Abordaje Funcional a EDSLs

Alberto Pardo   Marcos Viera

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

ECI 2024

Modelado de efectos computacionales por mónadas

Expresiones con Fallas

$$\texttt{data}\ \mathit{Expr} = \mathit{Val}\ \mathit{Int} \mid \mathit{Add}\ \mathit{Expr}\ \mathit{Expr} \mid \mathit{Div}\ \mathit{Expr}\ \mathit{Expr}$$

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

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

$$\begin{array}{ll} \mathit{divM} & :: \mathit{Int} \to \mathit{Int} \to \mathit{Maybe}\ \mathit{Int} \\ a\ \text{`}\mathit{divM}\text{`}\ b & = \texttt{if}\ b \equiv 0\ \texttt{then}\ \mathit{Nothing} \\ & \phantom{= \texttt{if}\ b \equiv 0\ }\texttt{else}\ \mathit{Just}\ (a\ \text{`}\mathit{div}\text{`}\ b) \end{array}$$

Evaluador con Fallas

$$
\begin{array}{lll}
\mathit{eval} & :: \mathit{Expr} \to \mathit{Maybe\ Int} \\
\mathit{eval}\ (\mathit{Val}\ n) & = \mathit{Just}\ n \\
\mathit{eval}\ (\mathit{Add}\ x\ y) & = \mathbf{case}\ \mathit{eval}\ x\ \mathbf{of} \\
 & \qquad \mathit{Nothing} \to \mathit{Nothing} \\
 & \qquad \mathit{Just}\ a \quad \to \mathbf{case}\ \mathit{eval}\ y\ \mathbf{of} \\
 & \qquad\qquad\qquad \mathit{Nothing} \to \mathit{Nothing} \\
 & \qquad\qquad\qquad \mathit{Just}\ b \quad \to \mathit{Just}\ (a + b) \\
\mathit{eval}\ (\mathit{Div}\ x\ y) & = \mathbf{case}\ \mathit{eval}\ x\ \mathbf{of} \\
 & \qquad \mathit{Nothing} \to \mathit{Nothing} \\
 & \qquad \mathit{Just}\ a \quad \to \mathbf{case}\ \mathit{eval}\ y\ \mathbf{of} \\
 & \qquad\qquad\qquad \mathit{Nothing} \to \mathit{Nothing} \\
 & \qquad\qquad\qquad \mathit{Just}\ b \quad \to a\ `\mathit{divM}`\ b
\end{array}
$$

Evaluador con Fallas - Applicative

$$
\begin{array}{lll}
eval & :: & Expr \rightarrow Maybe\ Int \\
eval\ (Val\ n) & = & pure\ n \\
eval\ (Add\ x\ y) & = & (+)\ \texttt{<\$>}\ eval\ x\ \texttt{<*>}\ eval\ y \\
eval\ (Div\ x\ y) & = & \texttt{case}\ eval\ x\ \texttt{of} \\
 & & \quad Nothing \rightarrow Nothing \\
 & & \quad Just\ a \quad \rightarrow \texttt{case}\ eval\ y\ \texttt{of} \\
 & & \qquad\qquad\qquad Nothing \rightarrow Nothing \\
 & & \qquad\qquad\qquad Just\ b \quad \rightarrow a\ \text{`}divM\text{`}\ b
\end{array}
$$

Evaluador con Fallas - Applicative

$$
\begin{array}{ll}
eval & :: Expr \rightarrow Maybe\ Int \\
eval\ (Val\ n) & = pure\ n \\
eval\ (Add\ x\ y) & = (+) \mathbin{<\$>} eval\ x \mathbin{<*>} eval\ y \\
eval\ (Div\ x\ y) & = \mathbf{case}\ eval\ x\ \mathbf{of} \\
& \quad\quad Nothing \rightarrow Nothing \\
& \quad\quad Just\ a \quad\ \rightarrow \mathbf{case}\ eval\ y\ \mathbf{of} \\
& \quad\quad\quad\quad\quad\quad Nothing \rightarrow Nothing \\
& \quad\quad\quad\quad\quad\quad Just\ b \quad\ \rightarrow a\ `divM`\ b
\end{array}
$$

No puedo representar la división con Functores Aplicativos, necesito el resultado de una computación para determinar el siguiente efecto.

