Modularity and Backpack

Maciek Makowski (@mmakowski)

11th January 2015

MirageOS

TODO picture of unikernel

MirageOS

TODO picture of HTTP app with dependencies

Exhibit 1

MirageOS

> *The compiler heavily emphasises static type checking, and the resulting binaries are fast native code with no runtime type information and the module system is among the most powerful in a general-purpose programming language in terms of permitting flexible and safe code reuse and refactoring.*

– Technical Background of MirageOS

ML Modules

Structures

```
structure IntInteger =
  struct
    type integer = int
    val zero = 0
    val succ n = n + 1
    fun add a b = a + b
    fun mul a b = a * b
  end
```

ML Modules

Signatures

```
signature INTEGER =
  sig
    type integer
    val zero: integer
    val succ: integer -> integer
    fun add: integer -> integer -> integer
    fun mul: integer -> integer -> integer
  end
```

ML Modules

Functors

```
functor RationalFun(I: INTEGER) =
  struct
    type rational = I.integer * I.integer
    fun nom (n, _) = n
    fun denom (_, d) = d
    fun add (n1, d1) (n2, d2) =
      (I.add (I.mul n1 d2) (I.mul n2 d1),
       I.mul d1 d2)
  end

structure IntRational = RationalFun(IntInteger)
```

ML Modules

TODO: Mirage app diagram

Exhibit 2

tagstream-conduit

```
data Dec builder string = Dec
  { decToS      :: builder -> string
  , decBreak    :: (Char -> Bool) -> string ->
                   (string, string)
  , decBuilder  :: string -> builder
  , decDrop     :: Int -> string -> string
  , decEntity   :: string -> Maybe string
  , decUncons   :: string -> Maybe (Char, string)
  }
```

Exhibit 3

tagsoup

```
class (Typeable a, Eq a) => StringLike a where
    empty      :: a
    cons       :: Char -> a -> a
    uncons     :: a -> Maybe (Char, a)
    toString   :: a -> String
    fromString :: String -> a
    -- [...]
```

Backpack

Signatures

```
module IntegerSig where

data Integer

zero :: Integer
succ :: Integer
add  :: Integer -> Integer -> Integer
mul  :: Integer -> Integer -> Integer
```

Backpack

Mixins

```
module Rational where

import qualified IntegerSig as I

data Rational = R I.Integer I.Integer

nom (R n _) = n
denom (R _ d) = d
add (R n1 d1) (R n2 d2) =
  R (I.add (I.mul n1 d2) (I.mul n2 d1))
    (I.mul d1 d2)
```