universität innsbruck

Functional Programming | WS 2017/2018 | LVA 703024

EXAM 2 | March 2, 2018

NAME: **MATRICULATION NUMBER:**

1 (a)

base case

We employ structural induction over xs. The base case, where $xs = \texttt{[]}$, is taken care of by the derivation:

$$\texttt{length}\ (\texttt{map}\ f\ \texttt{[]}) = \texttt{length}\ \texttt{[]} \qquad \text{(definition of } \texttt{map}\text{)}$$

(b)

induction hypothesis and statement to show in step case

In the step case $xs = z : zs$ for some element z and list zs. The induction hypothesis (IH) is

$$\texttt{length}\ (\texttt{map}\ f\ zs) = \texttt{length}\ zs$$

and the statement we have to show in order to conclude the step case is

$$\texttt{length}\ (\texttt{map}\ f\ (z : zs)) = \texttt{length}\ (z : zs)$$

(c)

step case

We prove the step case by the following derivation

$$\begin{aligned}
\texttt{length}\ (\texttt{map}\ f\ (z : zs)) &= \texttt{length}\ (f\ z : \texttt{map}\ f\ zs) && \text{(definition of } \texttt{map}\text{)} \\
&= 1 + \texttt{length}\ (\texttt{map}\ f\ zs) && \text{(definition of } \texttt{length}\text{)} \\
&= 1 + \texttt{length}\ zs && \text{(IH)} \\
&= \texttt{length}\ (z : zs) && \text{(definition of } \texttt{length}\text{)}
\end{aligned}$$

2 (a) *reduction to normal form*

The contracted redex of each β-step is underlined:

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

(b) *has normal form*

Complete the following table using ✔/✗:

λ-term	has normal form
$t_1\ t_1$	✗
$t_2\ t_2$	✔
$t_3\ t_3$	✔
$t_4\ t_4$	✗

(c) *alpha equivalence*

Complete the following table using ✔/✗:

$\equiv_\alpha$	t_1	t_2	t_3	t_4
t_1	✔	✗	✗	✔
t_2	✗	✔	✗	✗
t_3	✗	✗	✔	✗
t_4	✔	✗	✗	✔

3 (a) *equational reasoning*

```
map ((-) 1) [1,2] = 1 - 1 : map ((-) 1) [2]
                  = 1 - 1 : 1 - 2 : map ((-) 1) []
                  = 1 - 1 : 1 - 2 : []
                  = 0 : 1 - 2 : []
                  = 0 : -1 : []
                  = [0,-1]
```

(b) *kinds of recursion*

The function `map` is not tail recursive because after the recursive call a new list is constructed. Moreover, the function `map` is guardedly recursive because the only action after the recursive call is constructing data, i.e., the recursive call is "guarded" by one application of the list constructor '`:`'.

(c) *a tail recursive implementation*

```haskell
maptl f = mt []
  where
    -- the accumulator collects elements in reverse order,
    -- thus, at the very end, we reverse again
    mt acc [] = rev [] acc
    mt acc (x:xs) = mt (f x : acc) xs
    -- a tail recursive reverse function
    rev acc [] = acc
    rev acc (x:xs) = rev (x : acc) xs
```

4 (a) *Not typeable*

The core FP expression 1 1 is not typeable with respect to P, where $1 :: \mathsf{Int}$.

(b) *type inference part 1 – typing constraints*

$$\underline{E \rhd z\ (\lambda x.\, x) :: \alpha_0}$$
$$\Rightarrow^{(\mathsf{app})}$$
$$\underline{E \rhd z :: \alpha_1 \to \alpha_0}; E \rhd \lambda x.\, x :: \alpha_1$$
$$\Rightarrow^{(\mathsf{con})}$$
$$\mathsf{Int} \approx \alpha_1 \to \alpha_0; \underline{E \rhd \lambda x.\, x :: \alpha_1}$$
$$\Rightarrow^{(\mathsf{abs})}$$
$$\mathsf{Int} \approx \alpha_1 \to \alpha_0; \alpha_1 \approx \alpha_2 \to \alpha_3; \underline{E, x :: \alpha_2 \rhd x :: \alpha_3}$$
$$\Rightarrow^{(\mathsf{con})}$$
$$\mathsf{Int} \approx \alpha_1 \to \alpha_0; \alpha_1 \approx \alpha_2 \to \alpha_3; \alpha_2 \approx \alpha_3$$

(c) *type inference part 2 – unification*

$$\underline{\alpha_2 \approx \alpha_1 \to \alpha_0}; \alpha_3 \approx \mathsf{Int} \to \alpha_2; \alpha_2 \approx \alpha_3$$
$$\Rightarrow^{(\mathsf{v}_1)}_{\{\alpha_2 \mapsto \alpha_1 \to \alpha_0\}}$$
$$\underline{\alpha_3 \approx \mathsf{Int} \to \alpha_1 \to \alpha_0}; \alpha_1 \to \alpha_0 \approx \alpha_3$$
$$\Rightarrow^{(\mathsf{v}_1)}_{\{\alpha_3 \mapsto \mathsf{Int} \to \alpha_1 \to \alpha_0\}}$$
$$\underline{\alpha_1 \to \alpha_0 \approx \mathsf{Int} \to \alpha_1 \to \alpha_0}$$
$$\Rightarrow^{(\mathsf{d}_2)}_{\{\}}$$
$$\underline{\alpha_1 \approx \mathsf{Int}}; \alpha_0 \approx \alpha_1 \to \alpha_0$$
$$\Rightarrow^{(\mathsf{v}_1)}_{\{\alpha_1 \mapsto \mathsf{Int}\}}$$
$$\alpha_0 \approx \mathsf{Int} \to \alpha_0$$

At this stage no rule is applicable. As it is not possible to derive $\square$, the unification problem is not solvable.