Capturemos patrones

Definamos:

$$\begin{aligned}
&return :: a \rightarrow Maybe\ a \\
&return\ a = Just\ a
\end{aligned}$$

$$\begin{aligned}
&(\ggg\!=) :: Maybe\ a \rightarrow (a \rightarrow Maybe\ b) \rightarrow Maybe\ b \\
&m \ggg\!= f = \mathtt{case}\ m\ \mathtt{of} \\
&\qquad\qquad Nothing \rightarrow Nothing \\
&\qquad\qquad Just\ a \quad \rightarrow f\ a
\end{aligned}$$

Evaluador con Fallas

Entonces,

$$
\begin{aligned}
&\mathit{eval} && :: \mathit{Expr} \rightarrow \mathit{Maybe\ Int} \\
&\mathit{eval}\ (\mathit{Val}\ n) && = \mathit{return}\ n \\
&\mathit{eval}\ (\mathit{Add}\ x\ y) && = \mathit{eval}\ x \gg\!= (\lambda a \rightarrow \mathit{eval}\ y \gg\!= (\lambda b \rightarrow \mathit{return}\ (a + b))) \\
&\mathit{eval}\ (\mathit{Div}\ x\ y) && = \mathit{eval}\ x \gg\!= (\lambda a \rightarrow \mathit{eval}\ y \gg\!= (\lambda b \rightarrow a\ `\mathit{divM}`\ b))
\end{aligned}
$$

Evaluador con Fallas

$$
\begin{aligned}
&eval && :: Expr \to Maybe\ Int \\
&eval\ (Val\ n) && = return\ n \\
&eval\ (Add\ x\ y) && = eval\ x \ggg\!= \lambda a \to \\
& && \quad eval\ y \ggg\!= \lambda b \to \\
& && \quad return\ (a + b) \\
&eval\ (Div\ x\ y) && = eval\ x \ggg\!= \lambda a \to \\
& && \quad eval\ y \ggg\!= \lambda b \to \\
& && \quad a\ `divM`\ b
\end{aligned}
$$

La clase Monad

$$
\begin{array}{l}
\texttt{class}\ \mathit{Applicative}\ m \Rightarrow \mathit{Monad}\ m\ \texttt{where} \\
\quad (\ggg\!=) \quad :: m\ a \to (a \to m\ b) \to m\ b \\
\quad (\gg) \quad :: m\ a \to m\ b \to m\ b \\
\quad \mathit{return}\ :: a \to m\ a
\end{array}
$$

$$m \gg k = m \ggg\!= \lambda\_ \to k$$

La clase Monad

$$
\begin{array}{l}
\texttt{class}\ \mathit{Applicative}\ m \Rightarrow \mathit{Monad}\ m\ \texttt{where} \\
\quad (\ggg\!=) \quad :: m\ a \to (a \to m\ b) \to m\ b \\
\quad (\gg) \quad\ \ :: m\ a \to m\ b \to m\ b \\
\quad \mathit{return}\ :: a \to m\ a
\end{array}
$$

$$m \gg k = m \ggg\!= \lambda\_ \to k$$

Toda mónada es un functor aplicativo que cumple:

- $\mathit{pure} = \mathit{return}$
- $m1$ <*> $m2 = m1 \ggg\!= (\lambda f \to m2 \ggg\!= (\lambda x \to \mathit{return}\ (f\ x)))$

La clase Monad

$$
\begin{array}{l}
\texttt{class}\ \mathit{Applicative}\ m \Rightarrow \mathit{Monad}\ m\ \texttt{where} \\
\quad (\gg\!=)\ \ :: m\ a \to (a \to m\ b) \to m\ b \\
\quad (\gg)\ \ \ :: m\ a \to m\ b \to m\ b \\
\quad \mathit{return} :: a \to m\ a
\end{array}
$$

$$m \gg k = m \gg\!= \lambda\_ \to k$$

Toda mónada es un functor aplicativo que cumple:

