State Monad Example

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Objectives

- Define `get` and `put`.
- Write some stateful computations using the state monad.

The Definition

```haskell
data State s a = State { runState :: s -> (a,s) }

instance Monad (State s) where
  return = pure -- or ... return a = State (\s -> (a,s))
  x >>= f = State (\s -> let (y,s2) = runState x s
                             (z,s3) = runState (f y) s2
                          in (z,s3))
```

Incrementing a State, 1

- `State s a = State { runState :: s -> (a,s) }`
- How can we write something that will increment the s component?

```haskell
incState (State f) = ...
```

Incrementing a State, 2

- `State s a = State { runState :: s -> (a,s) }`
- How can we write something that will increment the s component?

```
incState (State f) = State (\s -> let (x,s0) = f s in (x, s0+1))
```

or …

```
incState f = State (\s -> let (x,s0) = runState f s in (x, s0+1))
```

Sample run:

```
*Main> let e1 = State (\s -> (5,s))
*Main> incState (State f) = State (\s -> let (x,s0) = f s in (x,s0+1))
*Main> runState (incState e1) 0
(5,1)
```

Two Common Functions

- Two common functions:

```haskell
get :: State s s
get = State (\s -> (s,s))

put :: a -> State a ()
put x = State (\s -> ((),x))
```

```
*Main> runState (State (\s -> (5,s)) >>= \v -> get) 8
(8,8)
*Main> runState (State (\s -> (5,s)) >>= put) 8
((),5)
```

Tracing Get

```
(State (\s -> (5,s)) >>= \v -> get)
```

```
State (\s1 -> let (x,s2) = (\s -> (5,s)) s1
                  (y,s3) = runState ((\v -> get)  x) s2
               in (y,s3)
```

```
State (\s1 -> let (x,s2) = (5,s1)
                  (y,s3) = runState ((\v -> get)  x) s2
               in (y,s3)
```

```
State (\s1 -> let (y,s3) = runState ((\v -> get) 5) s1
               in (y,s3)
```

```
State (\s1 -> (\s -> (s,s)) s1)
```

```
State (\s1 -> (s1,s1))
```

Tracing Put

```
(State (\s -> (5,s)) >>= put)
```

```
State (\s1 -> let (x,s2) = (\s -> (5,s)) s1
                  (y,s3) = runState (put x) s2
               in (y,s3)
```

```
State (\s1 -> let (x,s2) = (5,s1)
                  (y,s3) = runState (put x) s2
               in (y,s3)
```

```
State (\s1 -> let (y,s3) = runState (put 5) s1
               in (y,s3)
```

```
State (\s1 -> (\s -> ((),5)) s1)
```

```
State (\s1 -> ((),5))
```

Using Do Notation

- Bind notation can be cumbersome.
- Do notation to the rescue!

```
mmul a b = do
    x <- a         -- == a >>= \x -> (
    y <- b         --    b >>= \y ->
    return (x*y)   --      return (x*y))

Prelude> mmul [10] [20]
[200]
Prelude> mmul [10] [20,40]
[200,400]
Prelude> mmul (Just 5) Nothing
Nothing
```

Do Notation for States

```
sar x = do
    put (x*2)
    let y = 10
    z <- get        -- >>= \z -> get
    return y * z
```

```
*Main> runState (sar 7) 0
(140,14)
```