Introduction to Dependent Types

Eagan Technology Unconference

Joseph Ching

September 22, 2015

Table of Contents

Table of Contents

Table of Contents

Quick Question

How many are familiar with this topic?

A Joke

This is not a `m-` tutorial.

A Joke

This is not a `m-` tutorial.
Nor is it a `lens` tutorial

A Joke

This is not a `m-` tutorial.
Nor is it a `lens` tutorial (aka the new new `m-` tutorial...

A Joke

This is not a `m-` tutorial.
Nor is it a `lens` tutorial (aka the new new `m-` tutorial...
...because `arrows` *were* the new `m-` tutorials).

About This Talk

`Agda`, `Idris`, `Coq` and co* have full support for dependent types.

About This Talk

`Agda`, `Idris`, `Coq` and co* have full support for dependent types. Because of that, it's harder to see the build up, so we won't be directly using them in this talk.

About This Talk

`Agda`, `Idris`, `Coq` and co* have full support for dependent types. Because of that, it's harder to see the build up, so we won't be directly using them in this talk.

Honestly though, it's because they're way over my head :(

(*) There was another mini joke here...

About This Talk

But we will be using `Haskell` though :)

About This Talk

But we will be using `Haskell` though :)
It's not truely dependent, but we can do more and more with each `language extension` that comes along.

Test

Syntax highlighting test reference, to be removed later.

```
-- Comment
data Maybe a = Nothing | Just a
               deriving (Show, Eq)

fmap :: Functor f => (a -> b) -> f a -> f b
map _ []     = []
map f (x:xs) = f x : map f xs

type family TF a :: *
type instance TF Int = Bool
```

Test

Couldn't quite yet get listing to work with overlay yet.

```
{- block comment -}
foo :: Bool -> Int -> String
foo False 0 = "Bad"
foo True  0 = "Questionable"
foo False n = "Fake"
foo True  n = "Read"
```

Test

Pausing within listing is ok?

```
{-# LANGUAGE KitchenSink #-}
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
```

Test

Pausing within listing is ok?

```
{-# LANGUAGE KitchenSink #-}
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith _ []     _      = []
```

Test

Pausing within listing is ok?

```
{-# LANGUAGE KitchenSink #-}
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith _ []     _      = []
zipWith _ _      []     = []
```

Test

Pausing within listing is ok?

```
{-# LANGUAGE KitchenSink #-}
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith _ []     _      = []
zipWith _ _      []     = []
zipWith f (x:xs) (y:ys) = f x y : zipWith f xs ys
```

Values and Types

Values has types.
Values are classified by types.

```
..., -1, 0, 1, 2, 3 :: Int

True, False :: Bool

'a', 'b', 'c' :: Char

"abc" :: String ~ [Char]
```

Questions?