- $\mathit{pure} = \mathit{return}$
- $m1 \mathbin{<\!*\!>} m2 = m1 \gg\!= (\lambda f \to m2 \gg\!= (\lambda x \to \mathit{return}\ (f\ x)))$

No todo functor aplicativo es una mónada

Mónada Maybe

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

$$\begin{array}{l}
\mathtt{instance}\ \mathit{Monad}\ \mathit{Maybe}\ \mathtt{where} \\
\quad \mathit{return} \quad = \mathit{Just} \\
\quad m \gg\!\!= k = \mathtt{case}\ m\ \mathtt{of} \\
\qquad\qquad\qquad \mathit{Just}\ x \quad\ \to k\ x \\
\qquad\qquad\qquad \mathit{Nothing} \to \mathit{Nothing}
\end{array}$$

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

instance Functor Maybe where
  fmap f Nothing  = Nothing
  fmap f (Just a) = Just (f a)

instance Applicative Maybe where
  pure = Just
  (Just f) <*> (Just x) = Just (f x)
  _        <*> _        = Nothing
```

Mónada Maybe

```haskell
data Maybe a = Just a | Nothing

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

instance Functor Maybe where
  fmap f Nothing  = Nothing
  fmap f (Just a) = Just (f a)

instance Applicative Maybe where
  pure       = return
  m1 <*> m2 = m1 >>= (\f -> m2 >>= (\x -> return (f x)))
```

Leyes de mónadas

$$\begin{aligned}
\mathit{return}\ x \gg\!= f &= f\ x \\
m \gg\!= \mathit{return} &= m \\
(m \gg\!= f) \gg\!= g &= m \gg\!= \lambda x \to (f\ x \gg\!= g)
\end{aligned}$$

Composición de funciones monádicas

Composición de Kleisli.

$$(\gg\!\!=\!\!>) :: \mathit{Monad}\ m \Rightarrow (a \to m\ b) \to (b \to m\ c) \to a \to m\ c$$
$$f \gg\!\!=\!\!> g = \lambda a \to f\ a \gg\!= g$$

Propiedades:

$$\begin{aligned} \mathit{return} \gg\!\!=\!\!> f &= f \\ f \gg\!\!=\!\!> \mathit{return} &= f \\ f \gg\!\!=\!\!> (g \gg\!\!=\!\!> h) &= (f \gg\!\!=\!\!> g) \gg\!\!=\!\!> h \end{aligned}$$

Se prueban fácilmente usando las leyes de mónadas.

Notación do

$$
\begin{aligned}
\texttt{do}\ m &= m \\
\texttt{do}\ \{x \leftarrow m; m'\} &= m \gg\!= \lambda x \rightarrow \texttt{do}\ m' \\
\texttt{do}\ \{m; m'\} &= m \gg \texttt{do}\ m'
\end{aligned}
$$

Evaluador con Fallas (notación do)

$$
\begin{array}{l}
eval :: Expr \rightarrow Maybe\ Int \\
eval\ (Val\ n) \quad\ = return\ n \\
eval\ (Add\ x\ y) = \mathtt{do}\ a \leftarrow eval\ x \\
\qquad\qquad\qquad\qquad\ \ b \leftarrow eval\ y \\
\qquad\qquad\qquad\qquad\ \ return\ (a + b) \\
eval\ (Div\ x\ y)\ \ = \mathtt{do}\ a \leftarrow eval\ x \\
\qquad\qquad\qquad\qquad\ \ b \leftarrow eval\ y \\
\qquad\qquad\qquad\qquad\ \ a\ \text{`}divM\text{`}\ b
\end{array}
$$

Mónada Either

$$\texttt{data}\ \mathit{Either}\ a\ b = \mathit{Left}\ a \mid \mathit{Right}\ b$$

$$\begin{array}{l}
\texttt{instance}\ \mathit{Monad}\ (\mathit{Either}\ e)\ \texttt{where} \\
\quad \mathit{return} \qquad\qquad = \mathit{Right} \\
\quad \mathit{Left}\quad e \gg\!= _ = \mathit{Left}\ e \\
\quad \mathit{Right}\ a \gg\!= f = f\ a
\end{array}$$

Mónada Either

