

COMP3141

Software System Design and Implementation

Effects and IO Monad Practice

Christine Rizkallah
CSE, UNSW (and Data61)
Term 2 2019

Recall: The IO Type

A **procedure** that performs some side effects, returning a result of type `a` is written as `IO a`.

Recall: The IO Type

A **procedure** that performs some side effects, returning a result of type `a` is written as `IO a`.

World interpretation

`IO a` is an abstract type. But we can think of it as a function:

$$\text{RealWorld} \rightarrow (\text{RealWorld}, a)$$

(that's how it's implemented in GHC)

Recall: The IO Type

A **procedure** that performs some side effects, returning a result of type `a` is written as `IO a`.

World interpretation

`IO a` is an abstract type. But we can think of it as a function:

$$\text{RealWorld} \rightarrow (\text{RealWorld}, a)$$

(that's how it's implemented in GHC)

```
(>>=) :: IO a -> (a -> IO b) -> IO b
```

```
pure  :: a -> IO a
```

```
getChar :: IO Char
```

```
readLine :: IO String
```

```
putStrLn :: String -> IO ()
```

QuickChecking Monads

QuickCheck lets us test IO (and ST) using this special **property monad** interface:

```
monadicIO :: PropertyM IO () -> Property
pre       :: Bool -> PropertyM IO ()
assert    :: Bool -> PropertyM IO ()
run       :: IO a -> PropertyM IO a
```

QuickChecking Monads

QuickCheck lets us test IO (and ST) using this special **property monad** interface:

```
monadicIO :: PropertyM IO () -> Property
pre       :: Bool -> PropertyM IO ()
assert    :: Bool -> PropertyM IO ()
run       :: IO a -> PropertyM IO a
```

Example (Testing hash)

Let's test that our IO password hash function works like GHC's non-effectful one.

QuickChecking Monads

QuickCheck lets us test IO (and ST) using this special **property monad** interface:

```
monadicIO :: PropertyM IO () -> Property
pre       :: Bool -> PropertyM IO ()
assert    :: Bool -> PropertyM IO ()
run       :: IO a -> PropertyM IO a
```

Example (Testing hash)

Let's test that our IO password hash function works like GHC's non-effectful one.

- This implementation is functionally correct but is it secure?

QuickChecking Monads

QuickCheck lets us test IO (and ST) using this special **property monad** interface:

```
monadicIO :: PropertyM IO () -> Property
pre       :: Bool -> PropertyM IO ()
assert    :: Bool -> PropertyM IO ()
run       :: IO a -> PropertyM IO a
```

Example (Testing hash)

Let's test that our IO password hash function works like GHC's non-effectful one.

- This implementation is functionally correct but is it secure?
- Does functional correctness imply security?

QuickChecking Monads

QuickCheck lets us test IO (and ST) using this special **property monad** interface:

```
monadicIO :: PropertyM IO () -> Property
pre       :: Bool -> PropertyM IO ()
assert    :: Bool -> PropertyM IO ()
run       :: IO a -> PropertyM IO a
```

Example (Testing hash)

Let's test that our IO password hash function works like GHC's non-effectful one.

- This implementation is functionally correct but is it secure?
- Does functional correctness imply security?
- Could hash still be identity? Is it a good hash function?

Recall: State Monads

```
newtype State s a = State (s -> (s, a))
```

Recall: State Monads

```
newtype State s a = State (s -> (s, a))
```

State Monad

```
get :: State s s  
put :: s -> State s ()  
modify :: (s -> s) -> State s ()
```

Recall: State Monads

```
newtype State s a = State (s -> (s, a))
```

State Monad

```
get  :: State s s  
put  :: s -> State s ()  
modify :: (s -> s) -> State s ()
```

Here we use a **monadic** interface to simplify the passing of our state around, so that we don't need to manually plumb data around.

Fibonacci Example

Fibonacci Example

Example (Testing Fibonacci)

Let's test that our stateful Fibonacci function works like the pure one.

Fibonacci Example

Example (Testing Fibonacci)

Let's test that our stateful Fibonacci function works like the pure one.

- Does the performance of the abstract model matter when testing?

Fibonacci Example

Example (Testing Fibonacci)

Let's test that our stateful Fibonacci function works like the pure one.

- Does the performance of the abstract model matter when testing?
- But shouldn't our abstract model be as abstract as possible?

Fibonacci Example

Example (Testing Fibonacci)

Let's test that our stateful Fibonacci function works like the pure one.

- Does the performance of the abstract model matter when testing?
- But shouldn't our abstract model be as abstract as possible?
- This is a cost that testing incurs as opposed to formal verification

Homework

- ➊ New exercise out, due Tuesday next week.
- ➋ Last week's quiz is due on Friday.
- ➌ This week's quiz is due the following Friday.
- ➍ **Note:** Assignment 2 released next week!