University of Innsbruck **Institute of Computer Science**
1st Exam **January 23, 2009**

Functional Programming **WS 2008/2009** **LVA 703017**

Name: **Matr.Nr.:**

1. Consider the lambda-term $t = (\lambda xy.x\ (\lambda xy.y)\ y\ y)\ (\lambda xyz.z\ x\ y)\ (\lambda x.x)$.

[10] (a) Reduce t stepwise to normal form, using the leftmost innermost strategy.

[10] (b) Reduce t stepwise to normal form, using the leftmost outermost strategy.

[20] **2.** Consider the OCaml functions

```
let rec (@) xs ys = match xs with []    -> ys
                                | x::xs -> x::(xs @ ys)

let rec rev = function []    -> []
                     | x::xs -> (rev xs) @ [x]
```

Prove by induction that `rev`$(xs$ `@` $ys) = ($`rev` $ys)$ `@` $($`rev` $xs)$ for all lists xs and ys. You may use associativity of '`@`' and the fact that `[]` is a right identity of '`@`', i.e.,

$$(xs\ @\ ys)\ @\ zs = xs\ @\ (ys\ @\ zs) \qquad (\star)$$

$$xs\ @\ [\,] = xs \qquad (\star\star)$$

for all lists xs, ys, and zs.

3. Consider the OCaml functions `mem` and `unique`, defined by:

```
let rec mem y = function []    -> false
                       | x::xs -> x = y || mem y xs

let rec unique = function []    -> []
                        | x::xs -> if mem x xs then unique xs
                                                else x :: unique xs
```

[10] (a) Implement a tail-recursive variant of `unique`.

[10] (b) Use tupling to implement a function `percentage : 'a -> 'a list -> float` that determines for a given element x in a list xs the percentage it constitutes to the full list, e.g.,

`percentage 'a' ['a';'b';'c';'a']` $=$ `0.5`

4. Consider the λ-term $t = (\lambda x.y\ x)\ (\lambda yz.z\ y)\ w$.

[5] (a) Reduce t to normal form.

[5] (b) Give the set $\mathcal{FV}\mathsf{ar}(t)$ of free variables of t.

[5] (c) Give the set $\mathcal{BV}\mathsf{ar}(t)$ of bound variables of t.

[5] (d) Give the set $\mathcal{S}\mathsf{ub}(t)$ of all subterms of t.

5. Consider the typing environment

$$E = \{1 : \mathsf{int},\ + : \mathsf{int} \to \mathsf{int} \to \mathsf{int},\ \mathsf{p} : \mathsf{int} \to \mathsf{int} \to \mathsf{pair}(\mathsf{int}, \mathsf{int})\}.$$

[10] (a) Prove the typing judgment $E \vdash \textbf{let}\ x = 1\ \textbf{in}\ \mathsf{p}\ x\ (x + x) : \mathsf{pair}(\mathsf{int}, \mathsf{int})$.

[10] (b) Transform the type inference problem $E \rhd \textbf{let}\ x = 1\ \textbf{in}\ \mathsf{p}\ x\ (x + x) : \alpha_0$ into a unification problem.