# Trabajo Práctico N°4

Agustín Gurvich, Santiago Wirzt

13/11/2020

## Demostración de mónada State

Queremos demostrar las 3 leyes de mónadas para la mónada State

```
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')
```

### 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

```
State a >>= return
<=> def >>=
State (\s -> let (v :!: s') = runState (State a) s in runState (return v) s')
<=> def return
State (\s -> let (v :!: s') = runState (State a) s in runState (State (\s -> (v :!: s))) s')
<=> def runState
State (\s -> let (v :!: s') = runState (State a) s in (\s -> (v :!: s)) s')
<=> B-redex
State (\s -> let (v :!: s') = runState (State a) s in (v :!: s'))
<=> def runState
State (\s -> let (v :!: s') = a s in (v :!: s'))
<=> def let
State (\s -> a s)
<=> E-redex
State a
```

## monad.3

```
(State a >>= f) >>= g
<=> def de >>= f
(State (\s -> let (v :!: s') = runState (State a) s in runState (f v) s')) >>= g
<=> def de runState
(State (\s -> let (v :!: s') = a s in runState (f v) s')) >>= g
<=> def de >>= g
State (\t -> let (u :!: t') = runState (State (\s -> let (v :!: s') = a s
                                                        in runState (f v) s')) t
             in runState (g u) t')
<=> def de runState
State (\t -> let (u :!: t') = (\s -> let (v :!: s') = a s
                                      in runState (f v) s') t
             in runState (g u) t')
<=> B-Redex
State (\t -> let (u :!: t') = let (v :!: s') = a t in runState (f v) s' in runState (g u) t')

State a >>= (\x -> f x >>= g)
<=> def >>=
State (\s -> let (v :!: s') = runState (State a) s in runState ((\x->f x >>= g) v) s')
<=>def runState
State (\s -> let (v :!: s') = a s in runState ((\x->f x >>= g) v) s')
<=> B-Redex
State (\s -> let (v :!: s') = a s in runState (f v >>= g) s')
<=> def >>=
State (\s -> let (v :!: s') = a s
             in runState (State (\t -> let (u :!: t') = runState (f v) t
                                       in runState (g u) t')) s')
<=> def runState
State (\s -> let (v :!: s') = a s
             in (\t -> let (u :!: t') = runState (f v) t
                       in runState (g u) t') s')
<=> B-Redex
State (\s -> let (v :!: s') = a s in let (u :!: t') = runState (f v) s' in runState (g u) t')
<=> A-conversion
State (\t -> let (v :!: s') = a t in let (u :!: t') = runState (f v) s' in runState (g u) t')
```

Observamos que partiendo de ambas igualdades llegamos a un punto donde podemos aplicar el siguiente lema:
**Lema:** Si $y \notin FV(g\,x)$ entonces:

```
let x = let y = f1          let y = f1
        in h y       <=>    in let x = h y
in g1 x                        in g1 x
```

Así, si elegimos $v, s' \notin FV(f) \cup FV(g)$ tenemos que:

- x = (u :!: t')
- y = (v :!: s')
- f1 = a t
- h = ( \(v :!: s') → runState (f v) s')
- g1 = (\(u :!: t') → runState (g u) t')

Por lo que ahora tenemos:

```
let y = f1
in let x = h y
   in g1 x
<=> Reemplazamos por lo planteado anteriormente
State (\t -> let (v :!: s') = a t
in let (u :!: t') = (\v,s' -> runState (f v) s') (v :!: s')
   in (\u,t' -> runState (g u) t') (u :!: t'))
<=> B-Redex
State (\t -> let (v :!: s') = a t
in let (u :!: t') = runState (f v) s'
```

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

Con lo cual queda demostrado monad.3