Functional Programming
Practicals 4. Monads

Shin-Cheng Mu

July 2018

1. Show that $\mathit{scutprod}\ \mathit{xs} = \mathit{return}\ (\mathit{product}\ \mathit{xs})$ for all xs. You can use these rules regarding **if**:

$$\begin{aligned}
&\mathbf{if}\ p\ \mathbf{then}\ e\ \mathbf{else}\ e = e \quad ,\\
&(p \Rightarrow \mathit{e1} = \mathit{e2}) \Rightarrow \mathbf{if}\ p\ \mathbf{then}\ \mathit{e1}\ \mathbf{else}\ \mathit{e3} = \mathbf{if}\ p\ \mathbf{then}\ \mathit{e2}\ \mathbf{else}\ \mathit{e3} \quad .
\end{aligned}$$

Solution: We reason:

$$\begin{aligned}
&\quad \mathit{scutprod}\ \mathit{xs}\\
&= \quad \{\ \text{definition}\ \}\\
&\quad \mathit{catch}\ (\mathbf{if}\ \mathit{elem}\ 0\ \mathit{xs}\ \mathbf{then}\ \mathit{fail}\ \mathbf{else}\ \mathit{return}\ (\mathit{product}\ \mathit{xs}))\\
&\qquad\quad (\mathit{return}\ 0)\\
&= \quad \{\ f\ (\mathbf{if}\ p\ \mathbf{then}\ \mathit{e1}\ \mathbf{else}\ \mathit{e2}) = \mathbf{if}\ p\ \mathbf{then}\ f\ \mathit{e1}\ \mathbf{else}\ f\ \mathit{e2}\ \}\\
&\quad \mathbf{if}\ \mathit{elem}\ 0\ \mathit{xs}\ \mathbf{then}\ \mathit{catch}\ \mathit{fail}\ (\mathit{return}\ 0)\\
&\qquad\qquad\qquad \mathbf{else}\ \mathit{catch}\ (\mathit{return}\ (\mathit{product}\ \mathit{xs}))\ (\mathit{return}\ 0)\\
&= \quad \{\ \text{laws concerning}\ \mathit{catch}\ \}\\
&\quad \mathbf{if}\ \mathit{elem}\ 0\ \mathit{xs}\ \mathbf{then}\ \mathit{return}\ 0\\
&\qquad\qquad\qquad \mathbf{else}\ \mathit{return}\ (\mathit{product}\ \mathit{xs})\\
&= \quad \{\ \text{since}\ \mathit{product}\ \mathit{xs} = 0\ \text{if}\ \mathit{elem}\ 0\ \mathit{xs}\ \}\\
&\quad \mathbf{if}\ \mathit{elem}\ 0\ \mathit{xs}\ \mathbf{then}\ \mathit{return}\ (\mathit{product}\ \mathit{xs})\\
&\qquad\qquad\qquad \mathbf{else}\ \mathit{return}\ (\mathit{product}\ \mathit{xs})\\
&= \quad \{\ \text{laws regarding}\ \mathbf{if}\ \}\\
&\quad \mathit{return}\ (\mathit{product}\ \mathit{xs}) \quad .
\end{aligned}$$

2. Prove: $\mathit{insert}\ x\ \mathit{ys} \gg\!= (\mathit{return} \cdot \mathit{map}\ f) = \mathit{insert}\ (f\ x)\ (\mathit{map}\ f\ \mathit{ys})$.

Solution: Induction on ys. **Case**: $ys := []$:

$$\begin{array}{ll}
& \mathit{insert}\ x\ []\ \gg\!\!= (\mathit{return} \cdot \mathit{map}\ f) \\
= & \{\ \text{definition of } \mathit{insert}\ \} \\
& \mathit{return}\ [x] \gg\!\!= (\mathit{return} \cdot \mathit{map}\ f) \\
= & \{\ \text{monad law: left-identity}\ \} \\
& \mathit{return}\ (\mathit{map}\ f\ [x]) \\
= & \{\ \text{definition of } \mathit{xs}\ \} \\
& \mathit{return}\ [f\ x] \\
= & \{\ \text{definition of } \mathit{insert}\ \} \\
& \mathit{insert}\ (f\ x)\ (\mathit{map}\ f\ [])\quad .
\end{array}$$

