Solutions

1. Consider the λ-terms $I = \lambda x.x$ and $K' = \lambda xy.y$.

(12) (a) Reduce the term $(\lambda p.\, p\ K')\ ((\lambda xyf.\, f\ x\ y)\ I\ I)$ to normal form, using the leftmost innermost reduction strategy.

Solution.

$$\begin{aligned}
&(\lambda p.\, p\ K')\ ((\lambda xyf.\, f\ x\ y)\ I\ I)\\
&= (\lambda p.\, p\ (\lambda xy.\, y))\ (\underline{(\lambda xyf.\, f\ x\ y)\ (\lambda x.\, x)}\ (\lambda x.\, x))\\
&\to_\beta (\lambda p.\, p\ (\lambda xy.\, y))\ (\underline{(\lambda yf.\, f\ (\lambda x.\, x)\ y)\ (\lambda x.\, x)})\\
&\to_\beta \underline{(\lambda p.\, p\ (\lambda xy.\, y))\ (\lambda f.\, f\ (\lambda x.\, x)\ (\lambda x.\, x))}\\
&\to_\beta \underline{(\lambda f.\, f\ (\lambda x.\, x)\ (\lambda x.\, x))\ (\lambda xy.\, y)}\\
&\to_\beta \underline{(\lambda xy.\, y)\ (\lambda x.\, x)}\ (\lambda x.\, x)\\
&\to_\beta \underline{(\lambda y.\, y)\ (\lambda x.\, x)}\\
&\to_\beta \lambda x.\, x
\end{aligned}$$

(13) (b) Reduce the term $(\lambda p.\, p\ K')\ ((\lambda xyf.\, f\ x\ y)\ I\ I)$ to normal form, using the leftmost outermost reduction strategy.

Solution.

$$\begin{aligned}
&(\lambda p.\, p\ K')\ ((\lambda xyf.\, f\ x\ y)\ I\ I)\\
&= \underline{(\lambda p.\, p\ (\lambda xy.\, y))\ ((\lambda xyf.\, f\ x\ y)\ (\lambda x.\, x)\ (\lambda x.\, x))}\\
&\to_\beta \underline{(\lambda xyf.\, f\ x\ y)\ (\lambda x.\, x)}\ (\lambda x.\, x)\ (\lambda xy.\, y)\\
&\to_\beta \underline{(\lambda yf.\, f\ (\lambda x.\, x)\ y)\ (\lambda x.\, x)}\ (\lambda xy.\, y)\\
&\to_\beta \underline{(\lambda f.\, f\ (\lambda x.\, x)\ (\lambda x.\, x))\ (\lambda xy.\, y)}\\
&\to_\beta \underline{(\lambda xy.\, y)\ (\lambda x.\, x)}\ (\lambda x.\, x)\\
&\to_\beta \underline{(\lambda y.\, y)\ (\lambda x.\, x)}\\
&\to_\beta \lambda x.\, x
\end{aligned}$$

2. Consider the three Haskell functions

```
drop n xs | n <= 0  =  xs
drop _ []           =  []
drop n (_:xs)       =  drop (n-1) xs

length []      =  0
length (_:xs)  =  1 + length xs

last [x]      =  x
last (_:xs)   =  last xs
```

Prove by induction that `[last` $(y : xs)$`]` $=$ `drop (length` xs`)` $(y : xs)$ for every list xs and arbitrary list element y.

(5) (a) Base case.

Solution.

Base Case (xs = `[]`). By applying the definitions of the three functions, we prove the base case as follows: `drop` (`length []`) (y : `[]`) = `drop` 0 (y : `[]`) = [y] = `[last` (y : `[]`)`]`.

(20) (b) Step case.

Solution.

Step Case ($xs = z : zs$). The IH is that `[last` (u : zs)`]` = `drop` (`length` zs) (u : zs) for arbitrary u.

$$\begin{aligned}\texttt{[last}\ (y : z : zs)\texttt{]} &= \texttt{[last}\ (z : zs)\texttt{]} \\ &\stackrel{\mathsf{IH}}{=} \texttt{drop}\ (\texttt{length}\ zs)\ (z : zs) \\ &= \texttt{drop}\ (\texttt{length}\ (z : zs))\ (y : z : zs)\end{aligned}$$

3. Consider the Haskell function:

```
sum []      =  0
sum (x:xs)  =  x + sum xs
```

(12) (a) Implement a tail-recursive variant of `sum`.

Solution.

```
sum xs = sum' xs 0
  where
    sum' []     acc  =  acc
    sum' (x:xs) acc  =  sum' xs $! (x + acc)
```

Note: the use of `$!` above forces strict evaluation of `x + acc`, thereby avoiding a memory leak.

(13) (b) Use tupling to implement a function `average`, producing the same results as if defined via `average xs = sum xs / length xs` for all non-empty lists xs.

Solution.

```
average xs  =  if l == 0 then 0 else s / l
  where
    (s, l)              = average' xs 0 0
    average' [] s l     = (s, l)
    average' (x:xs) s l = (average' xs $! (s + x)) $! (l + 1)
```

4. Consider the typing environment $E = \{\mathsf{True} :: \mathsf{Bool}\}$.

(12) (a) Use type checking to decide whether the expression **let** $x = \mathsf{True}$ **in** $x\ x$ is of type Bool with respect to the environment E. Justify your answer.

Solution. By the rule (let), we need to be able to construct a proof tree for $E, x : \mathsf{Bool} \vdash x\ x :: \mathsf{Bool}$. Since x is not of function type, this is impossible.

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

$$\alpha_1 \to \alpha_2 \to \alpha_3 \approx \alpha_4 \to (\alpha_2 \to \alpha_2) \to \alpha_5$$

Solution. After two applications of rule (d_2), we obtain:

$$\begin{aligned} \alpha_1 &\approx \alpha_4 \\ \alpha_2 &\approx \alpha_2 \to \alpha_2 \\ \alpha_3 &\approx \alpha_5 \end{aligned}$$

Now, no rule is applicable to the second equation and thus there is no solution.