Solutions

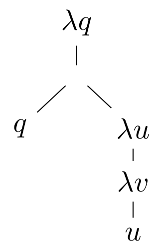
1. Consider the λ-term $t = (\lambda f\ x\ y.f\ ((\lambda x\ y\ p.p\ x\ y)\ x\ y))\ (\lambda q.q\ (\lambda u\ v.u))$.

[5] (a) Compute the sets $\mathcal{V}\mathsf{ar}(t)$, $\mathcal{BV}\mathsf{ar}(t)$, and $\mathcal{FV}\mathsf{ar}(t)$.

Solution. $\mathcal{V}\mathsf{ar}(t) = \mathcal{BV}\mathsf{ar}(t) = \{f, p, q, u, v, x, y\}$ and $\mathcal{FV}\mathsf{ar}(t) = \varnothing$.

[5] (b) Draw the syntax tree of $\lambda q.q\ (\lambda u\ v.u)$.

Solution.



[15] (c) Reduce t to normal form, using the leftmost innermost reduction strategy.

Solution.

$$\begin{aligned}
&(\lambda f\ x\ y.f\ (\underline{(\lambda x\ y\ p.p\ x\ y)\ x}\ y))\ (\lambda q.q\ (\lambda u\ v.u))\\
\to_\beta\ &(\lambda f\ x\ y.f\ (\underline{(\lambda y\ p.p\ x\ y)\ y}))\ (\lambda q.q\ (\lambda u\ v.u))\\
\to_\beta\ &\underline{(\lambda f\ x\ y.f\ (\lambda p.p\ x\ y))\ (\lambda q.q\ (\lambda u\ v.u))}\\
\to_\beta\ &\lambda x\ y.\underline{(\lambda q.q\ (\lambda u\ v.u))\ (\lambda p.p\ x\ y)}\\
\to_\beta\ &\lambda x\ y.\underline{(\lambda p.p\ x\ y)\ (\lambda u\ v.u)}\\
\to_\beta\ &\lambda x\ y.\underline{(\lambda u\ v.u)\ x}\ y\\
\to_\beta\ &\lambda x\ y.\underline{(\lambda v.x)\ y}\\
\to_\beta\ &\lambda x\ y.x
\end{aligned}$$

2. Consider the OCaml functions

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

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

let rec rev = function []    -> []
                     | x::xs -> rev xs @ [x]
```

Use induction over xs, to prove that `rev_append` $xs\ ys$ = `rev` xs `@` ys. Remember that `[x]` is just an abbreviation for x `::` `[]`. Additionally you may freely use the equation:

$$xs\ @\ (ys\ @\ zs) = (xs\ @\ ys)\ @\ zs \tag{1}$$

[5] (a) Give the base case of your induction proof.

Solution.

Base Case (xs = `[]`). By applying the definitions of `rev_append`, `rev`, and `@` we prove the base case as follows: `rev_append [] ` $ys = ys =$ `[] @` $ys =$ `rev [] @` ys.

[20] (b) Give the induction hypothesis and the step case of your induction proof.

Solution.

Step Case ($xs = z :: zs$). The IH is `rev_append` $zs\ ys$ = `rev` zs `@` ys for arbitrary ys.

$$\begin{aligned}
\texttt{rev_append}\ (z :: zs)\ ys &= \texttt{rev_append}\ zs\ (z :: ys) \\
&= \texttt{rev}\ zs\ \texttt{@}\ (z :: ys) && \text{by IH} \\
&= \texttt{rev}\ zs\ \texttt{@}\ (\texttt{[}z\texttt{]}\ \texttt{@}\ ys) \\
&= (\texttt{rev}\ zs\ \texttt{@}\ \texttt{[}z\texttt{]})\ \texttt{@}\ ys && \text{by (1)} \\
&= \texttt{rev}(z :: zs)\ \texttt{@}\ ys
\end{aligned}$$

3. Consider the OCaml functions

```
let rec count0 = function
  | []     -> 0
  | x::xs -> if x = 0 then 1 + count0 xs
                      else count0 xs

let rec count1 = function
  | []     -> 0
  | x::xs -> if x = 1 then 1 + count1 xs
                      else count1 xs
```

[12] (a) Use tupling to implement a function `count` that combines the effects of `count0` and `count1`.

Solution.

```
let rec count = function
  | []     -> (0,0)
  | x::xs ->
    let (z,o) = count xs in
    if x = 0 then (z+1,o) else
    if x = 1 then (z,o+1)
             else (z,o)
```

[13] (b) Give a tail-recursive implementation of the function `count` from (a).

Solution.

