RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://www-i2.informatik.rwth-aachen.de/lufgi2.html

Prof. Dr. Jürgen Giesl
Darius Dlugosz

Exercises *Functional Programming* – Sheet 3

Solutions will be collected until Thursday, May 17, 2001 (in 2 weeks!).

Important: On Thursday, May 17, there is a lecture course (Vorlesung) instead of practical course (Übung).

Exercise 1

The application of the function `foldr` $(\oplus)$ e xs yields a term in which the list elements are combined with the operator $\oplus$ in right–associative way.

`foldr` $(\oplus)$ e $[a_1, ..., a_n]$ = $a_1 \oplus (a_2 \oplus (...(a_n \oplus e)...))$

Dual to `foldr` define the function `foldl` where the parentheses group from left.

`foldl` $(\oplus)$ e $[a_1, ..., a_n]$ = $(...((e \oplus a_1) \oplus a_2)... \oplus a_n)$

Exercise 2 (1 points)

(a) Give a function f, so that `foldr` $f \neq$ `foldl` f.

(b) Define f and e, so that `foldl` f e $= id$.

(c) Define the function `map` using `foldr`.

Exercise 3 (2 points)

(a) The function `remdups` removes adjacent duplicates from a list. For example, `remdups [1,2,2,3,3,3,1,1] = [1,2,3,1]`. Define `remdups` using either `foldr` or `foldl`.

(b) Given a list $xs = [x_1, x_2, ..., x_n]$ of numbers, the sequence of successive maxima `ssm` xs is the longest subsequence $[x_{j_1}, x_{j_2}, ..., x_{j_m}]$ such that $j_1 = 1$ and $x_{j_l} < x_{j_k}$ for $j_l < j_k$. For example, the sequence of successive maxima of `[3,1,3,4,9,2,10,7]` is `[3,4,9,10]`. Define `ssm` in terms of `foldl`.

Exercise 4 (4 points)

The data types `Term` and `SP` are defined as follows:

```
data Term a = C Int | V a | Mul (Term a) (Term a) | Add (Term a) (Term a)

data SP a b = PutSP b (SP a b) | GetSP (a -> SP a b) | NullSP
```

(a) Define the functions `mapTerm` and `foldTerm` for the data type `Term` which works in the same way on terms as `map` and `fold` on lists.

(b) Define the functions `mapSP` and `foldSP` for the data type `SP` which works in the same way as `map` and `fold` on lists.

(c) Define the function `symcount :: Term a -> Int` using `mapTerm` and/or `foldTerm` which returns the number of functions (`Add`, `Mul`) in a given term.

(d) Define the function `varcount :: Term a -> Int` using `mapTerm` and/or `foldTerm` which returns the number of variables (`V a`) in a given term.

(e) Define the function `substitute :: Term a -> [(Term a, Term a)] -> Term a` using `mapTerm` and/or `foldTerm`. `substitute` t `[(V` x_1, t_1`), (V` x_2, t_2`), ..., (V` x_n, t_n`)]` returns the term t where each occurrence of the variable `V` x_1 is replaced by the term t_1, each occurrence of the variable `V` x_2 is replaced by the term t_2, etc.

For example:
```
substitute (Mul (Add (C 1) (V 1)) (Mul (Add (V 1) (V 2)) (V 3)))
           [(V 1, C 2), (V 2, C 1)]
         = (Mul (Add (C 1) (C 2)) (Mul (Add (C 2) (C 1)) (V 3)))
```