

Prof. Dr. Jürgen Giesl
Peter Schneider-Kamp

Exercises *Functional Programming* – Sheet 8

Solutions will be collected until Wednesday, June 22, 2005 in the exercise course.

Exercises can be solved both in English and in German.

Exercise 1 (2 + 2 + 2 + 2 points)

We define the following algebraic data types and the following functions.

```
data Nats = Zero | Succ Nats
data List a = Nil | Cons a (List a)
```

```
pred :: Nats -> Nats
pred (Succ x) = x
```

```
isZero :: Nats -> Bool
isZero Zero = True
isZero _    = False
```

```
head :: List a -> a
head (Cons x _) = x
```

```
tail :: List a -> List a
tail (Cons _ xs) = xs
```

```
isNil :: List a -> Bool
isNil Nil = True
isNil _   = False
```

```
fst :: (a,b) -> a
fst (x,y) = x
```

```
snd :: (a,b) -> b
snd (x,y) = y
```

Give simple Haskell-expressions that are equivalent to the following Haskell-expressions. Your solutions may use some of the functions defined above. You do not have to use the transformation rules which will be presented later in the lecture.

```
(a) let sum' = \ys -> case ys of
      Nil           -> 0
      (Cons x xs)  -> x + sum' xs
    in sum' someList
```

```
(b) minus numberOne numberTwo
    where minus x      Zero      = x
          minus (Succ x) (Succ y) = minus x y
          minus _      _         = Zero
```

```
(c) minus (numberOne, NumberTwo)
    where minus (x      , Zero) = x
          minus (Succ x, Succ y) = minus (x, y)
          minus _      _         = Zero
```

```
(d) minus numberOne numberTwo
    where minus x      0      = x
          minus (x+1) (y+1) = minus x y
          minus _      _      = 0
```

Exercise 2 (2 + 3 + 4 points)

For the following Haskell expressions give the value of $\mathcal{Val}[\![\text{exp}_i]\!] \rho$ for $i \in \{1 \dots 3\}$, $\rho = \omega + \rho'$ and $\rho'(y) = \text{False}$, $\rho'(x) = 3$.

Describe your computation in detail and for each higher-order function $f : \text{Dom} \rightarrow \text{Dom}$ that occurs in the calculation, determine what the function $f^i(\perp)$, $i \in \mathbb{N}$, computes.

```
exp1 := let not' = \x -> if x == False then True else False
        in not' y
```

```
exp2 := let odd' = \x -> if x == 0 then False
                        else not (odd' (x - 1))
        in odd' 4
```

```
exp3 := let plus = \x -> \y if x == 0 then y
          else 1 + (plus (x-1) y)
        in plus 2 4
```