Trabajo Practico 4

Facundo Emmanuel Messulam
Franco Ignacio Vallejos Vigier

6 de diciembre de 2021

1. Ejercicio 1

1.1. Apartado A

```
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')
```

1.1.1. Propiedad 1

Pruebo que: `return x >>= f = f x`.

```
return x = State (\s -> (x :!: s))
```

(Por definicion de return en `Monad State`)

```
return x >>= f = State (\s -> (x :!: s)) >>= f
```

(Uso bind en ambos lados: `>>= f`)

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

(Por definicion de bind en `Monad State`)

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

(Por definicion runState conviente un `State a` en su isomorfismo `Env -> Pair a Env`)

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

```
return x >>= f = State (\s -> runState (f x) s)
```

```
return x >>= f = f x
```

(Por definicion runState conviente un `State a` en su isomorfismo `Env -> Pair a Env` y remuevo un lambda redundante)

1.1.2. Propiedad 2

Pruebo que: `t >>= return = t`.

```
t >>= return = State (\s -> let (v :!: s') = runState t s in runState (return v) s')
```

(Definicion de `>>=`)

```
t >>= return = State (\s -> let (v :!: s') = runState t s in runState (State(\s -> (v :!: s) ) s')
```

(Definicion de return)

```
t >>= return = State (\s -> let (v :!: s') = runState t s in (v :!: s')
```

(Definicion de runState)

```
t >>= return = State(\s -> runState t s)
```

(Definicion de let)

```
State(runState t)
```

(Definicion de *eta*-reduccion)

```
t
```

`t = State h` si y solo si `runState(State h) = h` entonces `State(runState(State h)) = State h = t`.

1.1.3. Propiedad 3

Pruebo que: `(t >>= f) >>= g  = t >>= (\x -> f x >>= g)`
Propiedad 3.A

```
(t >>= f) >>= g
```

(Definición de `>>=`)

```
(State (\s -> let (v :!: s') = runState t s
              in runState (f v) s')) >>= g
```

(Definición de `>>=`)

```
State (\w -> let (u :!: w') = runState (State (\s -> let (v :!: s') = runState t s
                                                     in runState (f v) s')) w
              in runState (g u) w')
```

(Definición de `runState`)

```
State (\w ->
    let (u :!: w') = (\s -> let (v :!: s') = runState t s
                                in runState (f v) s') w
    in runState (g u) w')
```

(por β-reduccion y α-conversión)

```
State (\s ->
    let (u :!: w') = let (v :!: s') = runState t s
                     in runState (f v) s'
    in runState (g u) w')
```

Propiedad 3.B Por practicidad, podemos arrancar del otro lado de la igualdad y ver si llegamos a lo mismo:

```
t >>= (\x -> f x >>= g)
```

(Definicion de `>>=`, tomando v y $s' \notin (\mathrm{FV}(f) \cup \mathrm{FV}(.f\ x) \cup \mathrm{FV}(g))$ (llamo (1))

```
State (\s -> let (v :!: s') = runState t s
                       in runState ((\x. f x >>= g) v) s')
```

(β-reducción)

```
State (\s -> let (v :!: s') = runState t s
              in runState (f v >>= g)  s')
```

(Definición de `>>=`)

```
State (\s -> let (v :!: s') = runState t s
              in runState (State (\w -> let (u :!: w') = runState (f v) w
                                         in runState (g u) w')
                          )  s')
```

(Definición de `runState`)

```
State (\s -> let (v :!: s') = runState t s
              in (\w -> let (u :!: w') = runState (f v) w
                        in runState (g u) w') s')
```

(β-reducción)

```
State (\s -> let (v :!: s') = runState t s
              in  let (u :!: w') = runState (f v) s'
                  in runState (g u) w'
```

Propiedad let:

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

(mientras que $y \notin \mathrm{FV}(g\ x)$)

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

Por (1) podemos ver que $(v :! : s') \notin \mathrm{FV}(\mathrm{runState}(gu)w')$ (ya que esta expresión se derivó de las iniciales), entonces nos queda:

(Propiedad de Let enunciada)

```
State (\s ->
    let (u :!: w') = let (v :!: s') = runState t s
                     in runState (f v) s'
    in runState (g u) w')
```

Hemos llegado a lo mismo desde ambos lados de igualdad entre la parte A y la parte B de la propiedad 3, por lo queda demostrado.

2. Ejercicio 1b, 2 y 3

Hecho en la carpeta `src`.