University of Innsbruck | **Institute of Computer Science**
1st Exam | **February 1, 2013**

Functional Programming | **WS 2012/2013** | **LVA 703024**

Solutions

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

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

(8) (a) Reduce t to normal form using the leftmost outermost reduction strategy.

Solution.

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

(6) (b) Compute the free and the bound variables of t.

Solution. There are no free variables in t. The bound variables in t are x, y, z.

(5) (c) Give five subterms of t.

Solution. The subterms are t, $(\lambda xy.(\lambda z.z\ x)\ y)\ (\lambda x.x)$, $\lambda xy.(\lambda z.z\ x)\ y$, $\lambda y.(\lambda z.z\ x)\ y$, $(\lambda z.z\ x)\ y$, $\lambda z.z\ x$, $z\ x$, z, x, y, $\lambda x.x$, x, $\lambda x.x\ x$, $x\ x$, x, x.

(6) (d) Give a subterm of t that is in weak head normal form (WHNF) and one that is not.

Solution. The following subterms are in WHNF: $\lambda xy.(\lambda z.z\ x)\ y$, $\lambda y.(\lambda z.z\ x)\ y$, $\lambda z.z\ x$, z, x, y, $\lambda x.x$, and $\lambda x.x\ x$. The following subterms are not in WHNF: t, $(\lambda xy.(\lambda z.z\ x)\ y)\ (\lambda x.x)$, $(\lambda z.z\ x)\ y$, $z\ x$, and $x\ x$.

2. Consider the following OCaml functions:

```
let rec length = function []    -> 0
                        | _::xs -> 1 + length xs

let rec map f = function []    -> []
                       | x::xs -> f x::map f xs

let rec sum = function    [] -> 0
                    | x::xs -> x + sum xs

let succ x = x + 1
```

Prove by structural induction that for all integer lists xs

$$\texttt{sum}\ xs\ \texttt{+}\ \texttt{length}\ xs = \texttt{sum}\ (\texttt{map succ}\ xs)$$

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

Solution. In the base case ($xs = \texttt{[]}$).
Hence `sum [] + length [] = 0 = sum [] = sum (map succ [])` by the definition of `sum`, `length`, `map`, and `succ`.

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

Solution. In the step case ($xs = z :: zs$).
The induction hypothesis is

$$\texttt{sum}\ zs\ \texttt{+}\ \texttt{length}\ zs = \texttt{sum}\ (\texttt{map}\ \texttt{succ}\ zs)$$

We have to prove the property

$$\texttt{sum}\ (z :: zs)\ \texttt{+}\ \texttt{length}\ (z :: zs) = \texttt{sum}\ (\texttt{map}\ \texttt{succ}\ (z :: zs))$$

We show the property by transforming the lhs into the rhs.

$$\begin{aligned}
\texttt{sum}\ (z :: zs)\ \texttt{+}\ \texttt{length}\ (z :: zs) &= (z\ \texttt{+}\ \texttt{sum}\ zs)\ \texttt{+}\ \texttt{length}\ (z :: zs) && \text{(definition of sum)}\\
&= (z\ \texttt{+}\ \texttt{sum}\ zs)\ \texttt{+}\ (\texttt{1}\ \texttt{+}\ \texttt{length}\ zs) && \text{(definition of length)}\\
&= (z\ \texttt{+}\ \texttt{1})\ \texttt{+}\ (\texttt{sum}\ zs\ \texttt{+}\ \texttt{length}\ zs) && \\
& && \text{(associativity and commutativity of +)}\\
&\overset{\text{IH}}{=} (z\ \texttt{+}\ \texttt{1})\ \texttt{+}\ \texttt{sum}\ (\texttt{map}\ \texttt{succ}\ zs) && \\
&= \texttt{succ}\ z\ \texttt{+}\ \texttt{sum}\ (\texttt{map}\ \texttt{succ}\ zs) && \text{(definition of succ)}\\
&= \texttt{sum}\ (\texttt{succ}\ z :: \texttt{map}\ \texttt{succ}\ zs) && \text{(definition of sum)}\\
&= \texttt{sum}\ (\texttt{map}\ \texttt{succ}\ (z :: zs)) && \text{(definition of map)}
\end{aligned}$$

(10) **3.** (a) Implement a function `lensum` such that for all integer lists xs:

$$\texttt{lensum}\ xs = \texttt{sum}\ (\texttt{map}\ \texttt{succ}\ xs)$$

The function `lensum` may only traverse once over the list xs.

Solution.

```
let lensum = Lst.foldl (fun acc x -> (acc + succ x)) 0
```

(b) Determine for each of the following functions whether they are tail recursive or not:

(5) i.
```
let rec filter p = function
  | []    -> []
  | x::xs -> if p x then x::filter p xs else filter p xs
```

Solution. The function is not tail recursive, since if the predicate `p` is satisfied `::` is the last operation in the recursive call.

(5) ii.
```
let rec last = function
  | []    -> failwith "empty list"
  | [x]   -> x
  | _::xs -> last xs
```

Solution. The function is tail recursive, since nothing need to be done after the recursive call.

(5) iii.
```
let rec mem x = function []    -> false
                       | y::ys -> y = x || mem x ys
```

Hint: Note the (special) evaluation of `||`. If its left argument evaluates to **`true`** then immediately **`true`** is returned. Only otherwise its right argument is evaluated and returned.

Solution. The function is tail recursive, because of the special evaluation of `||`. If $x = y$ then `true` is returned. Only in the other case the recursive call is performed and its result is returned.

(13) **4.** (a) Consider the (empty) typing environment $E = \varnothing$. Transform the type inference problem $E \rhd \lambda f x.f\ x\ x : \alpha_0$ into a unification problem.

