Extensible Effects
A Library for Managing Side-Effects

Justin Bailey
Presented by the Functional Programming Guild

August 1, 2014

Side Effects

What does this function do?

```
// A Java method
public static int add(int a, int b) {
  ...
}
```

Side Effects

What does this function do?

```
// A Java method
public static int add(int a, int b) {
  System.exit(0);
}
```

EXPLICIT Side Effects

Use *IO* to represent effects:

$$getChar :: Handle \to IO\ Char$$
$$openFile :: FilePath \to IO\ Handle$$
$$putChar :: Handle \to Char \to IO\ ()$$

EXPLICIT Side Effects

What does this function do?

$$add :: Int \to Int \to Int$$
$$add\ a\ b = \ldots$$

Hint: $exitFailure :: IO\ ()$

EXPLICIT Side Effects

Possible?

$$
\begin{aligned}
&\mathit{add} :: \mathit{Int} \to \mathit{Int} \to \mathit{Int}\\
&\mathit{add}\ a\ b = \mathit{exitWith}\ (\mathit{ExitFailure}\ 0)
\end{aligned}
$$

Hint: $\mathit{exitFailure} :: \mathit{IO}\ ()$

EXPLICIT Side Effects

$IO\ () \not\equiv Int$

$$\begin{aligned}&add :: Int \rightarrow Int \rightarrow Int\\ &add\ a\ b = exitWith\ (ExitFailure\ 0) :: IO\ () \quad \text{-- no}\end{aligned}$$

EXPLICIT Side Effects

One *correct* possibility:

$add :: Int \to Int \to Int$
$add\ a\ b = a + b$ -- Int

EXPLICIT Side Effects

Types don't solve *everything* ...

$$add :: Int \to Int \to Int$$
$$add\ a\ b = add\ a\ b$$

IO: The “Sin” Bin

$$
\begin{aligned}
&\mathit{putStrLn} :: \mathit{String} \to \mathit{IO}\ ()\\
&\mathit{openFile} :: \mathit{FilePath} \to \mathit{FileMode} \to \mathit{IO}\ \mathit{Handle}\\
&\mathit{exitFailure} :: \mathit{IO}\ ()\\
&\mathit{exitSuccess} :: \mathit{IO}\ ()\\
&\mathit{forkIO} :: \mathit{IO}\ () \to \mathit{IO}\ \mathit{ThreadId}
\end{aligned}
$$

IO: The “Sin” Bin

What does this function do?

$$\begin{aligned}&add\ a\ b :: Int \to Int \to IO\ Int\\ &add\ a\ b = \ldots\end{aligned}$$

IO: The "Sin" Bin

Valid:

$$
\begin{aligned}
&add\ a\ b :: Int \rightarrow Int \rightarrow IO\ Int\\
&add\ a\ b = return\ (a + b)
\end{aligned}
$$

IO: The “Sin” Bin

Valid:

```
add a b :: Int → Int → IO Int
add a b = do
  exitFailure
  return 0
```

IO: The "Sin" Bin

Valid:

$$
\begin{aligned}
&\mathit{add}\ a\ b :: \mathit{Int} \to \mathit{Int} \to \mathit{IO}\ \mathit{Int}\\
&\mathit{add}\ a\ b = \mathbf{do}\\
&\quad \mathit{deleteAllFiles}\\
&\quad \mathit{return}\ (a + b)
\end{aligned}
$$

Break Effects into Smaller Types

$$\begin{aligned}&openFile :: FilePath \rightarrow FileMode \rightarrow OpenFile\ Handle\\&readAll :: Handle \rightarrow ReadFile\ String\\&writeAll :: Handle \rightarrow String \rightarrow WriteFile\ ()\end{aligned}$$

Break Effects into Smaller Types

But how do you combine effects?

```
cp :: FilePath → FilePath → ???
cp src dest = do
  s ← openFile src      -- OpenFile Handle
  contents ← readAll s     -- ReadFile String
  d ← openFile dest     -- OpenFile Handle
  writeAll d contents     -- WriteFile ()
```

Break Effects into Smaller Types

But how do you combine effects?

```
cp :: FilePath → FilePath → ???
cp src dest = do
  s ← openFile src :: OpenFile Handle
  contents ← readAll s :: ReadFile String
  d ← openFile dest :: OpenFile Handle
  writeAll d contents :: WriteFile ()
```

Extensible Effects

Make effects a *list*:

$$
\begin{aligned}
&\mathit{openFile} :: \mathit{FilePath} \to \mathit{FileMode}\\
&\quad \to \mathit{Eff}\ (\mathit{OpenFile} :> \mathit{effects})\ \mathit{Handle}\\
&\mathit{readAll} :: \mathit{Handle}\\
&\quad \to \mathit{Eff}\ (\mathit{ReadFile} :> \mathit{effects})\ \mathit{String}\\
&\mathit{writeAll} :: \mathit{Handle} \to \mathit{String}\\
&\quad \to \mathit{Eff}\ (\mathit{WriteFile} :> \mathit{effects})\ ()
\end{aligned}
$$