```
data Either a b = Left a | Right b
```

```
instance Monad (Either e) where
  return         = Right
  Left  e >>= _  = Left e
  Right a >>= f  = f a
```

Se corresponde con:

```
instance Applicative (Either e) where
  pure = Right
  Right f <*> Right a = Right (f a)
  Right f <*> Left e  = Left e
  Left e  <*> _       = Left e
```

Mónada Either

$$\mathtt{data}\ \mathit{Either}\ a\ b = \mathit{Left}\ a \mid \mathit{Right}\ b$$

$$\begin{array}{l}
\mathtt{instance}\ \mathit{Monad}\ (\mathit{Either}\ e)\ \mathtt{where} \\
\quad \mathit{return} \qquad\qquad = \mathit{Right} \\
\quad \mathit{Left}\ \ e \gg\!= _ = \mathit{Left}\ e \\
\quad \mathit{Right}\ a \gg\!= f = f\ a
\end{array}$$

Pero no con:

$$\begin{array}{l}
\mathtt{instance}\ \mathit{Monoid}\ e \Rightarrow \mathit{Applicative}\ (\mathit{Either}\ e)\ \mathtt{where} \\
\quad \mathit{pure} = \mathit{Right} \\
\quad \mathit{Right}\ f <\!*\!> \mathit{Right}\ a = \mathit{Right}\ (f\ a) \\
\quad \mathit{Left}\ e\ \ <\!*\!> \mathit{Right}\ _ = \mathit{Left}\ e \\
\quad \mathit{Right}\ _ <\!*\!> \mathit{Left}\ e\ \ = \mathit{Left}\ e \\
\quad \mathit{Left}\ e\ \ <\!*\!> \mathit{Left}\ e'\ \ = \mathit{Left}\ (e\ `\mathit{mappend}`\ e')
\end{array}$$

Mónada Either

$$\texttt{data}\ Either\ a\ b = Left\ a \mid Right\ b$$

$$\begin{array}{l}
\texttt{instance}\ Monad\ (Either\ e)\ \texttt{where} \\
\quad return \qquad\quad = Right \\
\quad Left\ \ \ e \gg\!= _ = Left\ e \\
\quad Right\ a \gg\!= f\ = f\ a
\end{array}$$

Pero no con:

$$\begin{array}{l}
\texttt{instance}\ Monoid\ e \Rightarrow Applicative\ (Either\ e)\ \texttt{where} \\
\quad pure = Right \\
\quad Right\ f \mathbin{\texttt{<*>}} Right\ a = Right\ (f\ a) \\
\quad Left\ e \ \ \mathbin{\texttt{<*>}} Right\ _ = Left\ e \\
\quad Right\ _ \mathbin{\texttt{<*>}} Left\ e \ \ = Left\ e \\
\quad Left\ e \ \ \mathbin{\texttt{<*>}} Left\ e' \ = Left\ (e\ `mappend`\ e')
\end{array}$$

$$Left\ a \mathbin{\texttt{<*>}} Left\ b = Left\ (a\ `mappend`\ b)$$

$$Left\ a \gg\!= (\lambda f \rightarrow Left\ b \gg\!= (\lambda x \rightarrow return\ (f\ x))) = Left\ a$$

Applicative no monádico

La instancia anterior de *Applicative* para *Either* **no** es una mónada.

$$
\begin{array}{l}
\mathtt{instance}\ \mathit{Monoid}\ e \Rightarrow \mathit{Monad}\ (\mathit{Either}\ e)\ \mathtt{where} \\
\quad \mathit{return} = \mathit{Right} \\
\quad \mathit{Left}\ e \gg\!= f\ = ?? \\
\quad \ldots
\end{array}
$$

Applicative no monádico

La instancia anterior de *Applicative* para *Either* **no** es una mónada.

$$
\begin{array}{l}
\texttt{instance}\ \mathit{Monoid}\ e \Rightarrow \mathit{Monad}\ (\mathit{Either}\ e)\ \texttt{where} \\
\quad \mathit{return} = \mathit{Right} \\
\quad \mathit{Left}\ e \gg\!= f = ?? \\
\quad \ldots
\end{array}
$$

- No podemos aplicar f en este caso porque sólo se aplica cuando la primera computación retorna un valor (*Right a*).
- Esto no ocurre en la instancia de *Applicative*.

Diferencia entre functores aplicativos y mónadas

La diferencia entre mónadas y functores aplicativos se puede apreciar en los siguientes operadores condicionales:

$$
\begin{array}{l}
ifM :: Monad\ m \Rightarrow m\ Bool \rightarrow m\ a \rightarrow m\ a \rightarrow m\ a \\
