University of Innsbruck | **Institute of Computer Science**
2nd Exam | **March 7, 2014**

Functional Programming | **WS 2013/2014** | **LVA 703024**

Solutions

The exam consists of 4 exercises. Explain your answers. You need 50 points to pass.

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

(8) (a) Reduce t to normal form. Give all β-steps!

Solution.

$$\begin{aligned} t &= \underline{(\lambda xy.y)\ ((\lambda xy.x\ x\ y)\ (\lambda x.x\ x))}\ (\lambda x.x\ x) \\ &\to_\beta \underline{(\lambda y.y)\ (\lambda x.x\ x)} \\ &\to_\beta \lambda x.x\ x \end{aligned}$$

(7) (b) Show that t admits an infinite $\to_\beta$ derivation.

Solution. Note that

$$\begin{aligned} t &= (\lambda xy.y)\ (\underline{(\lambda xy.x\ x\ y)\ (\lambda x.x\ x)})\ (\lambda x.x\ x) \\ &\to_\beta (\lambda xy.y)\ (\lambda y.\underline{(\lambda x.x\ x)\ (\lambda x.x\ x)}\ y)\ (\lambda x.x\ x) \\ &\to_\beta (\lambda xy.y)\ (\lambda y.\underline{(\lambda x.x\ x)\ (\lambda x.x\ x)}\ y)\ (\lambda x.x\ x) \\ &\to_\beta \ldots \end{aligned}$$

(10) (c) Complete the following table (correct: 1 point, wrong: 0 points):

λ-term	NF	WHNF
$x\ x$	✓	✗
x	✓	✓
$\lambda x.x$	✓	✓
$\lambda x.x\ y$	✓	✓
$(\lambda x.x)\ y$	✗	✗
$y\ (\lambda x.x)$	✓	✗

2. Consider the following OCaml type for binary trees

```
type 'a t = Empty | Node of ('a t * 'a * 'a t)
```

and the functions:

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

let rec mirror = function
 | Empty -> Empty
 | Node (l,x,r) -> Node(mirror r,x,mirror l)

