University of Innsbruck **Institute of Computer Science**
1st Exam **February 5, 2010**

Functional Programming **WS 2009/2010** **LVA 703017**

Name: **Matr.Nr.:**

1. Consider the λ-term $t = (\lambda f\ x\ y.f\ ((\lambda x\ y\ p.p\ x\ y)\ x\ y))\ (\lambda q.q\ (\lambda u\ v.u))$.

[5] (a) Compute the sets $\mathcal{V}\mathsf{ar}(t)$, $\mathcal{BV}\mathsf{ar}(t)$, and $\mathcal{FV}\mathsf{ar}(t)$.

[5] (b) Draw the syntax tree of $\lambda q.q\ (\lambda u\ v.u)$.

[15] (c) Reduce t to normal form, using the leftmost innermost reduction strategy.

2. Consider the OCaml functions

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

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

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

Use induction over xs, to prove that `rev_append` $xs\ ys =$ `rev` xs `@` ys. Remember that `[`x`]` is just an abbreviation for x `::` `[]`. Additionally you may freely use the equation:

$$xs \mathbin{@} (ys \mathbin{@} zs) = (xs \mathbin{@} ys) \mathbin{@} zs \tag{1}$$

[5] (a) Give the base case of your induction proof.

[20] (b) Give the induction hypothesis and the step case of your induction proof.

3. Consider the OCaml functions

```
let rec count0 = function
  | []     -> 0
  | x::xs -> if x = 0 then 1 + count0 xs
                      else count0 xs

let rec count1 = function
  | []     -> 0
  | x::xs -> if x = 1 then 1 + count1 xs
                      else count1 xs
```

[12] (a) Use tupling to implement a function `count` that combines the effects of `count0` and `count1`.

[13] (b) Give a tail-recursive implementation of the function `count` from (a).

4. Consider the CoreML expression $e = (\lambda x.x \mathbin{@} x)$ together with the environment $E = \{@ : \mathsf{list}(\alpha_0) \to \mathsf{list}(\alpha_0) \to \mathsf{list}(\alpha_0)\}$.

[10] (a) Use type checking to prove that e has the type $\tau = \mathsf{list}(\alpha_0) \to \mathsf{list}(\alpha_0)$.

[15] (b) First transform $E \rhd e : \mathsf{list}(\mathsf{int}) \to \mathsf{list}(\mathsf{int})$ into a unification problem and then apply unification to infer whether e really has the type $\mathsf{list}(\mathsf{int}) \to \mathsf{list}(\mathsf{int})$.