

1 Writer

```

1 let abstract_writer =  $\lambda\phi \subseteq \{\text{File.write}, \text{Socket.write}\}$ .
2    $\lambda f: \text{Unit} \rightarrow_{\phi} \text{Unit}$ .
3     f unit
4
5 let file_writer =  $\lambda x: \{\text{File}\}$ .
6    $\lambda y: \text{Unit}$ .
7     x.write
8
9 in abstract_writer  $\{\text{File}\}$  (file_writer File)

```

2 Map Function

Pseudo-Wyvern

```

1 def map(f:  $A \rightarrow_{\phi} B$ , l: List[A]): List[B] with  $\phi =$ 
2   if isnil l then []
3   else cons (f (head l)) (map (tail l f))

```

λ -Calculus

```

1 map =  $\lambda\phi. \lambda A. \lambda B.$ 
2    $\lambda f: A \rightarrow_{\phi} B.$ 
3     (fix ( $\lambda \text{map}: \text{List}[A] \rightarrow \text{List}[B]$ )).
4        $\lambda l: \text{List}[A].$ 
5         if isnil l then []
6         else cons (f (head l)) (map (tail l f)))

```

Typing

- This has the type: $\forall\phi.\forall A.\forall B.(A \rightarrow_{\phi} B) \rightarrow_{\emptyset} \text{List}[A] \rightarrow_{\phi} \text{List}[B]$ with \emptyset .
- $\text{map } \emptyset$ is a pure version of map.
- $\text{map } \{\text{File}.*\}$ is a version of map which can perform operations on File.

3 Dependency Injection

Pseudo-Wyvern

An HTTPServer module provides a single `init` method which returns a `Server` that responds to HTTP requests on the supplied socket.

```

1 module HTTPServer
2
3 def init(out: A <: {File, Socket}): Str  $\rightarrow_{A.write} \text{Unit}$  with  $\emptyset =$ 
4    $\lambda \text{msg}: \text{Str}.$ 
5     if (msg == ‘POST’) then out.write(‘post response’)
6     else if (msg == ‘GET’) then out.write(‘get response’)
7     else out.write(‘client error 400’)

```

The main module calls `HTTPServer.init` with the `Socket` it should be writing to.

```

1 module Main
2 require HTTPServer, Socket
3
4 def main(): Unit =
5   HTTPServer.init(Socket) ‘GET /index.html’

```

The testing module calls `HTTPServer.init` with a `LogFile`, perhaps so the responses of the server can be tested offline.

```

1 module Testing
2   require HTTPServer, LogFile
3
4   def testSocket(): =
5     HTTPServer.init(LogFile) 'GET /index.html'

```

λ -Calculus

The HTTPServer module:

```

1 MakeHTTPServer =  $\lambda x$ : Unit.
2    $\lambda \phi \subseteq \{\text{LogFile.write}, \text{Socket.write}\}.$ 
3      $\lambda f$ : Str  $\rightarrow_{\phi}$  Unit.
4      $\lambda \text{msg}$ : Str.
5       f msg

```

The Main module:

```

1 MakeMain =  $\lambda \text{hs}$ : HTTPServer.  $\lambda \text{sock}$ : {Socket}.
2    $\lambda x$ : Unit.
3     let socketWriter = ( $\lambda s$ : {Socket}.  $\lambda x$ : Unit. s.write) sock in
4     let theServer = hs {Socket.write} socketWriter in
5     theServer 'GET/index.html'

```

The Testing module:

```

1 MakeTest =  $\lambda \text{hs}$ : HTTPServer.  $\lambda lf$ : {LogFile}.
2    $\lambda x$ : Unit.
3     let logFileWriter = ( $\lambda l$ : {LogFile}.  $\lambda x$ : Unit. l.write) lf in
4     let theServer = hs {LogFile.write} logFileWriter in
5     theServer 'GET/index.html'

```

A single, desugared program for production would be:

```

1 let MakeHTTPServer =  $\lambda x$ : Unit.
2    $\lambda \phi \subseteq \{\text{LogFile.write}, \text{Socket.write}\}.$ 
3      $\lambda f$ : Str  $\rightarrow_{\phi}$  Unit.
4      $\lambda \text{msg}$ : Str.
5       f msg
6
7 in let Run =  $\lambda \text{Socket}$ : {Socket}.
8   let HTTPServer = MakeHTTPServer unit in
9   let Main = MakeMain HTTPServer Socket in
10  Main unit
11
12 in Run Socket

```

A single, desugared program for testing would be:

```

1 let MakeHTTPServer =  $\lambda x$ : Unit.
2    $\lambda \phi \subseteq \{\text{LogFile.write}, \text{Socket.write}\}.$ 
3      $\lambda f$ : Str  $\rightarrow_{\phi}$  Unit.
4      $\lambda \text{msg}$ : Str.
5       f msg
6
7 in let Run =  $\lambda \text{LogFile}$ : {LogFile}.
8   let HTTPServer = MakeHTTPServer unit in
9   let Main = MakeMain HTTPServer LogFile in
10  Main unit
11
12 in Run LogFile

```

Note how the HTTPServer code is identical in the testing and production examples.

Typing

To type `MakeHTTPServer`:

1. By ε -APP,

$$\begin{array}{l} x : \text{Unit}, \text{Writer} <: \text{Str} \rightarrow_{\{\text{Lf.write}, \text{S.write}\}} \text{Unit}, f : \text{Writer}, \text{msg} : \text{Str} \\ \vdash f \text{ msg} : \text{Unit} \end{array}$$

Types

- `HTTPServer.init` has the type $\lambda A <: \{\text{File}, \text{Socket}\}. A \rightarrow_{\emptyset} \text{Str} \rightarrow_{A.\text{write}} \text{Unit}$