Programación Funcional Avanzada

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

Transformadores de Mónadas

¿Qué pasa si quiero combinar Fallas y Variables?

$$\begin{aligned}\text{data } \mathit{Exp} = {} & \mathit{Num}\ \mathit{Int} \mid \mathit{Add}\ \mathit{Exp}\ \mathit{Exp} \\ & \mid \mathit{Div}\ \mathit{Exp}\ \mathit{Exp} \\ & \mid \mathit{Var}\ \mathit{ID} \mid \mathit{Assign}\ \mathit{ID}\ \mathit{Exp}\end{aligned}$$

Debería de poder componer efectos.

Composición

La composición de functores es un functor.

$$fmap' :: (Functor\ f, Functor\ g) \Rightarrow (a \to b) \to f\ (g\ a) \to f\ (g\ b)$$
$$fmap'\ f\ x = fmap\ (fmap\ f)\ x$$

La composición de functores aplicativos es un functor aplicativo.

$$pure' :: (Applicative\ f, Applicative\ g) \Rightarrow a \to f\ (g\ a)$$
$$pure' = pure\ .\ pure$$

$$(<**>) :: (Applicative\ f, Applicative\ g) \Rightarrow f\ (g\ (a \to b)) \to f\ (g\ a) \to f\ (g\ b)$$
$$f\ <**>\ x = (<*>)\ <\$>\ f\ <*>\ x$$

La composición de dos mónadas puede no ser una mónada (pero es un aplicativo).

$$return' :: (Monad\ m, Monad\ n) \Rightarrow a \to m\ (n\ a)$$
$$return' = return\ .\ return$$

$$(>>>=) :: (Monad\ m, Monad\ n) \Rightarrow m\ (n\ a) \to (a \to m\ (n\ b)) \to m\ (n\ b)$$
$$mna\ >>>=\ f = mna \gg\!= \lambda na \to \text{let } nmnb = na \gg\!= \lambda a \to return\ (f\ a)$$
$$\text{in } fmap\ join\ (swap\ nmnb)$$

Transformadores de Mónadas

Recordemos como es la mónada de estado.

$$\text{newtype } State\ s\ a = State\ \{ runState :: s \to (a, s) \}$$

Queremos transformar una mónada m en una mónada de estado.

$$\text{newtype } StateT\ s\ m\ a = StateT\ \{ runStateT :: s \to m\ (a, s) \}$$

$$\begin{aligned}
&\text{instance } Monad\ m \Rightarrow Monad\ (StateT\ s\ m) \text{ where} \\
&\quad return\ a = StateT\ \$\ \lambda s \to return\ (a, s) \\
&\quad n \gg\!= k = StateT\ \$\ \lambda s \to \text{do } (a, s') \leftarrow runStateT\ n\ s \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad runStateT\ (k\ a)\ s'
\end{aligned}$$

Notar que se usa la mónada interna m en la definición de la instancia.

Transformadores de Mónadas

También definimos la instancia de *MonadState* para *StateT s m*:

$$
\begin{aligned}
&\text{instance } \mathit{Monad}\ m \Rightarrow \mathit{MonadState}\ s\ (\mathit{StateT}\ s\ m) \text{ where} \\
&\quad \mathit{get}\ \ \ = \mathit{StateT}\ \$\ \lambda s \rightarrow \mathit{return}\ (s, s) \\
&\quad \mathit{put}\ s = \mathit{StateT}\ \$\ \backslash\_ \rightarrow \mathit{return}\ ((), s)
\end{aligned}
$$

