Programación Funcional

Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay

Mónadas

Mónadas

Modelado de efectos computacionales a través de **mónadas**.

Ejemplos de efectos:

- Entrada-salida
- Estado
- No determinismo
- Fallas
- Excepciones

Evaluador de Expresiones

$$\text{data } Exp = Num\ Int \mid Add\ Exp\ Exp$$

$$\begin{aligned} &eval :: Exp \to Int \\ &eval\ (Num\ n) \quad = n \\ &eval\ (Add\ x\ y) = eval\ x + eval\ y \end{aligned}$$

Evaluador con Fallas

Supongamos que deseamos agregar un operador de división a nuestras expresiones:

$$\text{data } \mathit{Exp} = \mathit{Num}\ \mathit{Int} \mid \mathit{Add}\ \mathit{Exp}\ \mathit{Exp} \mid \mathit{Div}\ \mathit{Exp}\ \mathit{Exp}$$

La operación de división debe controlar ahora el caso de falla por *división por cero*.

$$\text{data } \mathit{Maybe}\ a = \mathit{Just}\ a \mid \mathit{Nothing}$$

$$\mathit{divM} :: \mathit{Int} \to \mathit{Int} \to \mathit{Maybe}\ \mathit{Int}$$
$$a\ `\mathit{divM}`\ b = \text{if } b == 0 \text{ then } \mathit{Nothing}$$
$$\text{else } \mathit{Just}\ (a\ `\mathit{div}`\ b)$$

Evaluador con Fallas (2)

```
eval :: Exp → Maybe Int
eval (Num n)   = Just n
eval (Add x y) = case eval x of
                   Nothing → Nothing
                   Just a  → case eval y of
                               Nothing → Nothing
                               Just b  → Just (a + b)
eval (Div x y) = case eval x of
                   Nothing → Nothing
                   Just a  → case eval y of
                               Nothing → Nothing
                               Just b  → a `divM` b
```

Evaluador con Fallas (3)

Definamos:

$$
\begin{aligned}
&\mathit{return} \quad :: a \to \mathit{Maybe}\ a \\
&\mathit{return}\ a = \mathit{Just}\ a
\end{aligned}
$$

$$
\begin{aligned}
&(>>=) \quad :: \mathit{Maybe}\ a \to (a \to \mathit{Maybe}\ b) \to \mathit{Maybe}\ b \\
&m >>= f = \text{case } m \text{ of} \\
&\qquad\qquad \mathit{Nothing} \to \mathit{Nothing} \\
&\qquad\qquad \mathit{Just}\ a \quad \to f\ a
\end{aligned}
$$

Evaluador con Fallas (4)

Entonces,

$$
\begin{aligned}
&\mathit{eval} && :: \mathit{Exp} \to \mathit{Maybe\ Int} \\
&\mathit{eval}\ (\mathit{Num}\ n) && = \mathit{return}\ n \\
&\mathit{eval}\ (\mathit{Add}\ x\ y) && = \mathit{eval}\ x\ \texttt{>>=}\ \lambda a \to \mathit{eval}\ y\ \texttt{>>=}\ \lambda b \to \mathit{return}\ (a + b) \\
&\mathit{eval}\ (\mathit{Div}\ x\ y) && = \mathit{eval}\ x\ \texttt{>>=}\ \lambda a \to \mathit{eval}\ y\ \texttt{>>=}\ \lambda b \to a\ \texttt{`}\mathit{divM}\texttt{`}\ b
\end{aligned}
$$

La clase Monad

$$
\begin{aligned}
&\text{class } \mathit{Monad}\ m \text{ where} \\
&\quad (>>=) \quad :: m\ a \to (a \to m\ b) \to m\ b \\
&\quad (>>) \quad :: m\ a \to m\ b \to m\ b \\
&\quad \mathit{return} \quad :: a \to m\ a \\
&\quad \mathit{fail} \quad :: \mathit{String} \to m\ a
\end{aligned}
$$

$$
\begin{aligned}
m >> k &= m >>= \lambda\_ \to k \\
\mathit{fail}\ s &= \mathit{error}\ s
\end{aligned}
$$

