

Prof. Dr. Jürgen Giesl
Peter Schneider-Kamp
Stephan Swiderski

Exercises *Functional Programming* – Sheet 2

Solutions will be collected until Wednesday, Apr 25, 2007 in the exercise course.

Exercises can be solved both in English and in German.

Exercise 1 (2+2+3+2 points)

The following data structure represents lambda terms, i.e., terms build from lambda abstractions, applications, variables and function symbols:

```
data Term a = Var a | Funct a | App (Term a) (Term a) | Lam a (Term a)
```

Consider the following lambda term $t: \lambda x. ((f x) y)$. The term t could be represented as an object `t` of type `Term String` in Haskell:

```
Lam "x" (App (App (Funct "f") (Var "x"))) (Var "y"))
```

a) Give the most general type for the following Haskell expressions (where you may assume that `42` has type `Int`):

- (i) `App (Var 42)`
- (ii) `\x -> Lam x (Funct x)`
- (iii) `(\x -> App (App x x) x) (Var 42)`
- (iv) `(\x -> x) (\x -> Var)`

b) Make `Term` an instance of the `Show` class such that for the term `t` above the expression `show t` evaluates to `"\x". ((\"f\" \"x\") \"y\")`.

Hint: To output a backslash use the string `"\\"`.

c) Write the declarations (including a type declaration) for the function `getVars` that returns a list of all variables occurring in a given lambda term. For the term `t` above, the result would be `["x", "y"]` (or `["y", "x"]`). Your list should not contain duplicates. Note that `getVars (Lambda "x" (Funct "c"))` must return `["x"]` and not `[]`.

- d) Free variables are variables that are not bound by a lambda, i.e., for a term $\lambda x. s$ all variables of s are free except for x . Change the declarations from (c) such that `getVars` computes the free variables instead of all variables. For the term `t` above, the result would be `["y"]`.

Exercise 2 (1+3+2+1 points)

A substitution is a mapping σ from variables to terms (as defined in Exercise 1), such that σ maps all but finitely many variables to themselves. Because of this restriction, a substitution can be written as a (finite) list of pairs of variables and terms.

For example, a substitution mapping the variable x to the term t and the variable y to the term u , can be written as `[x/t, y/u]`.

- a) Define a Haskell type for substitutions using only a `type` declaration with built-in data types and the data structure `Term` from Exercise 1.
- b) Write the declarations (including a type declaration) for a function `apply` that takes a term t and a substitution σ and replaces all occurrences of the free variables in t . Here, any occurrence of the free variable x in t is replaced by the term $\sigma(x)$.

For example, when we apply the substitution `["y"/Func "c"]` to the term `t` from Exercise 1 we get a new term shown as `\ "x". (("f" "x") "c")`, i.e., represented by `Lam "x" (App (App (Func "f") (Var "x"))) (Func "c")`.

When applying the substitution `["x"/Func "c"]` to the term `t`, though, we get `\ "x". (("f" "x") "y")`, i.e., the term `t` remains unchanged.

- c) Define a data structure for substitutions like in (a) but without using any built-in data types. You may only use a single `data` declaration.
- d) Write a function like in (b) but using the data structure from (c).