University of Innsbruck | **Institute of Computer Science**
2nd Exam | **March 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-terms $\mathsf{if} = (\lambda xyz.x\ y\ z)$, $\mathsf{not} = (\lambda fxy.f\ y\ x)$, $\mathsf{true} = (\lambda xz.x)$, $\overline{1} = (\lambda fx.f\ x)$, and $\overline{0} = (\lambda fx.x)$.
Let $t = \mathsf{if}\ (\mathsf{not}\ \mathsf{true})\ \overline{1}\ \overline{0}$.

(14) (a) Reduce t to normal form using the leftmost innermost reduction strategy. Give all β-steps!

Solution.

$$\begin{aligned}
\mathsf{if}\ (\mathsf{not}\ \mathsf{true})\ \overline{1}\ \overline{0} &= (\lambda xyz.x\ y\ z)\ ((\lambda fxy.f\ y\ x)\ (\lambda xz.x))\ (\lambda fx.f\ x)\ (\lambda fx.x)\\
&\to_\beta (\lambda xyz.x\ y\ z)\ (\lambda xy.(\lambda xz.x)\ y\ x)\ (\lambda fx.f\ x)\ (\lambda fx.x)\\
&\to_\beta (\lambda xyz.x\ y\ z)\ (\lambda xy.(\lambda z.y)\ x)\ (\lambda fx.f\ x)\ (\lambda fx.x)\\
&\to_\beta (\lambda xyz.x\ y\ z)\ (\lambda xy.y)\ (\lambda fx.f\ x)\ (\lambda fx.x)\\
&\to_\beta (\lambda yz.(\lambda xy.y)\ y\ z)\ (\lambda fx.f\ x)\ (\lambda fx.x)\\
&\to_\beta (\lambda yz.(\lambda y.y)\ z)\ (\lambda fx.f\ x)\ (\lambda fx.x)\\
&\to_\beta (\lambda yz.z)\ (\lambda fx.f\ x)\ (\lambda fx.x)\\
&\to_\beta (\lambda z.z)\ (\lambda fx.x)\\
&\to_\beta \lambda fx.x = \overline{0}
\end{aligned}$$

(5) (b) Give five different, non-variable subterms of t.

Solution. The non-variable subterms of t are

$$\{t, \mathsf{if}\ (\mathsf{not}\ \mathsf{true})\ \overline{1}, \mathsf{if}\ (\mathsf{not}\ \mathsf{true}), \mathsf{not}\ \mathsf{true}, \mathsf{if}, \mathsf{not}, \mathsf{true}, \overline{1}, \overline{0}, x\ y\ z, x\ y, f\ y\ x, f\ y, f\ x\}.$$

(6) (c) Give three different subterms of t which are not in weak head normal form.

Solution. The subterms of t which are not in weak head normal form are

$$\{t, \mathsf{if}\ (\mathsf{not}\ \mathsf{true})\ \overline{1}, \mathsf{if}\ (\mathsf{not}\ \mathsf{true}), \mathsf{not}\ \mathsf{true}, x\ y\ z, x\ y, f\ y\ x, f\ y, f\ x\}.$$

2. Consider the functions '`@`', `app2`, and `foldr` defined by

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

let app2 xs ys = foldr (fun x xs -> x::xs) ys xs

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

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

$$xs\ \texttt{@}\ ys = \texttt{app2}\ xs\ ys\,.$$

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

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

$$\begin{aligned} \texttt{[]} \mathbin{\texttt{@}} ys &= ys && \text{(definition of `@`)} \\ &= \texttt{foldr (fun x xs -> x::xs)}\ ys\ \texttt{[]} && \text{(definition of foldr)} \\ &= \texttt{app2}\ \texttt{[]}\ ys && \text{(definition of app2)} \end{aligned}$$

(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

$$zs \mathbin{\texttt{@}} ys \ = \ \texttt{app2}\ zs\ ys.$$

We have to prove

$$(z :: zs) \mathbin{\texttt{@}} ys \ = \ \texttt{app2}\ (z :: zs)\ ys.$$

For the lhs we get:

$$(z :: zs) \mathbin{\texttt{@}} ys = z :: (zs \mathbin{\texttt{@}} ys) \qquad \text{(definition of `@`)}$$