Evaluador: combina estado con error

$$
\begin{aligned}
&\mathit{eval} :: \mathit{Exp} \to \mathit{StateT}\ (\mathit{Map}\ \mathit{ID}\ \mathit{Int})\ \mathit{Maybe}\ \mathit{Int}\\
&\mathit{eval}\ (\mathit{Num}\ n) = \mathit{return}\ n\\
&\mathit{eval}\ (\mathit{Add}\ x\ y) = \mathrm{do}\ a \leftarrow \mathit{eval}\ x\\
&\qquad\qquad\qquad\qquad\quad b \leftarrow \mathit{eval}\ y\\
&\qquad\qquad\qquad\qquad\quad \mathit{return}\ (a + b)\\
&\mathit{eval}\ (\mathit{Var}\ v) = \mathrm{do}\ s \leftarrow \mathit{get}\\
&\qquad\qquad\qquad\qquad\quad \mathit{return}\ (\mathit{fromJust}\ \$\ \mathit{lookup}\ v\ s)\\
&\mathit{eval}\ (\mathit{Assign}\ v\ x) = \mathrm{do}\ a \leftarrow \mathit{eval}\ x\\
&\qquad\qquad\qquad\qquad\quad s \leftarrow \mathit{get}\\
&\qquad\qquad\qquad\qquad\quad \mathit{put}\ (\mathit{insert}\ v\ a\ s)\\
&\qquad\qquad\qquad\qquad\quad \mathit{return}\ a\\
&\\
&\mathit{eval}\ (\mathit{Div}\ x\ y) = \mathrm{do}\ a \leftarrow \mathit{eval}\ x\\
&\qquad\qquad\qquad\qquad\quad b \leftarrow \mathit{eval}\ y\\
&\qquad\qquad\qquad\qquad\quad \mathrm{if}\ b \equiv 0\ \mathrm{then}\ ????\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathrm{else}\ \ \mathit{return}\ (a\ `\mathit{div}`\ b)
\end{aligned}
$$

Acceso a la Mónada Interna

Tenemos que poder acceder a la mónada interna, en este caso *Maybe*.

```
...
eval (Div x y) = do a ← eval x
                    b ← eval y
                    if b ≡ 0
                      then (StateT $ λs → Nothing)
                      else return (a `div` b)
```

Haciendo esto estamos exhibiendo la estructura de la mónada.

Transformador de Mónada

Para evitar exhibir la estructura de las mónadas al combinarlas se define el concepto de *transformador de mónadas*.

```
class MonadTrans t where
  lift :: Monad m ⇒ m a → t m a
```

Es una operación entre mónadas que debe satisfacer las siguientes leyes:

$$\begin{aligned} lift\ (return\ a) &= return\ a \\ lift\ (m \gg\!= f) &= lift\ m \gg\!= (lift\ .\ f) \end{aligned}$$

En el caso de *StateT*:

```
instance MonadTrans (StateT s) where
  lift m = StateT $ λs → do a ← m
                            return (a, s)
```

Evaluador con *lift*

```
eval :: Exp → StateT (Map ID Int) Maybe Int
eval (Num n)      = return n
eval (Add x y)    = do a ← eval x
                       b ← eval y
                       return (a + b)
eval (Var v)      = do s ← get
                       return (fromJust $ lookup v s)
eval (Assign v x) = do a ← eval x
                       s ← get
                       put (insert v a s)
                       return a
eval (Div x y)    = do a ← eval x
                       b ← eval y
                       if b ≡ 0
                         then lift Nothing
                         else return (a `div` b)
```

StateT

El transformador *StateT* es instancia de:

$$\begin{aligned}
&\mathit{Monad}\ m \Rightarrow \mathit{Monad}\ (\mathit{StateT}\ s\ m)\\
&\mathit{Monad}\ m \Rightarrow \mathit{MonadState}\ s\ (\mathit{StateT}\ s\ m)\\
&\mathit{MonadTrans}\ (\mathit{StateT}\ s)
\end{aligned}$$

y es también instancia de:

$$\begin{aligned}
&\mathit{MonadWriter}\ w\ m \Rightarrow \mathit{MonadWriter}\ w\ (\mathit{StateT}\ s\ m)\\
&\mathit{MonadError}\ e\ m \Rightarrow \mathit{MonadError}\ e\ (\mathit{StateT}\ s\ m)\\
&\mathit{MonadReader}\ r\ m \Rightarrow \mathit{MonadReader}\ r\ (\mathit{StateT}\ s\ m)\\
&\mathit{MonadFix}\ m \Rightarrow \mathit{MonadFix}\ (\mathit{StateT}\ s\ m)\\
&\mathit{MonadPlus}\ m \Rightarrow \mathit{MonadPlus}\ (\mathit{StateT}\ s\ m)\\
&\mathit{MonadIO}\ m \Rightarrow \mathit{MonadIO}\ (\mathit{StateT}\ s\ m)\\
&\mathit{MonadCont}\ m \Rightarrow \mathit{MonadCont}\ (\mathit{StateT}\ s\ m)
\end{aligned}$$