Mónada Maybe

```
data Maybe a = Just a | Nothing


instance Monad Maybe where
  return    = Just
  m >>= k = case m of
                Just x   -> k x
                Nothing -> Nothing
  fail _    = Nothing
```

Evaluador con Fallas

Volviendo al evaluador:

$$
\begin{aligned}
&\mathit{eval} && :: \mathit{Exp} \to \mathit{Maybe}\ \mathit{Int} \\
&\mathit{eval}\ (\mathit{Num}\ n) && = \mathit{return}\ n \\
&\mathit{eval}\ (\mathit{Add}\ x\ y) && = \mathit{eval}\ x >\!\!>\!\!= \lambda a \to \mathit{eval}\ y >\!\!>\!\!= \lambda b \to \mathit{return}\ (a + b) \\
&\mathit{eval}\ (\mathit{Div}\ x\ y) && = \mathit{eval}\ x >\!\!>\!\!= \lambda a \to \mathit{eval}\ y >\!\!>\!\!= \lambda b \to a\ `\mathit{divM}`\ b
\end{aligned}
$$

Evaluador con Fallas

$$
\begin{aligned}
&\mathit{eval} && :: \mathit{Exp} \to \mathit{Maybe}\ \mathit{Int} \\
&\mathit{eval}\ (\mathit{Num}\ n) && = \mathit{return}\ n \\
&\mathit{eval}\ (\mathit{Add}\ x\ y) && = \mathit{eval}\ x\ \texttt{>>=}\ \lambda a \to \\
&&& \quad \mathit{eval}\ y\ \texttt{>>=}\ \lambda b \to \\
&&& \quad \mathit{return}\ (a + b) \\
&\mathit{eval}\ (\mathit{Div}\ x\ y) && = \mathit{eval}\ x\ \texttt{>>=}\ \lambda a \to \\
&&& \quad \mathit{eval}\ y\ \texttt{>>=}\ \lambda b \to \\
&&& \quad a\ \texttt{`}\mathit{divM}\texttt{`}\ b
\end{aligned}
$$

Notación do

do m	m
do $x \leftarrow m$ $\quad m'$	$m >>= \lambda x \rightarrow$ do m'
do m $\quad m'$	$m >>$ do m'

Entonces

do $a \leftarrow$ *eval* x
$\quad b \leftarrow$ *eval* y
$\quad$ *return* $(a + b)$

eval $x >>= \lambda a \rightarrow$
do $b \leftarrow$ *eval* y
$\quad$ *return* $(a + b)$

eval $x >>= \lambda a \rightarrow$
eval $y >>= \lambda b \rightarrow$
do *return* $(a + b)$

eval $x >>= \lambda a \rightarrow$
eval $y >>= \lambda b \rightarrow$
return $(a + b)$

Evaluador con Fallas (notación do)

$$
\begin{aligned}
&\mathit{eval} :: \mathit{Exp} \to \mathit{Maybe\ Int} \\
&\mathit{eval}\ (\mathit{Num}\ n) \quad = \mathit{return}\ n \\
&\mathit{eval}\ (\mathit{Add}\ x\ y) = \mathrm{do}\ a \leftarrow \mathit{eval}\ x \\
&\qquad\qquad\qquad\qquad\quad\ \ b \leftarrow \mathit{eval}\ y \\
&\qquad\qquad\qquad\qquad\quad\ \ \mathit{return}\ (a + b) \\
&\mathit{eval}\ (\mathit{Div}\ x\ y) \ = \mathrm{do}\ a \leftarrow \mathit{eval}\ x \\
&\qquad\qquad\qquad\qquad\quad\ \ b \leftarrow \mathit{eval}\ y \\
&\qquad\qquad\qquad\qquad\quad\ \ a\ `\mathit{divM}`\ b
\end{aligned}
$$