ifM\ mb\ mt\ me = \texttt{do}\ b \leftarrow mb \\
\qquad\qquad\qquad\qquad\quad \texttt{if}\ b\ \texttt{then}\ mt\ \texttt{else}\ me
\end{array}
$$

No todas computaciones se ejecutan (se elije entre *mt* y *me*)

Diferencia entre functores aplicativos y mónadas

La diferencia entre mónadas y functores aplicativos se puede apreciar en los siguientes operadores condicionales:

$$
\begin{aligned}
&\mathit{ifM} :: \mathit{Monad}\ m \Rightarrow m\ \mathit{Bool} \to m\ a \to m\ a \to m\ a\\
&\mathit{ifM}\ \mathit{mb}\ \mathit{mt}\ \mathit{me} = \mathbf{do}\ b \leftarrow \mathit{mb}\\
&\qquad\qquad\qquad\qquad\quad \mathbf{if}\ b\ \mathbf{then}\ \mathit{mt}\ \mathbf{else}\ \mathit{me}
\end{aligned}
$$

No todas computaciones se ejecutan (se elije entre *mt* y *me*)

$$
\begin{aligned}
&\mathit{ifA} :: \mathit{Applicative}\ f \Rightarrow f\ \mathit{Bool} \to f\ a \to f\ a \to f\ a\\
&\mathit{ifA}\ \mathit{fb}\ \mathit{ft}\ \mathit{fe} = \mathit{cond} <\$> \mathit{fb} <*> \mathit{ft} <*> \mathit{fe}\\
&\quad \mathbf{where}\\
&\qquad \mathit{cond}\ b\ t\ e = \mathbf{if}\ b\ \mathbf{then}\ t\ \mathbf{else}\ e
\end{aligned}
$$

Las tres computaciones (*fb*, *ft* y *fe*) se ejecutan y finalmente se elije uno de los resultados.

Mónada de estado

$$\texttt{newtype}\ State\ s\ a = State\ (s \to (a, s))$$

$$runState :: State\ s\ a \to (s \to (a, s))$$
$$runState\ (State\ f) = f$$

$$\begin{array}{l}
\texttt{instance}\ Monad\ (State\ s)\ \texttt{where} \\
\quad return\ a = State\ \$\ \lambda s \to (a, s) \\
\quad m \gg\!= f = State\ \$\ \lambda s \to \texttt{let}\ (a, s') = runState\ m\ s \\
\quad \texttt{in}\ \ runState\ (f\ a)\ s'
\end{array}$$

Forma alternativa de escribir la definición de $(\gg\!=)$:

$$\begin{array}{l}
(State\ g) \gg\!= f = State\ \$\ \lambda s \to \texttt{let}\ (a, s') \ = g\ s \\
\phantom{(State\ g) \gg\!= f = State\ \$\ \lambda s \to \texttt{let}\ } State\ k = f\ a \\
 \texttt{in}\ \ k\ s'
\end{array}$$

Funciones sobre estado

$$get :: State\ s\ s$$
$$get = State\ \$\ \lambda s \rightarrow (s, s)$$

$$put :: s \rightarrow State\ s\ ()$$
$$put\ s = State\ \$\ \lambda\_ \rightarrow ((), s)$$

$$modify :: (s \rightarrow s) \rightarrow State\ s\ ()$$
$$modify\ f = get \gg\!= \lambda s \rightarrow put\ (f\ s)$$

$$evalState :: State\ s\ a \rightarrow s \rightarrow a$$
$$evalState\ m\ s = fst\ \$\ runState\ m\ s$$

$$execState :: State\ s\ a \rightarrow s \rightarrow s$$
$$execState\ m\ s = snd\ \$\ runState\ m\ s$$

Ejemplo: contar número de sumas en una expresión

$$
\begin{aligned}
&\mathit{tick} :: \mathit{State}\ \mathit{Int}\ ()\\
&\mathit{tick} = \mathit{modify}\ (+1)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{evalS} :: \mathit{Expr} \rightarrow \mathit{State}\ \mathit{Int}\ \mathit{Int}\\
&\mathit{evalS}\ (\mathit{Val}\ n) = \mathit{return}\ n\\
&\mathit{evalS}\ (\mathit{Add}\ e\ e') = \mathtt{do}\ a \leftarrow \mathit{evalS}\ e\\
&\qquad\qquad\qquad\qquad\quad\ b \leftarrow \mathit{evalS}\ e'\\
&\qquad\qquad\qquad\qquad\quad\ \mathit{tick}\\
&\qquad\qquad\qquad\qquad\quad\ \mathit{return}\ (a + b)
\end{aligned}
$$

$$\mathit{nroSumas}\ e = \mathit{execState}\ (\mathit{evalS}\ e)\ 0$$

Evaluador con Variables