For the rhs we get:

$$\begin{aligned} \texttt{app2}\ (z :: zs)\ ys &= \texttt{foldr (fun x xs -> x::xs)}\ ys\ (z :: zs) && \text{(definition of app2)} \\ &= \texttt{(fun x xs -> x::xs)}\ z \\ &\quad\ (\texttt{foldr}\ (\texttt{fun x xs -> x::xs})\ ys\ zs) && \text{(definition of foldr)} \\ &= z :: \texttt{foldr}\ (\texttt{fun x xs -> x::xs})\ ys\ zs \\ & && \text{(definition of anonymous function)} \\ &= z :: (\texttt{app2}\ zs\ ys\ ) && \text{(definition app2)} \\ &\overset{\text{IH}}{=} z :: (zs \mathbin{\texttt{@}} ys) \end{aligned}$$

3. Consider the following OCaml functions:

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

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

let rec prod = function []    -> 1
                      | x::xs -> x * prod xs
```

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

Solution. The function `length` is not tail recursive since the last action is adding 1. The funtion `reverse` is not tail recursive since its last action is the call to `@`. The function `prod` is not tail recursive since the last action is the multiplication.

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

Solution.

```
let length_tl xs =
 let rec length acc = function
  | [] -> acc
  | _ :: xs -> length (1 + acc) xs
 in length 0 xs


