University of Innsbruck | **Institute of Computer Science**
3rd Exam | **September 30, 2013**

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

Solutions

(10) **1.** (a) Write down the Church numerals for the natural numbers 0 and 3.

Solution. $\overline{0} = \lambda f x.x$ and $\overline{3} = \lambda f x.f\ (f\ (f\ x))$.

(15) (b) Reduce the lambda term $(\lambda m n f x.m\ f\ (n\ f\ x))\ (\lambda f x.f\ x)\ (\lambda f x.f\ (f\ x))$ to normal form.

Solution.

$$\begin{aligned}
\mathsf{add}\ \overline{1}\ \overline{2} &= (\lambda m n f x.m\ f\ (n\ f\ x))\ (\lambda f x.f\ x)\ (\lambda f x.f\ (f\ x)) \\
&\to_\beta (\lambda n f x.(\lambda f x.f\ x)\ f\ (n\ f\ x))\ (\lambda f x.f\ (f\ x)) \\
&\to_\beta \lambda f x.(\lambda f x.f\ x)\ f\ ((\lambda f x.f\ (f\ x))\ f\ x) \\
&\to_\beta \lambda f x.(\lambda x.f\ x)\ ((\lambda f x.f\ (f\ x))\ f\ x) \\
&\to_\beta \lambda f x.f\ ((\lambda f x.f\ (f\ x))\ f\ x) \\
&\to_\beta \lambda f x.f\ ((\lambda x.f\ (f\ x))\ x) \\
&\to_\beta \lambda f x.f\ (f\ (f\ x)) = \overline{3}
\end{aligned}$$

2. Consider the functions `length` defined by

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

and '`@`' defined by

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

Prove by structural induction on xs that for all lists xs and ys

$$\texttt{length}\ xs + \texttt{length}\ ys = \texttt{length}\ (xs\ \texttt{@}\ ys)$$

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

Solution. In the base case $xs = \texttt{[]}$. We have

$$\begin{aligned}
\texttt{length []} + \texttt{length}\ ys &= \texttt{0} + \texttt{length}\ ys && \text{(definition of length)} \\
&= \texttt{length}\ ys && \text{(definition of '+')} \\
&= \texttt{length}\ (\texttt{[]}\ \texttt{@}\ ys) && \text{(definition of '@')}
\end{aligned}$$

(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 IH is

$$\texttt{length}\ zs + \texttt{length}\ ys\ =\ \texttt{length}\ (zs\ \texttt{@}\ ys).$$

We have to prove

$$\texttt{length}\ (z :: zs) + \texttt{length}\ ys\ =\ \texttt{length}\ ((z :: zs)\ \texttt{@}\ ys).$$

For the lhs we get:

$$\begin{aligned} \texttt{length}\ (z :: zs) + \texttt{length}\ ys &= (1 + length\ zs) + \texttt{length}\ ys && \text{(definition of } \texttt{length}\text{)} \\ &= 1 + (\texttt{length}\ zs + \texttt{length}\ ys) && \text{(associativity of '+')} \\ &\overset{\text{IH}}{=} 1 + \texttt{length}\ (zs\ \texttt{@}\ ys) \end{aligned}$$

For the rhs we get:

$$\begin{aligned} \texttt{length}\ ((z :: zs)\ \texttt{@}\ ys) &= \texttt{length}\ (z :: (zs\ \texttt{@}\ ys)) && \text{(definition of '@')} \\ &= 1 + \texttt{length}\ (zs\ \texttt{@}\ xs) && \text{(definition of } \texttt{length}\text{)} \end{aligned}$$

3. Consider the functions `foldl` defined as

```
let rec foldl f b = function []    -> b
                           | x::xs -> foldl f (f b x) xs
```

and `foldr` defined as

```
let rec foldr f b = function []     -> b
                           | x::xs -> f x (foldr f b xs)
```

and `reverse` defined as

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

(15) (a) Which one is tail recursive and which one is not? Justify your answer.

Solution. The funtion `foldl` is tail recursive since its last action is the recursive call. The function `foldr` is not tail recursive since the last action is calling `f`. The function `reverse` is not tail recursive since the last action is calling `@`.

(10) (b) Give tail recursive definitions for those functions which are not tail recursive.

Solution.

