Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Programación Monádica

Cuarto trabajo práctico
Análisis del Lenguajes de Programación

Autores:
Arroyo, Joaquín
Caporalini, Joaquín

29 de noviembre de 2022

Se define la monada estado, para representar una computación que tiene acceso al estado del programa.

```
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 ahora que *State* es una monada.

monad.1

```
return x >>= f =
<def.return>
State (\s -> (x :!: s)) >>= f =
<def.bind>
State (\s -> let (v :!: s') = runState State (\s -> (x :!: s)) s
              in runSate (f v) s') =
<def.runState>
State (\s -> let (v :!: s') = (\s -> (x :!: s)) s
              in runSate (f v) s') =
<app>
State (\s -> let (v :!: s') = (x :!: s)
              in runSate (f v) s') =
<def.let>
State (\s -> runSate (f x) s) =
<eta-reduccion>
State runSate (f x) =
<def.runState>
f x
```

monad.2

```
m >>= return
<def.bind>
State (\s -> let (v :!: s') = runState m s
              in runSate (return v) s') =
<def.bind>
State (\s -> let (v :!: s') = runState m s
              in runSate (State (\s -> (v :!: s))) s') =
<def.bind>
State (\s -> let (v :!: s') = runState m s
              in (\s -> (v :!: s)) s') =
<def.runState>
State (\s -> let (v :!: s') = runState m s
              in (v :!: s')) =
<def.let>
State (\s -> runState m s) =
<def.let>
State runState m =
<def.runState>
m
```

monad.3

```
m >>= (\x -> f x >>= g) =
<def.bind> (el mas a la izquierda)
State (\s -> let (v :!: s') = runState m s
              in runState ((\x -> f x >>= g) v) s') =
<app>
State (\s -> let (v :!: s') = runState m s
              in runState (f v >>= g) s') =
<def.bind>
State (\s -> let (v :!: s') = runState m s
              in let (v' :!: s'') = runState (f v) s'
                  in runState (g v') s'') =
<prop.let>
State (\s -> let (v :!: s')   = runState m s
                 (v' :!: s'') = runState (f v) s'
              in runState (g v') s'') =
<prop.let>
State (\s -> let (v :!: s') = let (v' :!: s'') = runState m s
                               in runState (f v') s''
              in runState (g v) s') =
<app>
State (\s -> let (v :!: s') = (\s' -> let (v' :!: s'') = runState m s'
                                       in runState (f v') s'') s
              in runState (g v) s') =
<def.runState>
State (\s -> let (v :!: s') = runState State (\s' -> let (v' :!: s'') = runState m s'
                                                      in runState (f v') s'') s
              in runState (g v) s') =
<def.bind>
State (\s -> let (v :!: s') = runState (m >>= f) s
              in runState (g v) s') =
<def.bind>
(m>>=f)>>=g
```