University of Innsbruck | **Institute of Computer Science**
2nd Exam | **April 16, 2010**

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

Name: | **Matr.Nr.:**

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

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

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

2. Consider the OCaml type `type tree = E | N of tree * tree` together with the function

```
let rec mirror = function
  | E        -> E
  | N (l, r) -> N (mirror r, mirror l)
```

Prove by induction that `mirror` (`mirror` t) $= t$ for every value t of type `tree`.

[5] (a) Base case.

[20] (b) Step case.

3. Consider the OCaml function

```
let rec repeat i n =
  if n < 1 then []
           else i :: repeat i (n - 1)
```

[12] (a) Implement a tail-recursive variant of `repeat`.

[13] (b) Use tupling to implement a function `fraction : 'a -> 'a list -> float` that determines for a given element x in a list xs, the ratio (between 0 and 1) it constitutes to the whole list, e.g.,

`fraction 'a' ['a'; 'b'; 'c'; 'a'] = 0.5`

4. Consider the environment $E = \{1 : \mathsf{int}, 2 : \mathsf{int}, :: : \mathsf{int} \to \tau \to \tau, \mathsf{hd} : \tau \to \mathsf{int}, [] : \tau, \mathsf{tl} : \tau \to \tau\}$, where we use the abbreviation $\tau = \mathsf{list}(\mathsf{int})$ and :: is assumed to be a right-associative infix operator.

[12] (a) Prove the typing judgment $E \vdash \textbf{let}\ x = \mathsf{tl}\ (1 :: 2 :: [])\ \textbf{in}\ \mathsf{hd}\ x : \mathsf{int}$.

[13] (b) Solve the unification problem:

$$\begin{aligned} \alpha_3 \to \mathsf{list}(\alpha_3) \to \mathsf{list}(\alpha_3) &\approx \alpha_2 \to \alpha_1 \to \alpha_4; \\ \mathsf{bool} &\approx \alpha_2; \\ \mathsf{list}(\alpha_0) &\approx \alpha_1; \\ \mathsf{list}(\alpha_0) &\approx \alpha_4 \end{aligned}$$