Trabajo Práctico 1

Análisis de lenguajes de programación

LCC

Manuel Spreutels
Augusto Rabbia

Septiembre, 2024

1 Soluciones

1.1 Ejercicio 1

Apartado (a), demostremos que State es un mónada:

(monad.1): $return\ x \gg\!= f = f\ x$

```
return x ≫= f                                              = {def return}
State (λs → let (v:!:s') = runState State (λs → (x:!:s)) s
            in runState (f v) s')                          = {def ≫=}
State (λs → let (v:!:s') = (λs → (x:!:s)) s
            in runState (f v) s')                          = {aplicacion}
State (λs → let (v:!:s') = (x:!:s)
            in runState (f v) s')                          = {let}
State (λs → runState (f x) s)                              = {beta reduccion}
State runState (f x)                                       = {State.runState = id}
f x
```

(monad.2): $x \gg\!= return = x$

```
Demostramos hacia abajo
m ≫= return                                         = {def ≫=}
State (λs → let (v :!: s') = runState m s
            in runState (return v) s')              = {def return}
State (λs → let (v :!: s') = runState m s
            in runState (State (λe → (v :!: e))) s')  = {runState.State = id}
State (λs → let (v :!: s') = runState m s
            in (λe → (v :!: e)) s')                 = {aplicacion}
State (λs → let (v :!: s') = runState m s
            in (v :!: s'))                          = {let, (v :!: s') = runState m s}
State (λs → runState m s)                           = {beta reduccion}
State (runState m)                                  = {State.runState = id}
m
```

(monad.3): $m \gg\!= (\lambda x \to k\ x \gg\!= h) = (m \gg\!= k) \gg\!= h$

```
m ≫= (λx → k x ≫= h)                                   = {def bind}
State (λs → let (v :!: s') = runState m s
            in runState ((λx → k x ≫= h) v) s')        = {aplicacion}
State (λs → let (v :!: s') = runState m s
            in runState ((k v) ≫= h) s')               = {def bind}
```

```
State (λs1 → let (v1 :!: s1') = runState m s1
              in runState
              (State (λs2 → let (v2 :!: s2') = runState (k v1) s2
                             in runState (h v2) s2') s1'))  = {-{aplicacion,
                                                               runState.State = id} -}
State (λs1 → let (v1 :!: s1') = runState m s1
              in let (v2 :!: s2') = runState (k v1) s1'
                 in runState (h v2) s2')                   = {renombramos s1 por s2}
State (λs2 → let (v1 :!: s1') = runState m s2
              in let (v2 :!: s2') = runState (k v1) s1'
                 in runState (h v2) s2')                   = {let}
State (λs2 → let (v1 :!: s1') = runState m s2
                 (v2 :!: s2') = runState (k v1) s1'
              in runState (h v2) s2')
=====================================================
State (λs2 → let (v1 :!: s1') = runState m s2
                 (v2 :!: s2') = runState (k v1) s1'
              in runState (h v2) s2')
State (λs2 → let (v2 :!: s2') =                            = {let}
                    let (v1 :!: s1') = runState m s2
                    in runState (k v1) s1'
             in runState (h v2) s2')
State (λs2 → let (v2 :!: s2') =                            = {aplicacion}
                    (λs1 → let (v1 :!: s1') = runState m s1
                            in runState (k v1) s1') s2
              in runState (h v2) s2') >>= h
State (λs2 → let (v2 :!: s2') = runState                   = {runState.State = id}
                            (State (λs1 → let (v1 :!: s1') = runState m s1
                                           in runState (k v1) s1')) s2
             in runState (h v2) s2') >>= h
State (λs → let (v :!: s') = runState m s                  = {def bind}
             in runState (k v) s') >>= h
(m >> k) >>= h                                             = {def bind}
Demostramos hacia arriba
```