Funciones sobre Mónadas (ver Control.Monad)

Este patrón se repite mucho:

$$\begin{aligned} eval\ (Add\ x\ y) = \text{do}\ & a \leftarrow eval\ x \\ & b \leftarrow eval\ y \\ & return\ (a + b) \end{aligned}$$

Lo abstraemos con la función:

$$\begin{aligned} & liftM2 :: Monad\ m \Rightarrow (a \rightarrow b \rightarrow c) \rightarrow m\ a \rightarrow m\ b \rightarrow m\ c \\ & liftM2\ f\ m\ m' = \text{do}\ x \leftarrow m \\ & \qquad\qquad\qquad\qquad y \leftarrow m' \\ & \qquad\qquad\qquad\qquad return\ (f\ x\ y) \end{aligned}$$

Evaluador con Fallas (*liftM2*)

$$
\begin{aligned}
&\mathit{eval} :: \mathit{Exp} \to \mathit{Maybe}\ \mathit{Int} \\
&\mathit{eval}\ (\mathit{Num}\ n) \quad = \mathit{return}\ n \\
&\mathit{eval}\ (\mathit{Add}\ x\ y) = \mathit{liftM2}\ (+)\ (\mathit{eval}\ x)\ (\mathit{eval}\ y) \\
&\mathit{eval}\ (\mathit{Div}\ x\ y) \ = \mathrm{do}\ a \leftarrow \mathit{eval}\ x \\
&\qquad\qquad\qquad\qquad\quad\ \ b \leftarrow \mathit{eval}\ y \\
&\qquad\qquad\qquad\qquad\quad\ \ a\ `\mathit{divM}`\ b
\end{aligned}
$$

Preguntas

Otras Funciones sobre Mónadas (ver Control.Monad)

$$
\begin{aligned}
&\mathit{sequence} :: \mathit{Monad}\ m \Rightarrow [m\ a] \to m\ [a] \\
&\mathit{sequence}\ [\,] \qquad\quad = \mathit{return}\ [\,] \\
&\mathit{sequence}\ (m : ms) = \mathrm{do}\ x \leftarrow m \\
&\qquad\qquad\qquad\qquad\quad\ \ xs \leftarrow \mathit{sequence}\ ms \\
&\qquad\qquad\qquad\qquad\quad\ \ \mathit{return}\ (x : xs)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{filterM} :: \mathit{Monad}\ m \Rightarrow (a \to m\ \mathit{Bool}) \to [a] \to m\ [a] \\
&\mathit{filterM}\ \_\ [\,] \qquad\quad = \mathit{return}\ [\,] \\
&\mathit{filterM}\ p\ (x : xs) = \mathrm{do}\ b \leftarrow p\ x \\
&\qquad\qquad\qquad\qquad\quad\ \ ys \leftarrow \mathit{filterM}\ p\ xs \\
&\qquad\qquad\qquad\qquad\quad\ \ \mathit{return}\ (\mathrm{if}\ b\ \mathrm{then}\ x : ys\ \mathrm{else}\ ys)
\end{aligned}
$$

$$\mathit{mapM} :: \mathit{Monad}\ m \Rightarrow (a \to m\ b) \to [a] \to m\ [b]$$

$$\mathit{foldM} :: \mathit{Monad}\ m \Rightarrow (b \to a \to m\ b) \to b \to [a] \to m\ b$$

Mónada de estado

Computaciones que mantienen un estado

Definir una función *numTree* que tome un árbol binario con valores en las hojas y retorne uno igual pero con sus elementos numerados por una recorrida en orden.