```
data Expr = Val Int
          | Add Expr Expr
          | Var ID            -- variables
          | Assign ID Expr    -- asignación

eval :: Expr → State (Map ID Int) Int
eval (Val n)        = return n
eval (Add e e')     = do a ← eval e
                         b ← eval e'
                         return (a + b)
eval (Var v)        = do s ← get
                         return (fromJust $ lookup v s)
eval (Assign v e)   = do a ← eval e
                         s ← get
                         put (insert v a s)
                         return a
```

Mónada de Estado

$$\texttt{class}\ \mathit{Monad}\ m \Rightarrow \mathit{MonadState}\ s\ m \mid m \to s\ \texttt{where}$$
$$\quad \mathit{get} :: m\ s$$
$$\quad \mathit{put} :: s \to m\ ()$$

$$\mathit{modify} :: \mathit{MonadState}\ s\ m \Rightarrow (s \to s) \to m\ ()$$
$$\mathit{modify}\ f = \texttt{do}\ s \leftarrow \mathit{get}$$
$$\qquad\qquad\qquad \mathit{put}\ (f\ s)$$

Mónada de Estado

```
class Monad m => MonadState s m | m -> s where
  get :: m s
  put :: s -> m ()
```

```
modify :: MonadState s m => (s -> s) -> m ()
modify f = do s <- get
              put (f s)
```

```
instance MonadState s (State s) where
  get   = State $ \s -> (s, s)
  put s = State $ \_ -> ((), s)
```

Mónada List

$$
\begin{array}{l}
\mathtt{instance}\ \mathit{Monad}\ [\,]\ \mathtt{where} \\
\quad \mathit{return}\ x = [x] \\
\quad xs \gg\!= f = [y \mid x \leftarrow xs, y \leftarrow f\ x] \\
\qquad\qquad\qquad \text{-- concat (map f xs)}
\end{array}
$$

Ejemplo: Suma de todos los pares de valores de dos listas

$$
\begin{array}{l}
\mathit{sumnd} :: \mathit{Num}\ a \Rightarrow [a] \to [a] \to [a] \\
\mathit{sumnd}\ xs\ ys = \mathtt{do}\ x \leftarrow xs \\
\qquad\qquad\qquad\quad\ \ y \leftarrow ys \\
\qquad\qquad\qquad\quad\ \ \mathit{return}\ (x + y)
\end{array}
$$

$> \mathit{sumnd}\ [1,3]\ [4,7]$
$[5,8,7,10]$