Análisis de Lenguajes de Programación TP4

Belmonte Marina

Ejercicio 1

```haskell
newtype State a = State { runState :: Env -> Pair a Env }

instance Monad State where
  return x = State (\s -> (x :!: s))
  m >>= f = State (\s -> let (v :!: s') = runState m s in runState (f v) s')
```

Veamos que **State** es una mónada. Para ello demostraremos que su instancia verifica leyes de las mónadas:

(Monad.1) return x »= f = f x

(Monad.2) m »= return = m

(Monad.3) (m »= f) »= g = m »= $(\lambda x \xrightarrow{f} x >>= g)$

Monad.1

```
return x >>= f
= < def return >
State (\s -> (x :!: s)) >>= f
= < def >>= >
State (\s -> let (v :!: s') = runState State (\s -> (x :!: s)) s
                    in runState (f v) s')
= < def runState >
State (\s -> let (v :!: s') = (\s -> (x :!: s)) s i
                    n runState (f v) s')
= < beta-redex >
State (\s -> let (v :!: s') = (x :!: s)
                    in runState (f v) s')
= < def let >
State (\s -> runState (f x) s)
= < eta-redex >
State (runState (f x))
= < State . runState = id >
f x
```

Monad.2

```
m >>= return
= < def >>= >
State (\s -> let (v :!: s') = runState m s
                    in runState (return v) s')
= < def return >
State (\s -> let (v :!: s') = runState m s
                    in runState (State (\s -> (v :!: s))) s')
= < def runState >
State (\s -> let (v :!: s') = runState m s
                    in  (\s -> (v :!: s)) s')
= < beta-redex >
State (\s -> let (v :!: s') = runState m s
                    in  (v :!: s'))
= < def let >
State (\s -> runState m s)
= < eta-redex >
State (runState m)
= < State . runState = id >
m
```

Monad.3

```
(m >>= f) >>= g
= < def >>= >
State (\s -> let (v :!: s') = runState m s
                    in runState (f v) s') >>= g
= < def >>= >
State (\t -> let (u :!: t') = runState (State (\s -> let (v :!: s') = runState m s
                                                        in runState (f v) s'))  t
                    in runState (g u) t')
= < def runState >
State (\t -> let (u :!: t') = (\s -> let (v :!: s') = runState m s
                                        in runState (f v) s')  t
                    in runState (g u) t')
= < beta-redex >
State (\t -> let (u :!: t') = let (v :!: s') = runState m t
                                 in runState (f v) s'
                    in runState (g u) t')

= < * >
State (\s -> let (v :!: s') = runState m s
                    in (let (u :!: t') = runState (f v) s'
                            in runState (g u) t' ))
= < beta-redex >
State (\s -> let (v :!: s') = runState m s
                    in (\t -> let (u :!: t') = runState (f v) t
                                 in runState (g u) t') s')
= < def runState >
State (\s -> let (v :!: s') = runState m s
                    in runState (State (\t -> let (u :!: t') = runState (f v) t
                                                 in runState (g u) t')) s')
= < def >>= >
State (\s -> let (v :!: s') = runState m s
                    in runState (f v >>= g ) s')
= < beta-redex >
```

```
State (\s -> let (v :!: s') = runState m s
                    in runState ((\x -> f x >>= g ) v) s')
= < def >>= >
m >>= (\x -> f x >>= g )
```

(*) Considerando que vale el siguiente resultado:

Si $y \notin FV(gx)$, entonces:

```
let x = let y = f                let y = f
            in h y      sii          in let x = h y
    in g x                                  in g x
```

, y tomando

```
x = (u :!: t')
y = (v :!: s')
f = runState m t
g = (\(u :!: t') -> runState (g u) t')
h = (\(v :!: s') -> runState (f v) s')
```

podemos ver que $y \notin FV(gx)$, por lo que podemos utilizar el resultado.

Por lo tanto queda demostrado que se cumplen las tres reglas, es decir que **State** es una mónada.