Case: $ys := y : ys$:

$$\begin{array}{ll}
& \mathit{insert}\ x\ (y : ys) \gg\!\!= (\mathit{return} \cdot \mathit{map}\ f) \\
= & \{\ \text{definition of } \mathit{insert}\ \} \\
& (\mathit{return}\ (x : y : ys) \mathbin{[\!]} \mathit{insert}\ x\ ys \gg\!\!= (\mathit{return} \cdot (y{:}))) \gg\!\!= (\mathit{return} \cdot \mathit{map}\ f) \\
= & \{\ \text{left-distributivity}\ \} \\
& (\mathit{return}\ (x : y : ys) \gg\!\!= (\mathit{return} \cdot \mathit{map}\ f)) \mathbin{[\!]} \\
& ((\mathit{insert}\ x\ ys \gg\!\!= (\mathit{return} \cdot (y{:})) \gg\!\!= (\mathit{return} \cdot \mathit{map}\ f)) \\
= & \{\ \text{monad law: left-identity}\ \} \\
& \mathit{return}\ (\mathit{map}\ f\ (x : y : ys)) \mathbin{[\!]} \\
& ((\mathit{insert}\ x\ ys \gg\!\!= (\mathit{return} \cdot (y{:})) \gg\!\!= (\mathit{return} \cdot \mathit{map}\ f))
\end{array}$$

In the left branch of $(\mathbin{[\!]})$, $\mathit{map}\ f\ (x : y : ys)$ naturally expands to $f\ x : f\ y : \mathit{map}\ f\ ys$. Focus on the right branch of $(\mathbin{[\!]})$:

$$\begin{array}{ll}
& (\mathit{insert}\ x\ ys \gg\!\!= (\mathit{return} \cdot (y{:})) \gg\!\!= (\mathit{return} \cdot \mathit{map}\ f) \\
= & \{\ \text{monad law: associativity}\ \} \\
& \mathit{insert}\ x\ ys \gg\!\!= (\lambda zs \to \mathit{return}\ (y : zs) \gg\!\!= (\mathit{return} \cdot \mathit{map}\ f)) \\
= & \{\ \text{monad law: left-identity}\ \} \\
& \mathit{insert}\ x\ ys \gg\!\!= (\lambda zs \to \mathit{return}\ (\mathit{map}\ f\ (y : zs))) \\
= & \{\ \text{definition of } \mathit{map}\ \} \\
& \mathit{insert}\ x\ ys \gg\!\!= (\lambda zs \to \mathit{return}\ (f\ y : \mathit{map}\ f\ zs)) \\
= & \{\ \text{monad law: left-identity}\ \} \\
& \mathit{insert}\ x\ ys \gg\!\!= (\lambda zs \to \mathit{return}\ (\mathit{map}\ f) \gg\!\!= (\mathit{return} \cdot (f\ y{:}))) \\
= & \{\ \text{monad law: associativity}\ \} \\
& (\mathit{insert}\ x\ ys \gg\!\!= (\mathit{return} \cdot \mathit{map}\ f)) \gg\!\!= (\mathit{return} \cdot (f\ y{:})) \\
= & \{\ \text{induction}\ \} \\
& \mathit{insert}\ (f\ x)\ (\mathit{map}\ f\ ys) \gg\!\!= (\mathit{return} \cdot (f\ y{:}))\quad .
\end{array}$$

Back to the calculation:

$$\begin{array}{ll}
& \mathit{return}\ (\mathit{map}\ f\ (x : y : ys)) \mathbin{[\!]} \\
& ((\mathit{insert}\ x\ ys \gg\!\!= (\mathit{return} \cdot (y{:})) \gg\!\!= (\mathit{return} \cdot \mathit{map}\ f))
\end{array}$$

$$
\begin{aligned}
= &\quad \{\text{ calculations above }\}\\
&return\ (f\ x : f\ y : map\ f\ ys)\ [\!]\\
&insert\ (f\ x)\ (map\ f\ ys) \gg\!= (return \cdot (f\ y{:}))\\
= &\quad \{\text{ definition of } insert\ \}\\
&insert\ (f\ x)\ (f\ y : map\ f\ ys)\\
= &\quad \{\text{ definition of } map\ \}\\
&insert\ (f\ x)\ (map\ f\ (y : ys)) \quad .
\end{aligned}
$$