Solution.

$$E \rhd \lambda f x.f\ x\ x : \alpha_0$$
$$\overset{\mathsf{abs}}{\Rightarrow}$$
$$E, f : \alpha_1 \rhd \lambda x.f\ x\ x : \alpha_2; \alpha_0 \approx \alpha_1 \to \alpha_2$$
$$\overset{\mathsf{abs}}{\Rightarrow}$$
$$E, f : \alpha_1, x : \alpha_3 \rhd f\ x\ x : \alpha_4; \alpha_2 \approx \alpha_3 \to \alpha_4; \alpha_0 \approx \alpha_1 \to \alpha_2$$
$$\overset{\mathsf{app}}{\Rightarrow}$$
$$E, f : \alpha_1, x : \alpha_3 \rhd f\ x : \alpha_5 \to \alpha_4; E, f : \alpha_1, x : \alpha_3 \rhd x : \alpha_5; \alpha_2 \approx \alpha_3 \to \alpha_4; \alpha_0 \approx \alpha_1 \to \alpha_2$$
$$\overset{\mathsf{app}}{\Rightarrow}$$
$$E, f : \alpha_1, x : \alpha_3 \rhd f : \alpha_6 \to \alpha_5 \to \alpha_4; E, f : \alpha_1, x : \alpha_3 \rhd x : \alpha_6; E, f : \alpha_1, x : \alpha_3 \rhd x : \alpha_5; \alpha_2 \approx \alpha_3 \to \alpha_4; \alpha_0 \approx \alpha_1 \to \alpha_2$$
$$\overset{\mathsf{con}}{\Rightarrow}$$
$$\alpha_1 \approx \alpha_6 \to \alpha_5 \to \alpha_4; E, f : \alpha_1, x : \alpha_3 \rhd x : \alpha_6; E, f : \alpha_1, x : \alpha_3 \rhd x : \alpha_5; \alpha_2 \approx \alpha_3 \to \alpha_4; \alpha_0 \approx \alpha_1 \to \alpha_2$$
$$\overset{\mathsf{con}}{\Rightarrow}$$
$$\alpha_1 \approx \alpha_6 \to \alpha_5 \to \alpha_4; \alpha_3 \approx \alpha_6; E, f : \alpha_1, x : \alpha_3 \rhd x : \alpha_5; \alpha_2 \approx \alpha_3 \to \alpha_4; \alpha_0 \approx \alpha_1 \to \alpha_2$$
$$\overset{\mathsf{con}}{\Rightarrow}$$
$$\alpha_1 \approx \alpha_6 \to \alpha_5 \to \alpha_4; \alpha_3 \approx \alpha_6; \alpha_3 \approx \alpha_5; \alpha_2 \approx \alpha_3 \to \alpha_4; \alpha_0 \approx \alpha_1 \to \alpha_2$$

(12) (b) Solve (if possible) the unification problem:

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

Justify your answer.

Solution.

$$\begin{aligned}
&\alpha_3 \approx \alpha_5; \alpha_3 \approx \alpha_6; \alpha_1 \approx \alpha_6 \to \alpha_5 \to \alpha_4; \alpha_2 \approx \alpha_3 \to \alpha_4; \alpha_0 \approx \alpha_1 \to \alpha_2 \\
\overset{(\mathsf{v}_1)}{\Rightarrow}_{\{\alpha_3/\alpha_5\}} &\quad \alpha_5 \approx \alpha_6; \alpha_1 \approx \alpha_6 \to \alpha_5 \to \alpha_4; \alpha_2 \approx \alpha_5 \to \alpha_4; \alpha_0 \approx \alpha_1 \to \alpha_2 \\
\overset{(\mathsf{v}_1)}{\Rightarrow}_{\{\alpha_5/\alpha_6\}} &\quad \alpha_1 \approx \alpha_6 \to \alpha_6 \to \alpha_4; \alpha_2 \approx \alpha_6 \to \alpha_4; \alpha_0 \approx \alpha_1 \to \alpha_2 \\
\overset{(\mathsf{v}_1)}{\Rightarrow}_{\{\alpha_1/\alpha_6 \to \alpha_6 \to \alpha_4\}} &\quad \alpha_2 \approx \alpha_6 \to \alpha_4; \alpha_0 \approx (\alpha_6 \to \alpha_6 \to \alpha_4) \to \alpha_2 \\
\overset{(\mathsf{v}_1)}{\Rightarrow}_{\{\alpha_2/\alpha_6 \to \alpha_4\}} &\quad \alpha_0 \approx (\alpha_6 \to \alpha_6 \to \alpha_4) \to \alpha_6 \to \alpha_4 \\
\overset{(\mathsf{v}_1)}{\Rightarrow}_{\{\alpha_0/(\alpha_6 \to \alpha_6 \to \alpha_4) \to \alpha_6 \to \alpha_4\}} &\quad \square
\end{aligned}$$

A solution to the unification problem is the unifier

$$\begin{aligned}
\sigma &= \{\alpha_3/\alpha_5\}\{\alpha_5/\alpha_6)\}\{\alpha_1/\alpha_6 \to \alpha_6 \to \alpha_4\}\{\alpha_2/\alpha_6 \to \alpha_4\}\{\alpha_0/(\alpha_6 \to \alpha_6 \to \alpha_4) \to \alpha_6 \to \alpha_4\} \\
&= \{\alpha_3/\alpha_6, \alpha_5/\alpha_6, \alpha_1/\alpha_6 \to \alpha_6 \to \alpha_4, \alpha_2/\alpha_6 \to \alpha_4, \alpha_0/(\alpha_6 \to \alpha_6 \to \alpha_4) \to \alpha_6 \to \alpha_4\}.
\end{aligned}$$