Haskell and Circuit Design

Alex Wang

Shanghai Jiao Tong University

March 14, 2012

CONTENT

1. What's Haskell
2. Type in Haskell
3. Function in Haskell
4. Type Class
5. Monad
6. Lava – build circuit with Haskell

1 What's Haskell

2 Type in Haskell

3 Function in Haskell

4 Type Class

5 Monad

6 Lava – build circuit with Haskell

What's Haskell

What's Haskell

- http://haskell.org

What's Haskell

- http://haskell.org
- Haskell is functional language. In particular, it is a

What's Haskell

- http://haskell.org
- Haskell is functional language. In particular, it is a
  - polymorphically
  - statically typed
  - lazy
  - purely functional language

What's Haskell

- http://haskell.org
- Haskell is functional language. In particular, it is a
  - polymorphically
  - statically typed
  - lazy
  - purely functional language
- Haskell is based on the lambda calculus.

1. What’s Haskell
2. Type in Haskell
3. Function in Haskell
4. Type Class
5. Monad
6. Lava – build circuit with Haskell

Type in Haskell

Type in Haskell

Every expression and function has a type.

Type in Haskell

Every expression and function has a type.

- Strong types
  - Unexpected arguments, expect integer but get bool
  - No casting

Type in Haskell

Every expression and function has a type.

- Strong types
  - Unexpected arguments, expect integer but get bool
  - No casting
- Static types
  - The compiler know the type at compile time
  - True && "False"

Type in Haskell

Every expression and function has a type.

- Strong types
  - Unexpected arguments, expect integer but get bool
  - No casting
- Static types
  - The compiler know the type at compile time
  - True && "False"
- Type inference
  - Automatically deduce the types of expressions

Type in Haskell

- Type inference
    - Automatically deduce the types of expressions

Type in Haskell

- Type inference
  - Automatically deduce the types of expressions
- Basic types
  - Char, Bool, Int, Integer, []...

Type in Haskell

- Type inference
  - Automatically deduce the types of expressions
- Basic types
  - Char, Bool, Int, Integer, []...
- Define a new type
  - data BookInfo = Book Int String [String] deriving (show)

1. What's Haskell
2. Type in Haskell
3. Function in Haskell
4. Type Class
5. Monad
6. Lava – build circuit with Haskell

Function in Haskell

Function in Haskell

- Basic funtions
  - lines
  - map
  - ...
  - odd $3- > $ True

Function in Haskell

- Basic funtions
  - lines
  - map
  - ...
  - odd $3- >$ True
- Type of function

```
:t {function_name}
```

Function in Haskell

- Define a funtion

```
upperCase :: String -> String
upperCase (x:xs) = toUpper x : upperCase xs
upperCase [ ]  [ ]
```

Function in Haskell

- Define a funtion

```
upperCase :: String -> String
upperCase (x:xs) = toUpper x : upperCase xs
upperCase [ ] [ ]
```

- Anonymous (lambda functions)

```
unsafeHead = \(x:_) -> x
```

1 What's Haskell

2 Type in Haskell

3 Function in Haskell

4 Type Class

5 Monad

6 Lava – build circuit with Haskell

Type Class

Type Class

What are type classes?

Type Class

What are type classes?

- Define a set of functions that can have different implementations depending on the type of data they are given.

Type Class

What are type classes?

- Define a set of functions that can have different implementations depending on the type of data they are given.
- Without type classes

```
stringEq :: [Char] -> [Char] -> Bool
intEq      ::    Int  -> Int -> Bool
```

Type Class

- Definition of typeclass

```
class  BasicEq a where
       isEqual  ::  a -> a -> Bool
```

```
class  Monad m where
       >>= :: Monad m => m a -> (a -> m b) -> m b
```

Type Class

- Definition of typeclass

```
class BasicEq a where
      isEqual :: a -> a -> Bool
```

```
class Monad m where
      >>= :: Monad m => m a -> (a -> m b) -> m b
```

- Instance of typeclass

```
instance BasicEq Bool where
         isEqual True True = True
         isEqual False False = True
         isEqual _ _ False
```

Type Class

- Typeclass inheritance

```
class Monad m => Circuit m where
      and2, or2 :: (Bit, Bit) -> m Bit
```

1. What's Haskell
2. Type in Haskell
3. Function in Haskell
4. Type Class
5. **Monad**
6. Lava – build circuit with Haskell

Monad

Monad

Why monad?

Monad

Why monad?

- Actions & Sequencing

```
main = do    putStrLn Greetings! What is your name?
             inpStr <- getLine
             putStrLn & Welcome to Haskell, ++ inpStr
        ++ !
```

```
main  = putStrLn Greetings! What is your name? >>
        getLine >>=
        (\inpStr -> putStrLn $ Welcome to Haskell, ++
        inpStr ++ !)
```

Monad

Monad type class

```
class  Monad m where
```

Monad

Monad type class

```
class Monad m where
```

- >>=

```
>>= :: m a -> (a -> m b) -> m b
print foo >>= \_ -> print bar
```

Monad

Monad type class

```
class Monad m where
```

- >>=

```
>>= :: m a -> (a -> m b) -> m b
print foo >>= \_ -> print bar
```

- return

```
return  ::  a -> m a
return 9
```

Monad

- >>

```
>>  ::   m a -> m b -> m b
print foo >> print bar
```

Monad

- >>

```
>>   ::   m a -> m b -> m b
print  foo >> print bar
```

- fail

```
fail   String  -> m a
fail   error
```

1. What's Haskell
2. Type in Haskell
3. Function in Haskell
4. Type Class
5. Monad
6. Lava – build circuit with Haskell

Lava – build circuit with Haskell

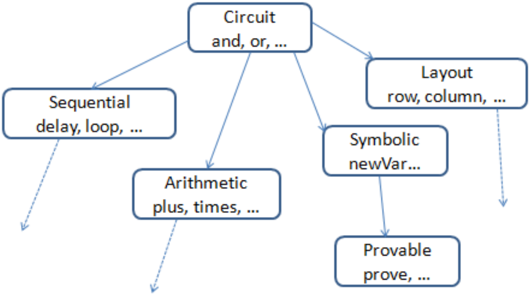
Lava – build circuit with Haskell

- Types
  - Bit (low,high)
  - NumSig
  - CmplxSig

Lava – build circuit with Haskell

- Types
  - Bit (low,high)
  - NumSig
  - CmplxSig

- Type class
  - Circuit
  - Sequential
  - Layout
  - Symbolic
  - Arithmetic
  - Provable



Lava – build circuit with Haskell

Lava – build circuit with Haskell

- Build in funtions

```
(>->) :: Circuit m => (a -> m b) -> (b -> m c) -> (a -> m c)
    compose :: Circuit m => [a -> m a] -> (a -> m a)
    one :: Circuit m => ([a] -> m [a]) -> ([a] -> m [a])
    two :: Circuit m => ([a] -> m [b]) -> ([a] -> m [b])
```

Thank you!

Q & A