$$\text{data } \mathit{Tree}\ a = \mathit{Leaf}\ a \mid \mathit{Fork}\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)$$

numTree

Un primer intento

$$
\begin{aligned}
&\mathit{numTree} :: \mathit{Tree}\ a \to \mathit{Tree}\ (\mathit{Int}, a) \\
&\mathit{numTree}\ (\mathit{Leaf}\ a) \ \ = \mathit{Leaf}\ (??, a) \\
&\mathit{numTree}\ (\mathit{Fork}\ l\ r) = \mathit{Fork}\ (\mathit{numTree}\ l) \\
&\qquad\qquad\qquad\qquad\qquad\quad (\mathit{numTree}\ r)
\end{aligned}
$$

numTree (2)

Un segundo intento

$$
\begin{aligned}
&\mathit{numTree} :: \mathit{Tree}\ a \to \mathit{Int} \to \mathit{Tree}\ (\mathit{Int}, a) \\
&\mathit{numTree}\ (\mathit{Leaf}\ a) \quad = \lambda \mathit{cont} \to \mathit{Leaf}\ (\mathit{cont}, a) \\
&\mathit{numTree}\ (\mathit{Fork}\ l\ r) = \lambda \mathit{cont} \to \mathit{Fork}\ (\mathit{numTree}\ l\ \mathit{cont}) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad (\mathit{numTree}\ r\ \mathit{cont})
\end{aligned}
$$

numTree (3)

Un tercer intento

$$
\begin{aligned}
&\mathit{numTree} :: \mathit{Tree}\ a \to \mathit{Int} \to \mathit{Tree}\ (\mathit{Int}, a) \\
&\mathit{numTree}\ (\mathit{Leaf}\ a) \quad = \lambda \mathit{cont} \to \mathit{Leaf}\ (\mathit{cont}, a) \\
&\mathit{numTree}\ (\mathit{Fork}\ l\ r) = \lambda \mathit{cont} \to \mathit{Fork}\ (\mathit{numTree}\ l\ \mathit{cont}) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad (\mathit{numTree}\ r\ (\mathit{cont} + 1))
\end{aligned}
$$

numTree (4)

La solución

$$
\begin{aligned}
&\mathit{numTree} :: \mathit{Tree}\ a \to \mathit{Int} \to (\mathit{Tree}\ (\mathit{Int}, a), \mathit{Int}) \\
&\mathit{numTree}\ (\mathit{Leaf}\ a) \quad = \lambda \mathit{cont} \to (\mathit{Leaf}\ (\mathit{cont}, a), \mathit{cont} + 1) \\
&\mathit{numTree}\ (\mathit{Fork}\ l\ r) = \lambda \mathit{cont} \to \mathsf{let}\ (l', \mathit{cont}') \quad = \mathit{numTree}\ l\ \mathit{cont} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad (r', \mathit{cont}'') = \mathit{numTree}\ r\ \mathit{cont}' \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \mathsf{in}\ \ (\mathit{Fork}\ l'\ r', \mathit{cont}'')
\end{aligned}
$$

Usando la **mónada de estado** (`Control.Monad.State`)

$$
\begin{aligned}
&\mathit{numTree} :: \mathit{Tree}\ a \to \mathit{State}\ \mathit{Int}\ (\mathit{Tree}\ (\mathit{Int}, a)) \\
&\mathit{numTree}\ (\mathit{Leaf}\ a) = \mathsf{do} \quad \mathit{cont} \leftarrow \mathit{get} \\
&\qquad\qquad\qquad\qquad\qquad\quad \mathit{put}\ (\mathit{cont} + 1) \\
&\qquad\qquad\qquad\qquad\qquad\quad \mathit{return}\ \$\ \mathit{Leaf}\ (\mathit{cont}, a) \\
&\mathit{numTree}\ (\mathit{Fork}\ l\ r) = \mathsf{do}\ l' \leftarrow \mathit{numTree}\ l \\
&\qquad\qquad\qquad\qquad\qquad\quad r' \leftarrow \mathit{numTree}\ r \\
&\qquad\qquad\qquad\qquad\qquad\quad \mathit{return}\ \$\ \mathit{Fork}\ l'\ r'
\end{aligned}
$$

Implementación de la Mónada de estado

Recordemos:

$$\begin{aligned}&\text{class } \mathit{Monad}\ m \text{ where}\\ &\quad \mathit{return} :: a \to m\ a\\ &\quad (>>=)\ :: m\ a \to (a \to m\ b) \to m\ b\end{aligned}$$

$$\begin{aligned}&\mathit{numTree} :: \mathit{Tree}\ a \to \mathit{Int} \to (\mathit{Tree}\ (\mathit{Int}, a), \mathit{Int})\\ &\mathit{numTree} :: \mathit{Tree}\ a \to \mathit{State}\ \mathit{Int}\ (\mathit{Tree}\ (\mathit{Int}, a))\end{aligned}$$

El tipo de las computaciones con estado:

$$\text{type } \mathit{State}\ s\ a = s \to (a, s)$$

La función *return*:

$$\mathit{return}\ a = \lambda s \to (a, s)$$

¿Cómo implemento (>>=)?

Implementación de la Mónada de estado (2)

La parte donde se va pasando el estado:

$$\begin{aligned} numTree\ (Fork\ l\ r) = \lambda cont \rightarrow\ & \text{let } (l', cont') = numTree\ l\ cont \\ & \quad\ (r', cont'') = numTree\ r\ cont' \\ & \text{in } (Fork\ l'\ r', cont'') \end{aligned}$$

Se puede escribir:

$$\begin{aligned} numTree\ (Fork\ l\ r) = \lambda cont \rightarrow\ & \text{let } (l', cont') = numTree\ l\ cont \\ & \text{in let } (r', cont'') = numTree\ r\ cont' \\ & \quad\ \text{in } (Fork\ l'\ r', cont'') \end{aligned}$$

Que equivale a:

$$\begin{aligned} numTree\ (Fork\ l\ r) = \lambda cont \rightarrow\ & \text{let } (l', cont') = numTree\ l\ cont \\ & \text{in } (\lambda a \rightarrow \lambda s \rightarrow \text{let } (r', cont'') = numTree\ r\ s \\ & \qquad\ \text{in } (Fork\ a\ r', cont''))\ l'\ cont' \end{aligned}$$

Implementación de la Mónada de estado (3)

Si definimos:

$$m >>= f = \lambda s \rightarrow \text{let } (a, s') = m\ s \text{ in } f\ a\ s'$$

Entonces:

$$numTree\ (Fork\ l\ r) = \lambda cont \rightarrow \text{let } (l', cont') = numTree\ l\ cont \text{ in } (\lambda a \rightarrow \lambda s \rightarrow \text{let } (r', cont'') = numTree\ r\ s \text{ in } (Fork\ a\ r', cont''))\ l'\ cont'$$

Se puede escribir:

$$numTree\ (Fork\ l\ r) = numTree\ l >>= \lambda a \rightarrow \lambda s \rightarrow \text{let } (r', cont'') = numTree\ r\ s \text{ in } (Fork\ l'\ r', cont'')$$

O incluso:

$$numTree\ (Fork\ l\ r) = numTree\ l >>= \lambda a \rightarrow numTree\ r\ s >>= \lambda b \rightarrow \lambda s \rightarrow (Fork\ a\ b, s)$$

Implementación de la Mónada de estado (4)

En resumen:

$$\text{type } \mathit{State}\ s\ a = s \to (a, s)$$

$$\begin{aligned}
&\text{instance } \mathit{Monad}\ (\mathit{State}\ s) \text{ where} \\
&\quad \mathit{return}\ a = \lambda s \to (a, s) \\
&\quad m \texttt{>>=}\ f = \lambda s \to \text{let } (a, s') = m\ s \\
&\qquad\qquad\qquad\qquad\ \ \text{in } \ f\ a\ s'
\end{aligned}$$

$$\begin{aligned}
\mathit{get} &= \lambda s \to (s, s) \\
\mathit{put}\ s &= \lambda _ \to ((), s)
\end{aligned}$$

Mónada de estado

$$\text{data } State\ s\ a = State\ (s \to (a, s))$$
$$runState\ (State\ f) = f$$

$$\text{instance } Monad\ (State\ s) \text{ where}$$
$$return\ a = State\ \$\ \lambda s \to (a, s)$$
$$m >\!\!>\!\!= f = State\ \$\ \lambda s \to \text{let } (a, s') = runState\ m\ s \text{ in } runState\ (f\ a)\ s'$$

$$get = State\ \$\ \lambda s \to (s, s)$$
$$put\ s = State\ \$\ \lambda\_ \to ((), s)$$

Evaluador con Estado

$$\text{data } \mathit{Exp} = \mathit{Num}\ \mathit{Int} \mid \mathit{Add}\ \mathit{Exp}\ \mathit{Exp} \mid \mathit{Var}\ \mathit{String} \mid \mathit{Assign}\ \mathit{String}\ \mathit{Exp}$$

$$\begin{aligned}
&\mathit{eval} :: \mathit{Exp} \to \mathit{State}\ [(\mathit{String}, \mathit{Int})]\ \mathit{Int} \\
&\mathit{eval}\ (\mathit{Num}\ n) &&= \mathit{return}\ n \\
&\mathit{eval}\ (\mathit{Add}\ e\ e') &&= \mathit{liftM2}\ (+)\ (\mathit{eval}\ e)\ (\mathit{eval}\ e') \\
&\mathit{eval}\ (\mathit{Var}\ v) &&= \text{do } s \leftarrow \mathit{get} \\
&&&\quad \mathit{return}\ (\mathit{fromJust}\ \$\ \mathit{lookup}\ v\ s) \\
&\mathit{eval}\ (\mathit{Assign}\ v\ e) &&= \text{do } a \leftarrow \mathit{eval}\ e \\
&&&\quad s \leftarrow \mathit{get} \\
&&&\quad \mathit{put}\ ((v, a) : s) \\
&&&\quad \mathit{return}\ a
\end{aligned}$$

Mónada Reader (`Control.Monad.Reader`)

Computaciones que leen valores de un ambiente compartido.

$$\text{data } \mathit{Exp} = \mathit{Num}\ \mathit{Int} \mid \mathit{Add}\ \mathit{Exp}\ \mathit{Exp} \mid \mathit{Var}\ \mathit{String} \mid \mathit{Let}\ \mathit{String}\ \mathit{Exp}\ \mathit{Exp}$$

$$\begin{aligned}
&\mathit{eval} :: \mathit{Exp} \to \mathit{Reader}\ [(\mathit{String}, \mathit{Int})]\ \mathit{Int} \\
&\mathit{eval}\ (\mathit{Num}\ n) = \mathit{return}\ n \\
&\mathit{eval}\ (\mathit{Add}\ e\ e') = \mathit{liftM2}\ (+)\ (\mathit{eval}\ e)\ (\mathit{eval}\ e') \\
&\mathit{eval}\ (\mathit{Var}\ v) = \text{do } s \leftarrow \mathit{ask} \\
&\qquad\qquad\qquad\qquad \mathit{return}\ (\mathit{fromJust}\ \$\ \mathit{lookup}\ v\ s) \\
&\mathit{eval}\ (\mathit{Let}\ v\ e\ b) = \text{do } a \leftarrow \mathit{eval}\ e \\
&\qquad\qquad\qquad\qquad \mathit{local}\ ((v, a):)\ (\mathit{eval}\ b)
\end{aligned}$$

Implementación de Mónada Reader

```
data Reader e a = Reader (e → a)
runReader (Reader r) = r
```

```
instance Monad (Reader e) where
  return a              = Reader $ λe → a
  (Reader r) >>= f = Reader $ λe → runReader (f (r e)) e
```

```
ask :: Reader e e
ask = Reader id
```

```
local :: (e → e) → Reader e a → Reader e a
local f c = Reader $ λe → runReader c (f e)
```

Preguntas