

First name	Last name	Matriculation number

(c) Consider the following data type declaration for natural numbers:

```
data Nats = Zero | Succ Nats
```

A graphical representation of the first four levels of the domain for `Nats` could look like this:



We define the following data type `Single`, which has only one data constructor `One`:

```
data Single = One
```

Sketch a graphical representation of the first three levels of the domain for the data type `DuoList Bool Single`.

First name	Last name	Matriculation number

(d) Write a Haskell function `printLength` that first reads a line from the user, then prints this string on the console and in the end also prints the length of this string on the console. Also give the type declaration for your function.

You may use the `do`-notation, but you are not obliged to use it. Some of the following pre-defined functions can be helpful:

- `getLine :: IO String` reads a line from the user
- `length :: String -> Int` has the length of a string as its result
- `show :: Int -> String` converts a number to a string
- `putStr :: String -> IO ()` writes a string to the console

An example run should look as given below. Here the string "`foo`" was read from the user.

```
Main> printLength
foo
foo3
```

First name	Last name	Matriculation number

(e) The *digit sum* of a natural number is the sum of all digits of its decimal representation. For example, the digit sum of the number 6042 is $6 + 0 + 4 + 2 = 12$. Write a Haskell function `digitSum :: Int -> Int` that takes a natural number and returns its digit sum. Your function may behave arbitrarily on negative numbers. It can be helpful to use the pre-defined functions `div, mod :: Int -> Int -> Int` to compute result and remainder of division, respectively. For example, `div 7 3` is `2` and `mod 7 3` is `1`.

Now implement a function `digitSumList :: Int -> Int -> [Int]` where `digitSumList n b` returns a list of all those numbers `x` where `0` $\leq$ `x` $\leq$ `b` and where the digit sum of `x` is `n`. Perform your implementation only with the help of a **list comprehension**, i.e., you should use exactly one declaration of the following form:

```
digitSumList ... = [ ... | ... ]
```

Of course, here you can (and should) make use of the function `digitSum` to compute the digit sum of a number.

First name	Last name	Matriculation number

Exercise 2 (4 + 5 = 9 points)

Consider the following Haskell declarations for the `square` function:

```
square :: Int -> Int
square 0     = 0
square (x+1) = 1 + 2*x + square x
```

(a) Please give the Haskell declarations for the higher-order function `f_square` corresponding to `square`, i.e., the higher-order function `f_square` such that the least fixpoint of `f_square` is `square`. In addition to the function declaration(s), please also give the type declaration of `f_square`. Since you may use full Haskell for `f_square`, you do not need to translate `square` into simple Haskell.

(b) We add the Haskell declaration `bot = bot`. For each $n \in \mathbb{N}$ please determine which function is computed by `f_square`^{n} `bot`. Here "`f_square`^{n} `bot`" represents the n-fold application of `f_square` to `bot`, i.e., it is short for $\underbrace{\texttt{f\_square (f\_square} \ldots \texttt{(f\_square}}_{n \text{ times}}\texttt{ bot)}\ldots\texttt{)}$.

Let $f_n : \mathbb{Z}_\perp \to \mathbb{Z}_\perp$ be the function that is computed by `f_square`^{n} `bot`.
Give f_n in **closed form**, i.e., using a non-recursive definition.

First name	Last name	Matriculation number

Exercise 3 (6 points)

Let D_1, D_2 be domains, let $\sqsubseteq_{D_2}$ be a complete partial order on D_2. As we know from the lecture, then also $\sqsubseteq_{D_1 \to D_2}$ is a complete partial order on the set of all functions from D_1 to D_2.

Prove that $\sqsubseteq_{D_1 \to D_2}$ is also a complete partial order on the set of all *constant* functions from D_1 to D_2. A function $f : D_1 \to D_2$ is called *constant* iff $f(x) = f(y)$ holds for all $x, y \in D_1$.

> *Hint:* The following lemma may be helpful:
>
> If S is a chain of functions from D_1 to D_2, then $\sqcup S$ is the function with:
>
> $$(\sqcup S)(x) = \sqcup\{f(x) \mid f \in S\}$$

First name	Last name	Matriculation number

Exercise 4 (4 + 5 = 9 points)

Consider the following data structure for polymorphic lists:

```
data List a = Nil | Cons a (List a)
```

(a) Please translate the following Haskell-expression into an equivalent lambda term (e.g., using $\mathcal{L}am$). Recall that pre-defined functions like `even` are translated into constants of the lambda calculus.

It suffices to give the result of the transformation.

```
let f = \x -> if (even x) then Nil else Cons x (f x)
    in f
```

First name	Last name	Matriculation number

(b) Let δ be the set of rules for evaluating the lambda terms resulting from Haskell, i.e., δ contains at least the following rules:

$$\begin{aligned} \texttt{fix} &\rightarrow \lambda f.\ f\ (\texttt{fix}\ f) \\ \texttt{plus}\ 2\ 3 &\rightarrow 5 \end{aligned}$$

Now let the lambda term t be defined as follows:

$$t = (\texttt{fix}\ (\lambda g\ x.\ \texttt{Cons}\ (\texttt{plus}\ x\ 3)\ \texttt{Nil}))\ 2$$

Please reduce the lambda term t by WHNO-reduction with the $\rightarrow_{\beta\delta}$-relation. You have to give **all** intermediate steps until you reach **weak head normal form** (and no further steps).

First name	Last name	Matriculation number

Exercise 5 (10 points)

Use the type inference algorithm $\mathcal{W}$ to determine the most general type of the following lambda term under the initial type assumption A_0. Show the results of all sub-computations and unifications, too. If the term is not well typed, show how and why the $\mathcal{W}$-algorithm detects this.

$$\lambda f.\ (\texttt{Succ}\ (f\ x))$$

The initial type assumption A_0 contains at least the following:

$$\begin{array}{lcl} A_0(\texttt{Succ}) & = & (\texttt{Nats} \to \texttt{Nats}) \\ A_0(f) & = & \forall a.\ a \\ A_0(x) & = & \forall a.\ a \end{array}$$