universität innsbruck

Functional Programming | WS 2018/2019 | LVA 703024

EXAM 2 | March 1, 2019

NAME: **MATRICULATION NUMBER:**

1 (a) *equational reasoning*

```
takeChain (<) [1,2,3,1] = 1 : takeChain (<) [2,3,1]          (since 1 < 2)
                        = 1 : 2 : takeChain (<) [3,1]        (since 2 < 3)
                        = 1 : 2 : [3] = [1,2,3]              (since not 3 < 1)

map (\x -> 0) [1,2,3,1] = 0 : map (\x -> 0) [2,3,1]
                        = 0 : 0 : map (\x -> 0) [3,1]
                        = 0 : 0 : 0 : map (\x -> 0) [1]
                        = 0 : 0 : 0 : 0 : map (\x -> 0) []
                        = 0 : 0 : 0 : 0 : [] = [0,0,0,0]
```

(b) *kinds of recursion*

Neither `map` nor `takeChain` are tail recursive. Both are guardedly recursive.

(c) *a tail recursive implementation*

```
map' f = go []
  where
    go acc (x:xs) = go (f x : acc) xs
    go acc [] = rev [] acc
      where
        rev acc [] = acc
        rev acc (x:xs) = rev (x:acc) xs
```

2 (a)

base case

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

$$\texttt{map}\ (\backslash x\ \texttt{->}\ x)\ \texttt{[]} = \texttt{[]} \qquad \text{(definition of } \texttt{map}\text{)}$$

(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

$$\texttt{map}\ (\backslash x\ \texttt{->}\ x)\ zs = zs$$

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

$$\texttt{map}\ (\backslash x\ \texttt{->}\ x)\ (z : zs) = z : zs$$

(c)

step case

We prove the step case by the following derivation

$$\begin{aligned}
\texttt{map}\ (\backslash x\ \texttt{->}\ x)\ (z : zs) &= (\backslash x\ \texttt{->}\ x)\ z : \texttt{map}\ (\backslash x\ \texttt{->}\ x)\ zs && \text{(definition of } \texttt{map}\text{)} \\
&= z : \texttt{map}\ (\backslash x\ \texttt{->}\ x)\ zs && \text{(beta reduction)} \\
&= z : zs && \text{(IH)}
\end{aligned}$$

3 (a) *conventions*

Using the notational conventions from the lecture we obtain

$$(\lambda x.\, x)\ (\lambda xy.\, y)\ ((\lambda x.\, x)\ (\lambda xy.\, y)\ a)$$

(b) *applicative order reduction*

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

(c) *alpha-equivalence*

Since we only have the variables x and y at our disposal, the only way to solve this exercise is by *shadowing* the bound variable of the outer abstraction in the nested one (see last two lines):

$$\begin{aligned}&(\lambda yx.\, x)\ y\\ &(\lambda xx.\, x)\ y\\ &(\lambda yy.\, y)\ y\end{aligned}$$

(d)

Consider the λ-term

$$\underline{(\lambda xy.\, y)\ (\underline{(\lambda x.\, x\ x)\ (\lambda x.\, x\ x)})}$$

that contains two redexes (marked by underlining). Contracting the outermost redex (the whole term) yields the normal form $\lambda y.\, y$ in a single β-step, whereas the rightmost redex can be contracted ad infinitum without changing the term, constituting the desired infinite β-reduction.

4 (a) *type checking*

1	$(+) :: \mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$	ins E
2	$f :: \mathsf{Int} \to \mathsf{Int}$	ins E
3	$x :: \mathsf{Int}$	ins E
4	$1 :: \mathsf{Int}$	ins E
5	$(+)\ x :: \mathsf{Int} \to \mathsf{Int}$	app $1, 3$
6	$x + 1 :: \mathsf{Int}$	app $5, 4$
7	$f\ (x + 1) :: \mathsf{Int}$	app $2, 6$

(b) *type inference – unification*

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

At this point the occurs check fails and consequently no rule is applicable. As it is not possible to derive $\square$, the unification problem is not solvable.

(c) *type inference – typing constraints*

$$\underline{E \rhd f\ f :: \alpha_0}$$
$$\Rightarrow^{(\mathsf{app})}$$
$$\underline{E \rhd f :: \alpha_1 \to \alpha_0}; E \rhd f :: \alpha_1$$
$$\Rightarrow^{(\mathsf{con})}$$
$$\mathsf{Int} \to \mathsf{Int} \approx \alpha_1 \to \alpha_0; \underline{E \rhd f :: \alpha_1}$$
$$\Rightarrow^{(\mathsf{con})}$$
$$\mathsf{Int} \to \mathsf{Int} \approx \alpha_1 \to \alpha_0; \mathsf{Int} \to \mathsf{Int} \approx \alpha_1$$