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

let prod_tl = Lst.foldl ( * ) 1
```

(13) **4.** (a) Consider the typing environment $E = \{mult : \mathsf{int} \to \mathsf{int} \to \mathsf{int}, 2 : \mathsf{int}\}$. Prove the judgement

$$E \vdash \lambda x.mult\ 2\ x : \mathsf{int} \to \mathsf{int}.$$

Solution.

$$\dfrac{\dfrac{\dfrac{}{E, x : \mathsf{int} \vdash mult : \mathsf{int} \to \mathsf{int} \to \mathsf{int}}\,(\mathsf{ref}) \quad \dfrac{}{E, x : \mathsf{int} \vdash 2 : \mathsf{int}}\,(\mathsf{ref})}{E, x : \mathsf{int} \vdash mult\ 2 : \mathsf{int} \to \mathsf{int}}\,(\mathsf{app}) \quad \dfrac{}{E, x : \mathsf{int} \vdash x : \mathsf{int}}\,(\mathsf{ref})}{\dfrac{E, x : \mathsf{int} \vdash mult\ 2\ x : \mathsf{int}}{E \vdash \lambda x.mult\ 2\ x : \mathsf{int} \to \mathsf{int}.}\,(\mathsf{abs})}\,(\mathsf{app})$$

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

$$\mathsf{int} \approx \alpha_1; \mathsf{list}(\beta_0) \approx \alpha_4; \alpha_2 \approx \alpha_5; \beta_1 \to \mathsf{list}(\beta_1) \to \mathsf{list}(\beta_1) \approx \alpha_5 \to \alpha_4 \to \alpha_3; \alpha_1 \to \alpha_0 \approx \mathsf{int} \to \alpha_3$$

Justify your answer.

Solution.

$$\begin{aligned}
&\mathsf{int} \approx \alpha_1; \mathsf{list}(\beta_0) \approx \alpha_4; \alpha_2 \approx \alpha_5; \beta_1 \to \mathsf{list}(\beta_1) \to \mathsf{list}(\beta_1) \approx \alpha_5 \to \alpha_4 \to \alpha_3; \alpha_1 \to \alpha_0 \approx \mathsf{int} \to \alpha_3 \\
\Rightarrow^{(\mathsf{v}_2)}_{\{\alpha_1/\mathsf{int}\}} \quad & \mathsf{list}(\beta_0) \approx \alpha_4; \alpha_2 \approx \alpha_5; \beta_1 \to \mathsf{list}(\beta_1) \to \mathsf{list}(\beta_1) \approx \alpha_5 \to \alpha_4 \to \alpha_3; \mathsf{int} \to \alpha_0 \approx \mathsf{int} \to \alpha_3 \\
\Rightarrow^{(\mathsf{d}_2)}_{\iota} \quad & \mathsf{list}(\beta_0) \approx \alpha_4; \alpha_2 \approx \alpha_5; \beta_1 \to \mathsf{list}(\beta_1) \to \mathsf{list}(\beta_1) \approx \alpha_5 \to \alpha_4 \to \alpha_3; \mathsf{int} \approx \mathsf{int}; \alpha_0 \approx \alpha_3 \\
\Rightarrow^{(\mathsf{t})}_{\iota} \quad & \mathsf{list}(\beta_0) \approx \alpha_4; \alpha_2 \approx \alpha_5; \beta_1 \to \mathsf{list}(\beta_1) \to \mathsf{list}(\beta_1) \approx \alpha_5 \to \alpha_4 \to \alpha_3; \alpha_0 \approx \alpha_3 \\
\Rightarrow^{(\mathsf{v}_1)}_{\{\alpha_2/\alpha_5\}} \quad & \mathsf{list}(\beta_0) \approx \alpha_4; \beta_1 \to \mathsf{list}(\beta_1) \to \mathsf{list}(\beta_1) \approx \alpha_5 \to \alpha_4 \to \alpha_3; \alpha_0 \approx \alpha_3 \\
\Rightarrow^{(\mathsf{v}_1)}_{\{\alpha_0/\alpha_3\}} \quad & \mathsf{list}(\beta_0) \approx \alpha_4; \beta_1 \to \mathsf{list}(\beta_1) \to \mathsf{list}(\beta_1) \approx \alpha_5 \to \alpha_4 \to \alpha_3 \\
\Rightarrow^{(\mathsf{d}_2)}_{\iota} \quad & \mathsf{list}(\beta_0) \approx \alpha_4; \beta_1 \approx \alpha_5; \mathsf{list}(\beta_1) \to \mathsf{list}(\beta_1) \approx \alpha_4 \to \alpha_3 \\
\Rightarrow^{(\mathsf{v}_1)}_{\{\beta_1/\alpha_5\}} \quad & \mathsf{list}(\beta_0) \approx \alpha_4; \mathsf{list}(\alpha_5) \to \mathsf{list}(\alpha_5) \approx \alpha_4 \to \alpha_3 \\
\Rightarrow^{(\mathsf{d}_2)}_{\iota} \quad & \mathsf{list}(\beta_0) \approx \alpha_4; \mathsf{list}(\alpha_5) \approx \alpha_4; \mathsf{list}(\alpha_5) \approx \alpha_3 \\
\Rightarrow^{(\mathsf{v}_2)}_{\{\alpha_4/\mathsf{list}(\alpha_5)\}} \quad & \mathsf{list}(\beta_0) \approx \mathsf{list}(\alpha_5); \mathsf{list}(\alpha_5) \approx \alpha_3 \\
\Rightarrow^{(\mathsf{d}_1)}_{\iota} \quad & \beta_0 \approx \alpha_5; \mathsf{list}(\alpha_5) \approx \alpha_3 \\
\Rightarrow^{(\mathsf{v}_1)}_{\{\beta_0/\alpha_5\}} \quad & \mathsf{list}(\alpha_5) \approx \alpha_3 \\
\Rightarrow^{(\mathsf{v}_2)}_{\{\alpha_3/\mathsf{list}(\alpha_5)\}} \quad & \square
\end{aligned}$$

Solutions

A solution to the unification problem is the unifier

$$\begin{aligned}\sigma &= \{\alpha_1/\mathsf{int}\}\{\alpha_2/\alpha_5\}\{\alpha_0/\alpha_3\}\{\beta_1/\alpha_5\}\{\alpha_4/\mathsf{list}(\alpha_5)\}\{\beta_0/\alpha_5\}\{\alpha_3/\mathsf{list}(\alpha_5)\} \\ &= \{\alpha_1/\mathsf{int}, \alpha_2/\alpha_5, \alpha_0/\mathsf{list}(\alpha_5), \beta_1/\alpha_5, \alpha_4/\mathsf{list}(\alpha_5), \beta_0/\alpha_5, \alpha_3/\mathsf{list}(\alpha_5)\}.\end{aligned}$$