Problems From Recitation

Paul Zagieboylo (pzagiebo+212@)

September 7, 2006

Useful lemmas (used without proof):

$$l_1 \texttt{@nil} \cong l_1 \tag{1}$$

$$l_1 \texttt{@}(l_2 \texttt{@} l_3) \cong (l_1 \texttt{@} l_2) \texttt{@} l_3 \tag{2}$$

1 Reverse of Append

Theorem: $\texttt{rev}(l_1 \texttt{@} l_2) \cong \texttt{rev}\, l_2 \texttt{@}\, \texttt{rev}\, l_1$

Proof: by structural induction on l_1.

Base case: $l_1 = \texttt{nil}$. Then $\texttt{rev}(\texttt{nil@}l_2) \Rightarrow \texttt{rev}\, l_2$ and $\texttt{rev}\, l_2 \texttt{@}\, \texttt{rev nil} \Rightarrow \texttt{rev}\, l_2 \texttt{@nil} \cong \texttt{rev}\, l_2$ by lemma 1.

Inductive step: $l_1 = x::l_1'$. We need to show that $\texttt{rev}((x::l_1') \texttt{@} l_2) \cong \texttt{rev}\, l_2 \texttt{@}\, \texttt{rev}(x::l_1')$.

Inductive hypothesis: $\texttt{rev}(l_1' \texttt{@} l_2) \simeq \texttt{rev}\, l_2 \texttt{@}\, \texttt{rev}\, l_1'$.

$$\begin{aligned}
\texttt{rev}((x::l_1') \texttt{@} l_2) &\Rightarrow \texttt{rev}(x::(l_1' \texttt{@} l_2)) \\
&\Rightarrow \texttt{rev}(l_1' \texttt{@} l_2) \texttt{@} [x] \\
\text{(by IH)} &\cong (\texttt{rev}\, l_2 \texttt{@}\, \texttt{rev}\, l_1') \texttt{@} [x] \\
\text{(by lemma 2)} &\cong \texttt{rev}\, l_2 \texttt{@} (\texttt{rev}\, l_1' \texttt{@} [x]) \\
\texttt{rev}\, l_2 \texttt{@} (\texttt{rev}(x::l_1')) &\Rightarrow \texttt{rev}\, l_2 \texttt{@} (\texttt{rev}\, l_1' \texttt{@} [x])
\end{aligned}$$

2 Cross product

Here is the specification of the function:

```
(* val Xprod: ('a list * 'b list) -> ('a * 'b) list
 * Xprod ([x1, x2, ..., xm], [y1, y2, ..., ym]) ==>
 *    [(x1, y1), (x2, y1), ..., (xm, y1),
 *     (x1, y2), (x2, y2), ..., (xm, y2),
 *     ...,
 *     (x1, yn), (x2, yn), ..., (xm, yn)]
 * no invariants or effects. *)
```

The simplest way to write this function is given below; convince yourself how and why it works.

```
fun Xprod (l1:'a list, l2:'b list) :('a * 'b) list =
    foldr (fn (y:'b, acc:('a * 'b) list) =>
              foldr (fn (x:'a, acc':('a * 'b) list) => (x, y)::acc')
                    acc l1)
          nil l2
```

Here is a rather cute tail-recursive way to do this task. See if you can figure out how this one works:

```
fun Xprod (l1:'a list, l2:'b list) :('a * 'b) list =
  let
    (* val Xprod'' : 'a list * 'b * ('a * 'b) list -> ('a * 'b) list
     * Xprod'' ([x1, x2, ..., xm], y, acc) ==> (xm, y)::...::(x2, y)::(x1, y)::acc
     * no invariants or effects *)
    fun Xprod'' (nil:'a list, y:'b, acc:('a * 'b) list) : ('a * 'b) list = acc
      | Xprod'' (x::l1', y, acc) = Xprod''(l1', y, (x,y)::acc)
    (* val Xprod' : 'a list * 'b list * ('a * 'b) list -> ('a * 'b) list
     * Xprod' ([x1, x2, ..., xm], [y1, y2, ..., yn], acc) ==>
     *    (xm, yn)::...::(x2, yn)::(x1, yn)::
     *    ...::
     *    (xm, y2)::...::(x2, y2)::(x1, y2)::
     *    (xm, y1)::...::(x2, y1)::(x1, y1)::acc
     *    no invariants or effects *)
    fun Xprod' (l1:'a list, nil:'b list, acc:('a * 'b) list) :('a * 'b list) = acc
      | Xprod' (l1, (y::l2'), acc) = Xprod'(l1, l2', Xprod''(l1, y, acc))
  in
    rev (Xprod' (l1, l2, nil))
  end
```

3 Double Reversal

Theorem: $\texttt{rev}(\texttt{rev}\, l) \cong l$

Proof: by structural induction on l.

Base case: $l = \texttt{nil}$. Then $\texttt{rev}(\texttt{rev}\,\texttt{nil}) \Rightarrow \texttt{rev}\,\texttt{nil} \Rightarrow \texttt{nil}$.

Inductive step: $l = x::l'$. We need to show that $\texttt{rev}(\texttt{rev}(x::l')) \cong x::l'$

Inductive hypothesis: $\texttt{rev}(\texttt{rev}\, l') \cong l'$.

$$
\begin{aligned}
\texttt{rev}(\texttt{rev}(x::l')) &\Rightarrow \texttt{rev}(\texttt{rev}\, l'\texttt{@}[x]) \\
\text{(by theorem 1)} &\cong \texttt{rev}[x]\texttt{@}\,\texttt{rev}(\texttt{rev}\, l') \\
&\Rightarrow [x]\texttt{@}\,\texttt{rev}(\texttt{rev}\, l') \\
\text{(by IH)} &\cong [x]\texttt{@}l' \\
&\Rightarrow x::l'
\end{aligned}
$$