Extensible Effects

Make effects a *list*:

$$
\begin{aligned}
&\mathit{openFile} :: \mathit{FilePath} \rightarrow \mathit{FileMode}\\
&\quad \rightarrow \mathit{Eff}\ (\mathit{OpenFile} :> \mathit{effects})\ \mathit{Handle}\\
&\mathit{readAll} :: \mathit{Handle}\\
&\quad \rightarrow \mathit{Eff}\ (\mathit{ReadFile} :> \mathit{effects})\ \mathit{String}\\
&\mathit{writeAll} :: \mathit{Handle} \rightarrow \mathit{String}\\
&\quad \rightarrow \mathit{Eff}\ (\mathit{WriteFile} :> \mathit{effects})\ ()
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{cp} :: \mathit{FilePath} \rightarrow \mathit{FilePath}\\
&\quad \rightarrow \mathit{Eff}\ (\mathit{OpenFile} :> \mathit{ReadFile} :> \mathit{WriteFile} :> \mathit{effects})\\
&\quad \rightarrow \mathit{Eff}\ \mathit{effects}\ ()\\
&\mathit{cp}\ \mathit{src}\ \mathit{dest} = ..\quad \text{-- almost the same code}
\end{aligned}
$$

Example: The *ReadableFile* Effect

Ideally:

$$\begin{aligned}&\mathit{runReadable} :: \mathit{FilePath} \to \mathit{Eff}\ (\mathit{ReadableFile} :> r) \to \mathit{Eff}\ r\ \mathit{result}\\&\mathit{getReadableHandle} :: (\mathit{Member}\ \mathit{ReadableFile}\ r) \Rightarrow \mathit{Eff}\ r\ \mathit{Handle}\end{aligned}$$

Example: The *ReadableFile* Effect

Actually:

$$
\begin{array}{l}
\mathit{getReadableHandle} :: (\mathit{Member}\ \mathit{ReadableFile}\ r) \Rightarrow \mathit{Eff}\ r\ \mathit{Handle} \\
\mathit{runReadableFile} :: (\mathit{SetMember}\ \mathit{Lift}\ (\mathit{Lift}\ \mathit{IO})\ r) \Rightarrow \mathit{FilePath} \\
\quad \rightarrow \mathit{Eff}\ (\mathit{ReadableFile} :> r)\ \mathit{result} \\
\quad \rightarrow \mathit{Eff}\ r\ \mathit{result}
\end{array}
$$

Reading a File

Primitives for reading files:

$$
\begin{aligned}
&\mathit{readChar} :: (\mathit{Member}\ \mathit{ReadFile}\ r) \Rightarrow \mathit{Eff}\ \ r\ \ \mathit{Char} \\
&\mathit{atEOF} :: (\mathit{Member}\ \mathit{ReadFile}\ r) \Rightarrow \mathit{Eff}\ \ r\ \ \mathit{Bool}
\end{aligned}
$$

Reading a File

ReadFile requires *ReadableFile*:

$$\begin{aligned}
&runReadFile :: (SetMember\ Lift\ (Lift\ IO)\ r, Member\ ReadableFile\ r)\\
&\quad \Rightarrow Eff\ (ReadFile :> r)\ result\\
&\quad \rightarrow Eff\ r\ result
\end{aligned}$$

Writing a File

WritableFile and *WriteFile*:

$$\begin{aligned}
&\mathit{getWriteableHandle} :: (\mathit{Member}\ \mathit{WriteableFile}\ r) \Rightarrow \mathit{Eff}\ \ r\ \mathit{Handle}\\
&\mathit{runWriteableFile} :: (\mathit{SetMember}\ \mathit{Lift}\ (\mathit{Lift}\ \mathit{IO})\ r) \Rightarrow \mathit{FilePath}\\
&\quad \rightarrow \mathit{Eff}\ \ (\mathit{WriteableFile} :> r)\ \mathit{result}\\
&\quad \rightarrow \mathit{Eff}\ \ r\ \mathit{result}\\
&\mathit{writeChar} :: (\mathit{Member}\ \mathit{WriteFile}\ r) \Rightarrow \mathit{Char} \rightarrow \mathit{Eff}\ \ r\ ()
\end{aligned}$$

Writing a File

WriteFile requires *WriteableFile*:

$$
\begin{array}{l}
\mathit{runWriteFile} :: (\mathit{SetMember}\ \mathit{Lift}\ (\mathit{Lift}\ \mathit{IO})\ r, \mathit{Member}\ \mathit{WriteableFile}\ r \\
\quad \Rightarrow \mathit{Eff}\ (\mathit{WriteFile} :> r)\ \mathit{result} \\
\quad \rightarrow \mathit{Eff}\ r\ \mathit{result}
\end{array}
$$

Impossible Programs

What does this program do?

```
cp_r src dst = runReadableFile src $ runReadFile $ do
  s ← getReadableHandle
  d ← getWriteableHandle
  contents ← readAll s
  writeAll d contents
```

Impossible Programs

Compilation error:

```
*Main> cp_r "Setup.hs" "foo"

<interactive>:30:1:
   No instance for (Member (* -> *) * WriteFile r0)
     arising from a use of ‘cp_r’
   ...
```

Copying a File (Safely)

```
cp src dest = runLift $ runWriteableFile dest $ runWriteFile $
  runReadableFile src $ runReadFile $ do
    h ← readChar
      writeChar h
```