3. **Sorting**. Shown below is a very slow sorting algorithm, which non-deterministically generates an arbitrary permutation of its input, and succeeds only if the permutation happens to be sorted:

$$slowsort\ xs = perm\ xs \gg\!= sorted \quad ,$$

with auxiliary functions defined below:

$$
\begin{aligned}
&perm\ [\,] = return\ [\,]\\
&perm\ (x : xs) = perm\ xs \gg\!= insert\ x \quad ,\\
&guard\ b = \textbf{if}\ b\ \textbf{then}\ return\ ()\ \textbf{else}\ fail \quad ,\\
&all\ p\ [\,] = \mathsf{True}\\
&all\ p\ (x : xs) = p\ x \wedge all\ p\ xs \quad ,\\
&sorted\ [\,] = return\ [\,]\\
&sorted\ (x : xs) = guard\ (all\ (x \leqslant)\ xs) \gg\!= \lambda () \rightarrow\\
&\qquad sorted\ xs \gg\!= (return \cdot (x{:})) \quad .
\end{aligned}
$$

For this exercise we assume that the input list has type $\mathsf{List\ Int}$.

(a) Write down the types of each of the functions above, and explain what they do.

(b) Consider the following function:

$$
\begin{aligned}
&sinsert\ x\ [\,] = return\ [x]\\
&sinsert\ x\ (y : xs) = \textbf{if}\ x \leqslant y\ \textbf{then}\ return\ (x : y : xs)\\
&\qquad \textbf{else}\ sinsert\ x\ xs \gg\!= (return \cdot (y{:})) \quad .
\end{aligned}
$$

Do you believe that the following property (1) is true? Can you explain what it means in words?

$$insert\ x\ xs \gg\!= sorted \;=\; sorted\ xs \gg\!= sinsert\ x \quad . \tag{1}$$

(c) Assuming that (1) is true. Derive a faster (well, $O(n^2)$) sorting algorithm.

Solution: Consider *slowsort xs* and do induction on the input *xs*. For the base case:

$$
\begin{aligned}
& \mathit{slowsort}\ [\,] \\
= \quad & \{\ \text{definition of } \mathit{slowsort}\ \} \\
& \mathit{perm}\ [\,] \gg\!= \mathit{sorted} \\
= \quad & \{\ \text{definition of } \mathit{perm}\ \} \\
& \mathit{return}\ [\,] \gg\!= \mathit{sorted} \\
= \quad & \{\ \text{monad law}\ \} \\
& \mathit{sorted}\ [\,] \\
= \quad & \{\ \text{definition of } \mathit{sorted}\ \} \\
& \mathit{return}\ [\,] \quad .
\end{aligned}
$$

For the inductive case:

$$
\begin{aligned}
& \mathit{slowsort}\ (x : xs) \\
= \quad & \{\ \text{definition of } \mathit{slowsort}\ \} \\
& \mathit{perm}\ (x : xs) \gg\!= \mathit{sorted} \\
= \quad & \{\ \text{definition of } \mathit{perm}\ \} \\
& (\mathit{perm}\ xs \gg\!= \mathit{insert}\ x) \gg\!= \mathit{sorted} \\
= \quad & \{\ \text{monad law: associativity of } (\gg\!=)\ \} \\
& \mathit{perm}\ xs \gg\!= (\lambda ys \to \mathit{insert}\ x\ ys \gg\!= \mathit{sorted}) \\
= \quad & \{\ \text{by (1)}\ \} \\
& \mathit{perm}\ xs \gg\!= (\lambda ys \to \mathit{sorted}\ ys \gg\!= \mathit{sinsert}\ x) \\
= \quad & \{\ \text{monad law: associativity of } (\gg\!=)\ \} \\
& (\mathit{perm}\ xs \gg\!= \mathit{sorted}) \gg\!= \mathit{sinsert}\ x \\
= \quad & \{\ \text{definition of } \mathit{slowsort}\ \} \\
& \mathit{slowsort}\ xs \gg\!= \mathit{sinsert}\ x \quad .
\end{aligned}
$$

We have thus constructed:

$$
\begin{aligned}
\mathit{slowsort}\ [\,] &= \mathit{return}\ [\,] \\
\mathit{slowsort}\ (x : xs) &= \mathit{slowsort}\ xs \gg\!= \mathit{sinsert}\ x \quad ,
\end{aligned}
$$

which is actually, as you might have guessed, insertion sort.