```
let count' xs =
  let rec aux (z,o) = function
    | []     -> (z,o)
    | x::xs -> if x = 0 then aux (z+1,o) xs else
```

```
                    if x = 1 then aux (z,o+1) xs
                             else aux (z,o) xs
      in
      aux (0,0) xs
```

4. Consider the CoreML expression $e = (\lambda x.x \,@\, x)$ together with the environment $E = \{@ : \mathsf{list}(\alpha_0) \to \mathsf{list}(\alpha_0) \to \mathsf{list}(\alpha_0)\}$.

[10] (a) Use type checking to prove that e has the type $\tau = \mathsf{list}(\alpha_0) \to \mathsf{list}(\alpha_0)$.

Solution.

$$\dfrac{\dfrac{\dfrac{}{E' \vdash @ : \mathsf{list}(\alpha_0) \to \mathsf{list}(\alpha_0) \to \mathsf{list}(\alpha_0)}\,(\mathsf{ref}) \quad \dfrac{}{E' \vdash x : \mathsf{list}(\alpha_0)}\,(\mathsf{ref})}{E' \vdash (@)\ x : \mathsf{list}(\alpha_0) \to \mathsf{list}(\alpha_0)}\,(\mathsf{app}) \quad \dfrac{}{E' \vdash x : \mathsf{list}(\alpha_0)}\,(\mathsf{ref})}{\dfrac{E' = E, x : \mathsf{list}(\alpha_0) \vdash x \,@\, x : \mathsf{list}(\alpha_0)}{E \vdash e : \tau}\,(\mathsf{abs})}\,(\mathsf{app})$$

[15] (b) First transform $E \triangleright e : \mathsf{list}(\mathsf{int}) \to \mathsf{list}(\mathsf{int})$ into a unification problem and then apply unification to infer whether e really has the type $\mathsf{list}(\mathsf{int}) \to \mathsf{list}(\mathsf{int})$.

Solution.

Solutions

$$
\begin{aligned}
& E \rhd e : \mathsf{list}(\mathsf{int}) \to \mathsf{list}(\mathsf{int}) \\
& \Rightarrow_{(\mathsf{abs})} \\
& E' = E, x : \alpha_1 \rhd x \mathbin{@} x : \alpha_2; \\
& \mathsf{list}(\mathsf{int}) \to \mathsf{list}(\mathsf{int}) \approx \alpha_1 \to \alpha_2 \\
& \Rightarrow_{(\mathsf{app})} \\
& E' \rhd (@)\ x : \alpha_3 \to \alpha_2; E' \rhd x : \alpha_3; \\
& \mathsf{list}(\mathsf{int}) \to \mathsf{list}(\mathsf{int}) \approx \alpha_1 \to \alpha_2 \\
& \Rightarrow_{(\mathsf{app})} \\
& E' \rhd @ : \alpha_4 \to \alpha_3 \to \alpha_2; E' \rhd x : \alpha_4; E' \rhd x : \alpha_3; \\
& \mathsf{list}(\mathsf{int}) \to \mathsf{list}(\mathsf{int}) \approx \alpha_1 \to \alpha_2 \\
& \Rightarrow^{+}_{(\mathsf{con})} \\
& \mathsf{list}(\alpha_0) \to \mathsf{list}(\alpha_0) \to \mathsf{list}(\alpha_0) \approx \alpha_4 \to \alpha_3 \to \alpha_2; \\
& \alpha_1 \approx \alpha_4; \\
& \alpha_1 \approx \alpha_3; \\
& \mathsf{list}(\mathsf{int}) \to \mathsf{list}(\mathsf{int}) \approx \alpha_1 \to \alpha_2
\end{aligned}
$$

$$
\begin{aligned}
& \mathsf{list}(\alpha_0) \to \mathsf{list}(\alpha_0) \to \mathsf{list}(\alpha_0) \approx \alpha_4 \to \alpha_3 \to \alpha_2; \\
& \alpha_1 \approx \alpha_4; \\
& \alpha_1 \approx \alpha_3; \\
& \mathsf{list}(\mathsf{int}) \to \mathsf{list}(\mathsf{int}) \approx \alpha_1 \to \alpha_2 \\
& \Rightarrow^{(\mathsf{d_2})+}_{\iota} \\
& \mathsf{list}(\alpha_0) \approx \alpha_4; \\
& \mathsf{list}(\alpha_0) \approx \alpha_3; \\
& \mathsf{list}(\alpha_0) \approx \alpha_2; \\
& \alpha_1 \approx \alpha_4; \\
& \alpha_1 \approx \alpha_3; \\
& \mathsf{list}(\mathsf{int}) \approx \alpha_1; \\
& \mathsf{list}(\mathsf{int}) \approx \alpha_2 \\
& \Rightarrow^{(\mathsf{v_2})+}_{\{\alpha_1/\mathsf{list}(\mathsf{int}), \alpha_2/\mathsf{list}(\alpha_0), \alpha_3/\mathsf{list}(\alpha_0), \alpha_4/\mathsf{list}(\alpha_0)\}} \\
& \mathsf{list}(\mathsf{int}) \approx \mathsf{list}(\alpha_0); \\
& \mathsf{list}(\mathsf{int}) \approx \mathsf{list}(\alpha_0); \\
& \mathsf{list}(\mathsf{int}) \approx \mathsf{list}(\alpha_0) \\
& \Rightarrow^{(\mathsf{d_1})+}_{\iota} \\
& \mathsf{int} \approx \alpha_0; \mathsf{int} \approx \alpha_0; \mathsf{int} \approx \alpha_0 \\
& \Rightarrow^{(\mathsf{v_2})}_{\{\alpha_0/\mathsf{int}\}} \\
& \mathsf{int} \approx \mathsf{int}; \mathsf{int} \approx \mathsf{int} \\
& \Rightarrow^{(\mathsf{t})+}_{\iota} \\
& \square
\end{aligned}
$$