Beautiful Types

by Mike Harris

Haskell

Identity

```
id ::
a -> a
```

Constant

```
const ::
a -> b -> a
```

What does this do?

```
(a -> b) -> [a] -> [b]
```

Map

```
map ::
(a -> b) -> [a] -> [b]
```

What does this do?

```
forall a b. (b -> a -> b) -> b -> [a] -> b
```

Fold

```
foldl ::
forall a b. (b -> a -> b) -> b -> [a] -> b
```

What does this do?

```
Int -> [a] -> [a]
```

Take

```
take ::
Int -> [a] -> [a]
```

What happens if Int < length [a]?

```
take ::
Int -> [a] -> [a]
```

[]

```
take _ []          =  []
```

What happens if Int < 0?

```haskell
take ::
Int -> [a] -> [a]
```

[]

```
take n _          | n <= 0 =  []
```

Take

```
take n _          | n <= 0 =  []
take _ []                 =  []
take n (x:xs)             =  x : take (n-1) xs
```

Idris

What does this do?

```
(n : Nat) -> (xs : List a) -> List a
```

Take

```
take :
(n : Nat) -> (xs : List a) -> List a
```

What happens if Int < length [a]?

```
take :
(n : Nat) -> (xs : List a) -> List a
```

```
                    []

take (S n) []       = []
```

Take

```
take Z       xs        = []
take (S n)   []        = []
take (S n)   (x::xs)   = x :: take n xs
```

What does this do?

```
(n : Nat) -> Vect (n + m) elem -> Vect n elem
```

Take

```
take :
(n : Nat) -> Vect (n + m) elem -> Vect n elem
```

Take

```
take Z       xs          = []
take (S k) (x :: xs) = x :: take k xs
```

What does this do?

```
(n : Nat) -> Vect (n + m) elem -> Vect m elem
```

Drop

```
drop :
(n : Nat) -> Vect (n + m) elem -> Vect m elem
```

Drop

```
drop Z       xs          = xs
drop (S k) (x :: xs) = drop k xs
```

Types that tells you what they are, are beautiful.

Thank you!

Mike Harris

@MikeMKH

https://github.com/MikeMKH/talks/tree/master/beautiful-types

S steps

- Learn more about Idris - http://docs.idris-lang.org/en/latest/tutorial/index.html
- Try Idris - https://tryidris.herokuapp.com/console
- Watch about Type Driven Development with Idris - https://www.youtube.com/watch?v=X36ye-1x_HQ

Haskell Source Code

- id - http://hackage.haskell.org/package/base-4.10.1.0/docs/src/GHC.Base.html#id
- const - http://hackage.haskell.org/package/base-4.10.1.0/docs/src/GHC.Base.html#const
- map - http://hackage.haskell.org/package/base-4.10.1.0/docs/src/GHC.Base.html#map
- foldl - http://hackage.haskell.org/package/base-4.10.1.0/docs/src/GHC.List.html#foldl
- take - http://hackage.haskell.org/package/base-4.10.1.0/docs/src/GHC.List.html#take

Idris Source Code

- List take - https://github.com/idris-lang/Idris-dev/blob/beb8e9cdb881f540094b4f457fd03d44af116103/libs/prelude/Prelude/List.idr#L191-L194
- Vect take - https://github.com/idris-lang/Idris-dev/blob/beb8e9cdb881f540094b4f457fd03d44af116103/libs/base/Data/Vect.idr#L104-L106
- Vect drop - https://github.com/idris-lang/Idris-dev/blob/beb8e9cdb881f540094b4f457fd03d44af116103/libs/base/Data/Vect.idr#L110-L112

Images

- Idris logo - Created by Heath Johns (https://twitter.com/edwinbrady/status/566261662303653888) and added to GitHub by Jan de Muijnck-Hughes, Public Domain, https://github.com/idris-lang/Idris-dev/blob/master/icons/text-x-idris.svg

- Haskell logo - Thought up by Darrin Thompson and produced by Jeff Wheeler - Thompson-Wheeler logo on the haskell wiki, Public Domain, https://commons.wikimedia.org/w/index.php?curid=8479507