4. Assuming the following implementation of MonadNondet:

$$
\begin{aligned}
& \textbf{instance}\ \mathsf{MonadNondet}\ \mathsf{Maybe}\ \textbf{where} \\
& \quad \mathsf{Nothing} \mathbin{[\!]} m = m \\
& \quad \mathsf{Just}\ x \mathbin{[\!]} m = \mathsf{Just}\ x
\end{aligned}
$$

Think of a counterexample for which the left distributivity law does not hold.

Solution: Recall the law:

$$(m1 \mathbin{[\!]} m2) \mathbin{>\!\!>\!\!=} f \;=\; (m1 \mathbin{>\!\!>\!\!=} f) \mathbin{[\!]} (m2 \mathbin{>\!\!>\!\!=} f) \;\; .$$

Let $m1 = \mathsf{Just}\ 1$, $m2 = \mathsf{Just}\ 2$, and $f\ x = \mathbf{if}\ even\ x\ \mathbf{then}\ \mathsf{Just}\ ()\ \mathbf{else}\ \mathsf{Nothing}$. The LHS reduces to

$$\begin{aligned} & (\mathsf{Just}\ 1 \mathbin{[\!]} \mathsf{Just}\ 2) \mathbin{>\!\!>\!\!=} f \\ = {} & \mathsf{Just}\ 1 \mathbin{>\!\!>\!\!=} f \\ = {} & \mathsf{Nothing} \;\; , \end{aligned}$$

while the RHS reduces to

$$\begin{aligned} & (\mathsf{Just}\ 1 \mathbin{>\!\!>\!\!=} f) \mathbin{[\!]} (\mathsf{Just}\ 2 \mathbin{>\!\!>\!\!=} f) \\ = {} & \mathsf{Nothing} \mathbin{[\!]} return\ () \\ = {} & return\ () \;\; . \end{aligned}$$

5. Consider the standard prelude function *foldl*:

$$\begin{aligned} & foldl :: (b \to a \to b) \to b \to \mathsf{List}\ a \to b \\ & foldl\ (\oplus)\ s\ [\,] \qquad\;\; = s \\ & foldl\ (\oplus)\ s\ (x : xs) = foldl\ (\oplus)\ (s \oplus x)\ xs \;\; , \end{aligned}$$

and define:

$$\begin{aligned} & loop :: \mathsf{MonadState}\ b\ m \Rightarrow (b \to a \to b) \to \mathsf{List}\ a \to m\ b \\ & loop\ (\oplus)\ [\,] \qquad\;\; = get \\ & loop\ (\oplus)\ (x : xs) = get \mathbin{>\!\!>\!\!=} \lambda s \to \\ & \qquad\qquad\qquad\qquad put\ (s \oplus x) \gg loop\ xs \;\; . \end{aligned}$$

Prove that $put\ s \gg loop\ (\oplus)\ xs = put\ (foldl\ (\oplus)\ s\ xs) \gg get$.

Solution: Induction on xs.
Case $xs := [\,]$:

$$\begin{aligned} & put\ s \gg loop\ (\oplus)\ [\,] \\ = {} & \quad \{ \text{ definition of } loop \ \} \\ & put\ s \gg get \\ = {} & \quad \{ \text{ definition of } foldl \ \} \\ & put\ (foldl\ (\oplus)\ s\ [\,]) \gg get \;\; . \end{aligned}$$

Case $xs := x : xs$:

$$
\begin{aligned}
& put\ s \gg loop\ (\oplus)\ (x : xs) \\
= \quad & \{\ \text{definition of } loop\ \} \\
& put\ s \gg get \gg\!= \lambda s \to put\ (s \oplus x) \gg loop\ xs \\
= \quad & \{\ \text{put-get}\ \} \\
& put\ s \gg return\ s \gg\!= \lambda s \to put\ (s \oplus x) \gg loop\ xs \\
= \quad & \{\ \text{monad law: left identity}\ \} \\
& put\ s \gg put\ (s \oplus x) \gg loop\ xs \\
= \quad & \{\ \text{put-put}\ \} \\
& put\ (s \oplus x) \gg loop\ xs \\
= \quad & \{\ \text{induction}\ \} \\
& put\ (foldl\ (\oplus)\ (s \oplus x)\ xs) \gg get \\
= \quad & \{\ \text{definition of } foldl\ \} \\
& put\ (foldl\ (\oplus)\ s\ (x : xs)) \gg get \quad .
\end{aligned}
$$