State Monad (3D)

Young Won Lim
8/22/17

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Based on

Haskell in 5 steps
https://wiki.haskell.org/Haskell_in_5_steps

State Monad

The Haskell type **State** describes **functions**
that take a state
and return both a result and an updated state,
which are given back in a tuple.

The **state function** is wrapped by a data type definition
which comes along with a **runState accessor**
no need for pattern matching


```
newtype State s a = State { runState :: s -> (s, a) }
                            accessor
```

Control.Monad.**Trans**.**State**, **transformers** package. (focused here)
Control.Monad.**State**, **mtl** package.

https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

State Monad

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

s : the type of the state,
a : the type of the produced result
s -> (a, s) : function type

State String,
State Int,
State SomeLargeDataStructure,
and so forth.

Calling the type **State** is arguably a bit of a misnomer
because the wrapped value is not the state itself
but a **state processor** (**accessor function**: **runState**)

https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

State Monad – state function

Control.Monad.**Trans**.**State**, **transformers** package. (focused here)

no State constructor

but a **state function**

state :: (s -> (s, a)) -> **State** s a

Control.Monad.**State**, **mtl** package

Implements the State in somewhat different way

https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

Instantiating a State Monad

to **wrap a function type** and give it a name.

for every type s, **State** s can be made a *Monad instance*,

the *instance* is **State** s, and not just **State**

(**State** can't be made an instance of Monad,
as it takes two type parameters, rather than one.)

State String,
State Int,
State SomeLargeDataStructure,
and so forth.

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

instance Monad (State s) where
    return
    (>>=);
```

https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

Instantiating a State Monad

instance **Monad** (**State** s) where

many different **State** monads,
one for each possible type of state -

- **State** String,
- **State** Int,
- **State** SomeLargeDataStructure,
- and so forth.

only need to write one implementation of

- **return** and
- (**>>=**);

these methods will be able to deal with all choices of s.

https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

State Monad – return method

```
instance Monad (State s) where

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

giving a value (x) to **return** produces a function **state**
which takes a state (s) and returns it unchanged,
together with value x we want to be returned.

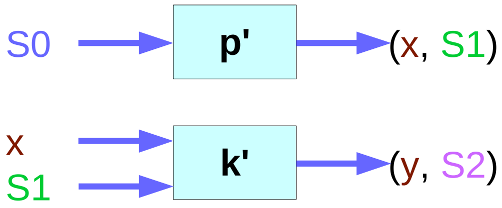
As a finishing step, the function is wrapped up with the state function.

```
state :: (s -> (a, s)) -> State s a

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

https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

State Monad – binding operator

```
instance Monad (State s) where

(>>=) :: State s a -> (a -> State s b) -> State s b
p >>= k = q where
    p' = runState p            -- p' :: s -> (a, s)
    k' = runState . k          -- k' :: a -> s -> (b, s)
    q' s0 = (y, s2) where      -- q' :: s -> (b, s)
        (x, s1) = p' s0        -- (x, s1) :: (a, s)
        (y, s2) = k' x s1      -- (y, s2) :: (b, s)
    q = state q'
```

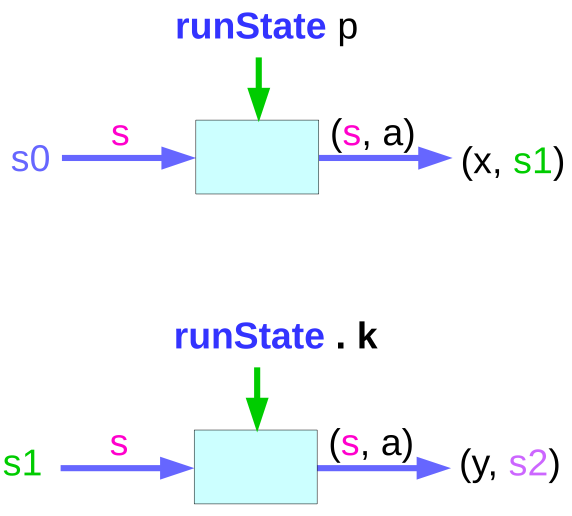



```
p  >>= k = state $ \ s0 ->
   let (x, s1) = runState p s0    -- running the first processor on s0.
   in runState (k x) s1           -- running the second processor on s1.
```

https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

State Monad – binding operator

```
instance Monad (State s) where

(>>=) :: State s a -> (a -> State s b) -> State s b
p >>= k = q where
    p' = runState p              -- p' :: s -> (a, s)
    k' = runState . k            -- k' :: a -> s -> (b, s)
    q' s0 = (y, s2) where        -- q' :: s -> (b, s)
        (x, s1) = p' s0          -- (x, s1) :: (a, s)
        (y, s2) = k' x s1        -- (y, s2) :: (b, s)
    q = state q'
```

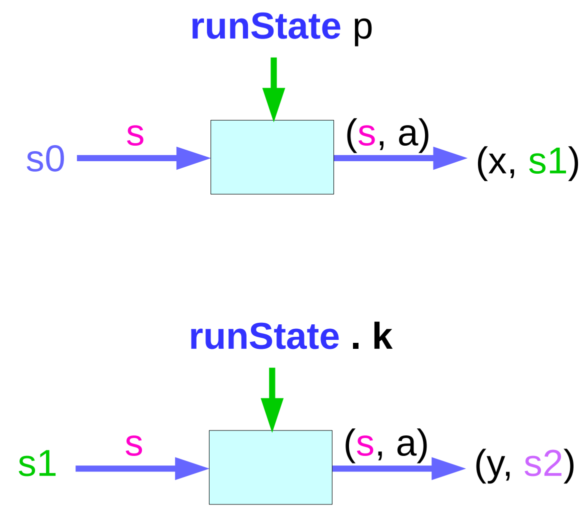



```
newtype State s a = State { runState :: s -> (s, a) }

state :: (s -> (a, s)) -> State s a
```

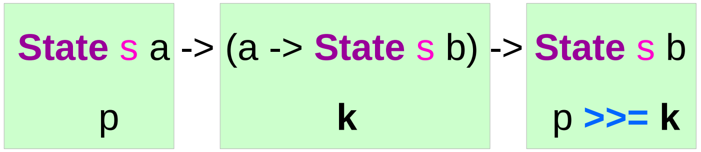
https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

State Monad – binding operator

```
instance Monad (State s) where

(>>=) :: State s a -> (a -> State s b) -> State s b
p >>= k = q where
    p' = runState p             -- p' :: s -> (a, s)
    k' = runState . k           -- k' :: a -> s -> (b, s)
    q' s0 = (y, s2) where       -- q' :: s -> (b, s)
        (x, s1) = p' s0         -- (x, s1) :: (a, s)
        (y, s2) = k' x s1       -- (y, s2) :: (b, s)
    q = state q'
```




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

state :: (s -> (a, s)) -> State s a
```



https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf