University of Innsbruck | **Institute of Computer Science**
1st Exam | **February 2, 2011**

Functional Programming | **WS 2010/2011** | **LVA 703017**

Solutions

1. Consider the λ-terms $I \stackrel{\text{def}}{=} (\lambda x.\, x)$, $K \stackrel{\text{def}}{=} (\lambda xy.\, x)$, and $S \stackrel{\text{def}}{=} (\lambda xyz.\, x\ z\ (y\ z))$.

(12) (a) Reduce the term $K\ I\ S\ S$ to normal form, using the leftmost innermost reduction strategy.

Solution.

$$\begin{aligned} K\ I\ S\ S &= \underline{(\lambda xy.\, x)\ (\lambda x.\, x)}\ (\lambda xyz.\, x\ z\ (y\ z))\ (\lambda xyz.\, x\ z\ (y\ z)) \\ &\to_\beta \underline{(\lambda yx.\, x)\ (\lambda xyz.\, x\ z\ (y\ z))}\ (\lambda xyz.\, x\ z\ (y\ z)) \\ &\to_\beta \underline{(\lambda x.\, x)\ (\lambda xyz.\, x\ z\ (y\ z))} \\ &\to_\beta \lambda xyz.\, x\ z\ (y\ z) \end{aligned}$$

(13) (b) Reduce the term $K\ I\ S\ S$ to normal form, using the leftmost outermost reduction strategy.

Solution.

$$\begin{aligned} K\ I\ S\ S &= \underline{(\lambda xy.\, x)\ (\lambda x.\, x)}\ (\lambda xyz.\, x\ z\ (y\ z))\ (\lambda xyz.\, x\ z\ (y\ z)) \\ &\to_\beta \underline{(\lambda yx.\, x)\ (\lambda xyz.\, x\ z\ (y\ z))}\ (\lambda xyz.\, x\ z\ (y\ z)) \\ &\to_\beta \underline{(\lambda x.\, x)\ (\lambda xyz.\, x\ z\ (y\ z))} \\ &\to_\beta \lambda xyz.\, x\ z\ (y\ z) \end{aligned}$$

2. Consider the Haskell functions:

```
[] ++ ys       =  ys
(x:xs) ++ ys   =  x : (xs ++ ys)

take n _ | n <= 0  =  []
take _ []          =  []
take n (x:xs)      =  x : take (n-1) xs

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

Prove by induction over xs that `take` i xs `++ drop` i $xs = xs$ for every list xs and an arbitrary integer i. (*Hint:* You will need a case distinction on i—either $i \leq 0$ or $i > 0$—in the base case as well as the step case.)

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

Solution.

- **Base Case** ($xs =$ `[]`). If $i \leq 0$ then we can use the respective first equations of `take` and `drop` to obtain `take` i `[] ++ drop` i `[] = [] ++ [] = []`. Otherwise $i > 0$ and hence we may use the respective second equations to obtain the same result.

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

Solution.

- **Step Case** ($xs = z$ `:` zs). The IH is `take` n zs `++ drop` n $zs = zs$ for every n. If $i \leq 0$ we conclude the proof as in the base case. Otherwise, $i = j + 1$ for some $j \geq 0$. We conclude

by the derivation:

$$\begin{aligned}\texttt{take}\ i\ xs\ \texttt{++}\ \texttt{drop}\ i\ xs &= \texttt{take}\ (j+1)\ (z\ \texttt{:}\ zs)\ \texttt{++}\ \texttt{drop}\ (j+1)\ (z\ \texttt{:}\ zs)\\ &= (z\ \texttt{:}\ \texttt{take}\ j\ zs)\ \texttt{++}\ \texttt{drop}\ j\ zs\\ &= z\ \texttt{:}\ (\texttt{take}\ j\ zs\ \texttt{++}\ \texttt{drop}\ j\ zs)\\ &\overset{\mathsf{IH}}{=} z\ \texttt{:}\ zs\\ &= xs\end{aligned}$$

3. Consider the Haskell functions:

```
prod []     =  1
prod (x:xs) =  x * prod xs

gmean xs  =  prod xs ** (1 / fromIntegral (length xs))
```

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

Solution.

```
prod = p 1
  where p acc []     = acc
        p acc (x:xs) = p (x * acc) xs
```

(13) (b) Use tupling to implement a more efficient variant of `gmean` (which need not be tail-recursive, however).

Solution.

```
gmean xs = p ** (1 / fromIntegral n)
  where (p, n) = gm xs
        gm []     = (1, 0)
        gm (x:xs) = (x*p, n+1)
          where (p, n) = gm xs
```

4. Consider the typing environment

$$E = \{+ :: \mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}, \mathsf{head} :: \mathsf{List}(\alpha_0) \to \alpha_0, \mathsf{Nil} :: \mathsf{List}(\alpha_0), 4 :: \mathsf{Int}\}.$$

(13) (a) Use type checking to prove the typing judgment:

$$E \vdash \textbf{let}\ h = \mathsf{head}\ \mathsf{Nil}\ \textbf{in}\ h + 4 :: \mathsf{Int}$$

Solution.

1	Nil :: List(Int)	ins $E\{\alpha_0/\mathsf{Int}\}$
2	head :: List(Int) → Int	ins $E\{\alpha_0/\mathsf{Int}\}$
3	head Nil :: Int	app 2, 1
4	h :: Int	assumption
5	$+$ :: Int → Int → Int	ins E
6	4 :: Int	ins E
7	$(+)\ h$ :: Int → Int	app 5, 4
8	$h + 4$ :: Int	app 7, 6
9	**let** h = head Nil **in** $h + 4$ :: Int	let 3, 4–8

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

$$\alpha_1 \to \alpha_1 \approx \alpha_2 \to \alpha_0$$
$$\alpha_0 \approx \alpha_1 \to \alpha_1$$

Justify your answer.

Solution.

$$\alpha_1 \to \alpha_1 \approx \alpha_2 \to \alpha_0; \alpha_0 \approx \alpha_1 \to \alpha_1$$
$$\Rightarrow_{\iota}^{(\mathsf{d}_2)}$$
$$\alpha_1 \approx \alpha_2; \alpha_1 \approx \alpha_0; \alpha_0 \approx \alpha_1 \to \alpha_1$$
$$\Rightarrow^{(\mathsf{v}_1)}_{\{\{\alpha_1/\alpha_2\}\}}$$
$$\alpha_2 \approx \alpha_0; \alpha_0 \approx \alpha_2 \to \alpha_2$$
$$\Rightarrow^{(\mathsf{v}_1)}_{\{\{\alpha_2/\alpha_0\}\}}$$
$$\alpha_0 \approx \alpha_0 \to \alpha_0$$

At this point we are stuck, since the occur-check fails. Hence, the initial problem has no solution.