let rec flatten = function
 | Empty       -> []
 | Node(l,v,r) -> (flatten l)@(v::flatten r)
```

Solutions

Prove by structural induction that for all binary trees t

$$\texttt{reverse}\ (\texttt{flatten}\ t) = \texttt{flatten}\ (\texttt{mirror}\ t)$$

You may use

$$xs\ \texttt{@}\ (ys\ \texttt{@}\ zs) = (xs\ \texttt{@}\ ys)\ \texttt{@}\ zs \qquad (\star)$$
$$\texttt{reverse}(xs\ \texttt{@}\ ys) = \texttt{reverse}\ ys\ \texttt{@}\ \texttt{reverse}\ xs \qquad (\star\star)$$

Hint: You may abbreviate `flatten` by `f`, `reverse` by `r`, and `mirror` by `m`.

(5) (a) Base case: Show the base case.

Solution. In the base case ($t = \texttt{Empty}$). We have to show

$$\texttt{reverse}\ (\texttt{flatten}\ \texttt{Empty}) = \texttt{flatten}\ (\texttt{mirror}\ \texttt{Empty})$$

We transform the lhs into the rhs:

$$\begin{aligned}
\texttt{reverse}\ (\texttt{flatten}\ \texttt{Empty}) &= \texttt{reverse}\ \texttt{[]} && \text{(definition of } \texttt{flatten}\text{)} \\
&= \texttt{[]} && \text{(definition of } \texttt{reverse}\text{)} \\
&= \texttt{flatten}\ \texttt{Empty} && \text{(definition of } \texttt{flatten}\text{)} \\
&= \texttt{flatten}\ (\texttt{mirror}\ \texttt{Empty}) && \text{(definition of } \texttt{mirror}\text{)}
\end{aligned}$$

(20) (b) Step case: Identify the property to prove (5 points), the induction hypothesis (5 points), and prove the step case (10 points).

Solution. In the step case ($t = \texttt{Node}(l, x, r)$).
The property to prove is

$$\texttt{reverse}\ (\texttt{flatten}\ (\texttt{Node}(l, x, r))) = \texttt{flatten}\ (\texttt{mirror}\ (\texttt{Node}(l, x, r)))\,.$$

The induction hypothesis is

$$\texttt{reverse}\ (\texttt{flatten}\ l) = \texttt{flatten}\ (\texttt{mirror}\ l)$$

and

$$\texttt{reverse}\ (\texttt{flatten}\ r) = \texttt{flatten}\ (\texttt{mirror}\ r)\,.$$

We transform the lhs into the rhs:

$$
\begin{aligned}
&\texttt{reverse}\ (\texttt{flatten}\ (\texttt{Node}(l, x, r))) \\
&= \texttt{reverse}\ (\texttt{flatten}\ l\ \texttt{@}\ x\ \texttt{::}\ \texttt{flatten}\ r) && \text{(definition of } \texttt{flatten}\text{)} \\
&= \texttt{reverse}\ (x\ \texttt{::}\ \texttt{flatten}\ r)\ \texttt{@}\ \texttt{reverse}\ (\texttt{flatten}\ l) && (\star\star) \\
&= (\texttt{reverse}\ (\texttt{flatten}\ r)\ \texttt{@}\ [x])\ \texttt{@}\ \texttt{reverse}\ (\texttt{flatten}\ l) && \text{(definition of } \texttt{reverse}\text{)} \\
&\overset{\text{IH}}{=} (\texttt{flatten}\ (\texttt{mirror}\ r)\ \texttt{@}\ [x])\ \texttt{@}\ \texttt{flatten}\ (\texttt{mirror}\ l) \\
&= \texttt{flatten}\ (\texttt{mirror}\ r)\ \texttt{@}\ ([x]\ \texttt{@}\ \texttt{flatten}\ (\texttt{mirror}\ l)) && (\star) \\
&= \texttt{flatten}\ (\texttt{mirror}\ r)\ \texttt{@}\ (x\ \texttt{::}\ \texttt{flatten}\ (\texttt{mirror}\ l)) && \text{(definition of } \texttt{@}\text{)} \\
&= \texttt{flatten}\ (\texttt{Node}\ (\texttt{mirror}\ r, x, \texttt{mirror}\ l)) && \text{(definition of } \texttt{flatten}\text{)} \\
&= \texttt{flatten}\ (\texttt{mirror}\ (\texttt{Node}(l, x, r))) && \text{(definition of } \texttt{mirror}\text{)}
\end{aligned}
$$

3. Consider the functions `reverse` and `flatten` from Exercise 2.

(10) (a) Determine for each of the two functions if they are tail recursive or not.

Solution. Both functions are not tail recursive. Both functions call `@` after the recursive calls.

(5) (b) Determine the runtime of `flatten` by means of O-notation.

Hint: Perform a worst-case analysis.

Solution. For a tree of size n there are $O(n)$ recursive calls to `flatten`. In each call we must append to the flattened left subtree the flattened right subtree, resulting in runtime $O(n^2)$ in total. Since `@` is defined by recursion over its first argument the worst case is when the tree is degenerated such that it consists of a left branch only.

(10) (c) Use parameter accumulation to implement a variant of `flatten` that runs in $O(n)$ time.

Solution.

```
let flatten_lin t =
 let rec flatten acc = function
  | Empty -> acc
  | Node(l,v,r) ->
   let acc = flatten acc r in
   flatten (v::acc) l
 in flatten [] t
;;
```

4. Consider the typing environment $E = \varnothing$ and the CoreML expression $e = \lambda f x.f\ (f\ x)$.

(10) (a) Transform the type inference problem $E \rhd e : \alpha_0$ into a unification problem.

Solution. We only show the result (many example solutions are in old exams):

$$\alpha_1 \approx \alpha_5 \to \alpha_4; \alpha_1 \approx \alpha_6 \to \alpha_5; \alpha_3 \approx \alpha_6; \alpha_2 \approx \alpha_3 \to \alpha_4; \alpha_0 \approx \alpha_1 \to \alpha_2$$

(10) (b) Solve (if possible) the unification problem

$$\alpha_1 \approx \alpha_5 \to \alpha_4; \alpha_1 \approx \alpha_6 \to \alpha_5; \alpha_3 \approx \alpha_6; \alpha_2 \approx \alpha_3 \to \alpha_4; \alpha_0 \approx \alpha_1 \to \alpha_2$$

Solution. We only show the result (many example solutions are in old exams).

$$\{\alpha_1/\alpha_6 \to \alpha_6, \alpha_5/\alpha_6, \alpha_4/\alpha_6, \alpha_3/\alpha_6, \alpha_2/\alpha_6 \to \alpha_6, \alpha_0/(\alpha_6 \to \alpha_6) \to \alpha_6 \to \alpha_6\}$$

(5) (c) Give an OCaml expression corresponding to the CoreML expression e.

Solution. `fun f x -> f (f x)`