Evaluador con *MonadError*

```
eval :: Exp → StateT (Map ID Int) Maybe Int
eval (Num n)      = return n
eval (Add x y)    = do a ← eval x
                       b ← eval y
                       return (a + b)
eval (Var v)      = do s ← get
                       return (fromJust $ lookup v s)
eval (Assign v x) = do a ← eval x
                       s ← get
                       put (insert v a s)
                       return a
eval (Div x y)    = do a ← eval x
                       b ← eval y
                       if b ≡ 0
                         then throwError ()
                         else return (a `div` b)
```

Evaluador Genérico

$$
\begin{aligned}
&\mathit{eval} :: (\mathit{MonadError}\ ()\ m, \mathit{MonadState}\ (\mathit{Map}\ \mathit{ID}\ \mathit{Int})\ m) \\
&\quad \Rightarrow \mathit{Exp} \to m\ \mathit{Int} \\
&\mathit{eval}\ (\mathit{Num}\ n) = \mathit{return}\ n \\
&\mathit{eval}\ (\mathit{Add}\ x\ y) = \text{do}\ a \leftarrow \mathit{eval}\ x \\
&\qquad\qquad\qquad\qquad\quad\ b \leftarrow \mathit{eval}\ y \\
&\qquad\qquad\qquad\qquad\quad\ \mathit{return}\ (a + b) \\
&\mathit{eval}\ (\mathit{Var}\ v) = \text{do}\ s \leftarrow \mathit{get} \\
&\qquad\qquad\qquad\qquad\quad\ \mathit{return}\ (\mathit{fromJust}\ \$\ \mathit{lookup}\ v\ s) \\
&\mathit{eval}\ (\mathit{Assign}\ v\ x) = \text{do}\ a \leftarrow \mathit{eval}\ x \\
&\qquad\qquad\qquad\qquad\quad\ s \leftarrow \mathit{get} \\
&\qquad\qquad\qquad\qquad\quad\ \mathit{put}\ (\mathit{insert}\ v\ a\ s) \\
&\qquad\qquad\qquad\qquad\quad\ \mathit{return}\ a \\
&\mathit{eval}\ (\mathit{Div}\ x\ y) = \text{do}\ a \leftarrow \mathit{eval}\ x \\
&\qquad\qquad\qquad\qquad\quad\ b \leftarrow \mathit{eval}\ y \\
&\qquad\qquad\qquad\qquad\quad\ \text{if}\ b \equiv 0\ \text{then}\ \mathit{throwError}\ () \\
&\qquad\qquad\qquad\qquad\qquad\qquad\ \text{else}\ \ \mathit{return}\ (a\ `\mathit{div}`\ b)
\end{aligned}
$$

Por ejemplo la instancia de *MonadPlus*, si la mónada transformada es instancia de *MonadPlus*, es:

```
instance (MonadPlus m) ⇒ MonadPlus (StateT s m) where
  mzero = StateT $ λ_ → mzero
  mplus (StateT m) (StateT n)
        = StateT $ λs → mplus (m s) (n s)
```

El orden importa

El orden en que las mónadas se combinan importa. La composición de dos transformadores t y t' en general no es conmutativa.

Por ejemplo, consideremos el siguiente transformador:

$$\text{newtype } \mathit{MaybeT}\ m\ a = \mathit{MaybeT}\ \{\mathit{runMaybeT} :: m\ (\mathit{Maybe}\ a)\}$$

que modela computaciones en la mónada m que dan resultados que pueden fallar (o sea, resultados en la mónada *Maybe*).

Las combinaciones

$$\mathit{StateT}\ s\ \mathit{Maybe}\ a$$
$$\mathit{MaybeT}\ (\mathit{State}\ s)\ a$$

dan respectivamente lo siguiente:

$$s \to \mathit{Maybe}\ (a, s)$$
$$s \to (\mathit{Maybe}\ a, s)$$