Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Análisis de lenguajes de programación

Trabajo Práctico 4

Autor:
Tomás Castro Rojas

2 de diciembre de 2021

1. Ejercicio 1

Recordemos la definición de `State`:

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

y su instancia de Mónada:

```haskell
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')
```

Vamos a demostrar que `State` es una mónada. Para ello basta probar que su instancia cumple con las siguientes 3 propiedades:

1. return x $\gg=$ f = f x
2. t $\gg=$ return = t
3. (t $\gg=$ f) $\gg=$ g = t $\gg=$ ($\lambda x \rightarrow$ f x $\gg=$ g)

Comenzamos con la prueba

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 in runState (f v) s')
= < B-redex >
State (\ s -> let (v :!: s') = (x :!: s) in runState (f v) s')
= < def Let >
State (\ s -> runState (f x) s)
= < E-redex >
State (runState (f x))
= < State . runState = Id >
f x
```

Monad.2

```
t >>= return
= < def >>= >
State (\ s -> let (v :!: s') = runState t s in runState (return v) s')
= < def return >
State (\ s -> let (v :!: s') = runState t s in runState State (\ s -> (v :!: s)) s')
= < def runState >
State (\ s -> let (v :!: s') = runState t s in (\ s -> (v :!: s)) s')
= < B-redex >
State (\ s -> let (v :!: s') = runState t s in (v :!: s'))
= < def Let >
State (\ s -> runState t s)
```

```
= < def E-redex >
State (runState t)
= < State . runState = Id >
t
```

Monad.3

Desarrollamos primero el lado izquierdo

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

Ahora desarrollamos el lado derecho

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

Veamos que las expresiones obtenidas son equivalentes. Para esto nos valemos del siguiente Lema:
Si y $\notin FV($g' x$)$ entonces

```
let x = let y = f'              let y = f'
        in h' y         <=>     in let x = h' y
in g' x                            in g' x
```

Tomando la expresión obtenida del lado izquierdo (*) tenemos:

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

Es facil ver que efectivamente `y` $\notin FV($`g' x`$)$ entonces reemplazando nos queda

```
State (\ e -> let (u :!: c) = (let (v :!: s') = runState t e in runState (f v) s')
in runState (g u) c) (*)
= < Lema >
State (\ e -> let (v :!: s') = runState t e in
(let (u :!: c) = runState (f v) s' in runState (g u) c)) (**)
```

Con lo cual concluimos que `(t >>= f) >>= g = t >>= (\barra x -> f x >>= g)`.
Por lo tanto, al cumplirse las 3 propiedades, `State` es una mónada.