Transformadores de mónadas

Cecilia Manzino

19/11/2024

Uso

- Es un constructor de tipos que toma una mónada como argumento y devuelve una mónada.
- Es usado para combinar los efectos de las mónadas de manera modular.
- Un transformador T es una generalización de una mónada M, por ejemplo aplicando T a la mónada identidad se obtiene M.

Ejemplo: StateT

```haskell
-- Monada de estado
newtype State s a = St (s -> (a, s))

-- Combinamos State con  m
newtype StateT s m a = StT (s -> m (a, s))
runStT (StT f) = f

instance Monad m =>
         Monad (StateT s m) where
  return a = StateT $\s -> return (a,s)
  StT f >>= k =
     StateT $\s -> do (a,s') <- f s
                      runStT (k a) s'
```

Operaciones

Definimos las operaciones de la mónada `State` para *StateT*:

```
get :: Monad m => StateT s m s
get = StateT $\e -> return (e,e)

put :: Monad m => s -> StateT s m ()
put e = StateT $\_-> return ((),e)
```

Obs: Estas definiciones pueden definirse dentro de la instancia de `MonadState` para `StateT s m`.

Definición

Un transformador de mónada consiste en:

1. Un constructor de tipos `t :: (* -> *)-> * -> *`, tal que si `m` es una mónada `t m` también lo es.

2. Un operador `lift :: Monad m =>m a -> t m a` que satisface las siguientes leyes:

```
lift (return a) = return a
lift (m >>= k) = (lift m) >>= (lift.k)
```

La clase `MonadTrans` contiene como método esta operación:

```
class MonadTrans t where
    lift :: Monad m => m a -> t m a
```

Ejemplo: StateT

1. Damos la instancia de `Monad` para `StateT s m` (suponiendo que `m` es una mónada) y probamos que es una mónada.

   Definimos las operaciones de la mónada `State s` (`get` y `put`) cambiando el tipo monádico a `StateT s m`.

2. Definimos `lift` para `StateT s`.

```
instance MonadTrans (StateT s) where
  --lift :: Monad m => m a -> StateT s m a
  lift op = StateT $\s -> do a <- op
                             return (a,s)
```

Ejemplo de uso de StateT s

```haskell
data Exp = Lit Int
         | Var String
         | Let String Exp

type Env = [(String, Int)]
```

- El evaluador debe manejar un estado y controlar errores.
- La mónada `StateT Env Maybe` combina ambos efectos.
- La mónada `Maybe` tiene una sola operación: `throw`, elevamos esta operación usando `lift`:

```haskell
throwErr = lift throw
```

Ejemplo: Evaluador

```
eval :: Exp -> StateT Env Maybe Int
eval (Int n)   = return n

eval (Let v x) = do a <- eval x
                    s <- get
                    put ((v,a):s)
                    return a

eval (Var v) =
    do s <- get
       maybe throwErr return (lookup v s)
```

Otros transformadores de mónadas

```haskell
newtype MaybeT m a =
      MaybeT { runMaybeT :: m (Maybe a) }

newtype ReaderT r m a =
      ReaderT { runReaderT :: r -> m a}

newtype WriterT w m a =
      WriterT { runWriterT :: m (a,w) }
```

Ejercicio: Dar la instancia de `Monad` para `MaybeT m`, suponiendo que `m` es una mónada.

Combinando 3 mónadas

```haskell
type TripleMonad a =
        MaybeT (ReaderT Env IO) a

-- nos olvidamos de las capas
fromReader :: ReaderT Env IO a ->
                    TripleMonad a
fromReader = lift

fromIO :: IO a -> TripleMonad a
fromIO = lift . lift
```

Obs: La mónada `IO` es siempre la mónada más interna.