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

let foldr_tl f b xs = foldl (fun x y -> f y x) b (rev xs);;
```

(13) **4.** (a) Consider the typing environment

$$E = \{+ : \mathsf{int} \to \mathsf{int} \to \mathsf{int}, 2 : \mathsf{int}, 3 : \mathsf{int}\}.$$

Transform the type inference problem

$$E \rhd \textbf{let}\ x = 3\ \textbf{in}\ +\ x\ 2 : \alpha_0$$

into a unification problem.

Solution.

$$E \rhd \textbf{let } x = 3 \textbf{ in } + \; x \; 2 : \alpha_0$$
$$\overset{\mathsf{let}}{\Rightarrow}$$
$$E \rhd 3 : \alpha_1; E, x : \alpha_1 \rhd + \; x \; 2 : \alpha_0$$
$$\overset{\mathsf{con}}{\Rightarrow}$$
$$\mathsf{int} \approx \alpha_1; E, x : \alpha_1 \rhd + \; x \; 2 : \alpha_0$$
$$\overset{\mathsf{app}}{\Rightarrow}$$
$$\mathsf{int} \approx \alpha_1; E, x : \alpha_1 \rhd + \; x : \alpha_2 \to \alpha_0; E, x : \alpha_1 \rhd 2 : \alpha_2$$
$$\overset{\mathsf{app}}{\Rightarrow}$$
$$\mathsf{int} \approx \alpha_1; E, x : \alpha_1 \rhd + : \alpha_3 \to \alpha_2 \to \alpha_0; E, x : \alpha_1 \rhd x : \alpha_3; E, x : \alpha_1 \rhd 2 : \alpha_2$$
$$\overset{\mathsf{con}}{\Rightarrow}$$
$$\mathsf{int} \approx \alpha_1; \mathsf{int} \to \mathsf{int} \to \mathsf{int} \approx \alpha_3 \to \alpha_2 \to \alpha_0; E, x : \alpha_1 \rhd x : \alpha_3; E, x : \alpha_1 \rhd 2 : \alpha_2$$
$$\overset{\mathsf{con}}{\Rightarrow}$$
$$\mathsf{int} \approx \alpha_1; \mathsf{int} \to \mathsf{int} \to \mathsf{int} \approx \alpha_3 \to \alpha_2 \to \alpha_0; \alpha_1 \approx \alpha_3; E, x : \alpha_1 \rhd 2 : \alpha_2$$
$$\overset{\mathsf{con}}{\Rightarrow}$$
$$\mathsf{int} \approx \alpha_1; \mathsf{int} \to \mathsf{int} \to \mathsf{int} \approx \alpha_3 \to \alpha_2 \to \alpha_0; \alpha_1 \approx \alpha_3; \mathsf{int} \approx \alpha_2$$

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

$$\mathsf{int} \approx \alpha_1; \mathsf{int} \to \mathsf{int} \to \mathsf{int} \approx \alpha_3 \to \alpha_2 \to \alpha_0; \alpha_1 \approx \alpha_3; \mathsf{int} \approx \alpha_2$$

Justify your answer.

Solution.

$$\begin{array}{ll}
& \mathsf{int} \approx \alpha_1; \mathsf{int} \to \mathsf{int} \to \mathsf{int} \approx \alpha_3 \to \alpha_2 \to \alpha_0; \alpha_1 \approx \alpha_3; \mathsf{int} \approx \alpha_2 \\
\Rightarrow^{(\mathsf{v}_2)}_{\{\alpha_1/\mathsf{int}\}} & \mathsf{int} \to \mathsf{int} \to \mathsf{int} \approx \alpha_3 \to \alpha_2 \to \alpha_0; \mathsf{int} \approx \alpha_3; \mathsf{int} \approx \alpha_2 \\
\Rightarrow^{(\mathsf{d}_2)}_{\iota} & \mathsf{int} \approx \alpha_3; \mathsf{int} \to \mathsf{int} \approx \alpha_2 \to \alpha_0; \mathsf{int} \approx \alpha_3; \mathsf{int} \approx \alpha_2 \\
\Rightarrow^{(\mathsf{v}_2)}_{\{\alpha_3/\mathsf{int}\}} & \mathsf{int} \to \mathsf{int} \approx \alpha_2 \to \alpha_0; \mathsf{int} \approx \mathsf{int}; \mathsf{int} \approx \alpha_2 \\
\Rightarrow^{(\mathsf{d}_2)}_{\iota} & \mathsf{int} \approx \alpha_2; \mathsf{int} \approx \alpha_0; \mathsf{int} \approx \mathsf{int}; \mathsf{int} \approx \alpha_2 \\
\Rightarrow^{(\mathsf{v}_2)}_{\{\alpha_2/\mathsf{int}\}} & \mathsf{int} \approx \alpha_0; \mathsf{int} \approx \mathsf{int}; \mathsf{int} \approx \mathsf{int} \\
\Rightarrow^{(\mathsf{v}_2)}_{\{\alpha_0/\mathsf{int}\}} & \mathsf{int} \approx \mathsf{int}; \mathsf{int} \approx \mathsf{int} \\
\Rightarrow^{(\mathsf{t})}_{\{\iota\}} & \mathsf{int} \approx \mathsf{int} \\
\Rightarrow^{(\mathsf{t})}_{\{\iota\}} & \square
\end{array}$$

A solution to the unification problem is the unifier

$$\begin{aligned}
\sigma &= \{\alpha_1/\mathsf{int}\}\{\alpha_3/\mathsf{int}\}\{\alpha_2/\mathsf{int}\}\{\alpha_0/\mathsf{int}\} \\
&= \{\alpha_1/\mathsf{int}, \alpha_3/\mathsf{int}, \alpha_2/\mathsf{int}, \alpha_0/\mathsf{int}\}.
\end{aligned}$$