Programming with Effect Handlers in Links

PLInG, Autumn 2015, University of Edinburgh

Daniel Hillerström
daniel.hillerstrom@ed.ac.uk
http://homepages.inf.ed.ac.uk/s1467124/

November 3, 2015

School of Informatics
The University of Edinburgh

WARNING

This talk may contain traces of jargon such as *monads*, *effects*, *algebras* and *handlers*. However the following code examples in this talk may be performed at home.

What this talk is about

Handlers for algebraic effects provide a compelling alternative to monads as a basis for effectful programming.

- **Key idea:** Separate effect signatures from their implementation.
- **"The effect":** High-degree of modularity.

Definitions will follow later. . .

PART 1: Effectively, it’s a problem

Programs are inherently effectful

Programs may...

- ... halt prematurely
- ... diverge
- ... be stateful (e.g. modify a global state)
- ... communicate via a network
- ... print to standard out

A pure[1] program is not much fun.

[1] By pure we mean a program that has no effects.

Fundamental different approaches to effects

Imperative Repeatedly performs implicit effects on shared global state.

Functional Encapsulates effects in a computational context.

Fundamental different approaches to effects

Imperative Repeatedly performs implicit effects on shared global state.

Functional Encapsulates effects in a computational context.

This talk is oriented around *functional* programming with effects.

Effectful computations (I)

$a \to b$	Mathematical pure function
$a \to b$	C/C++ (impure) function
$a \to b$	ML (impure) function

Let's be explicit about effects

Effect annotation

An effect annotation gives a static description of the potential run-time behaviour of a computation.

Benefits

- Serves as documentation (clarity)
- Compiler can apply specific optimisations
- Possible to reason more precisely about programs

Enter the Monad

"Shall I be pure or impure?"

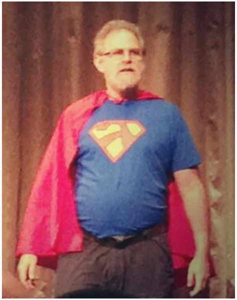


Figure 1: Philip Wadler aka. Lambda Man

- The Essence of Functional Programming [1]
- The Marriage of Effects and Monads [2]

Effectful computations (II)

$a \to b$	Mathematical pure function
$a \to b$	C/C++ (impure) function
$a \to b$	ML (impure) function
$a \to m\,b$	Haskell impure function

Effectful computations (II)

$a \to b$	Mathematical pure function
$a \to b$	C/C++ (impure) function
$a \to b$	ML (impure) function
$a \to m\,b$	Haskell impure function

m can be considered an effect annotation

Monads

Definition

A monad is a triple $(m, return, bind)$ where

- m is a type constructor
- $return : a \to m\, a$
- $bind : m\, a \to (a \to m\, b) \to m\, b$

Great! Many monads, many effects

A couple of monads and their "effect interpretation"

IO a	May perform I/O, returns a
Reader r a	May read from r, returns a
Writer w a	May write to w, returns a
State s a	May read/write some state s, returns a
Maybe a	May fail, returns a on success

Effectful computations (III)

$a \to b$	Mathematical pure function
$a \to b$	C/C++ (impure) function
$a \to b$	ML (impure) function
$a \to m_1\ m_2\ b$	Haskell impure function
$\not\simeq$	
$a \to m_2\ m_1\ b$	

IO Monad is equivalent to a calzone pizza

Figure 2: “There’s only meat sauce inside”, they said, but you can’t really be sure.

The importance of effect ordering [3]

Two signatures[2]:

- $A \stackrel{\text{def}}{=}$ `WriterT String Maybe`
    - Returns nothing on failure
- $B \stackrel{\text{def}}{=}$ `MaybeT (Writer String)`
    - Returns a pair on failure

[2] Elided details about Monad Transformers

PART 2: Exit the Monad, Enter the Handler

Algebraic effects

Figure 3: Gordon Plotkin

Figure 4: John Power

Algebraic effects and computations

Definition

Algebraic effect An algebraic effect is a collection of abstract operations, e.g. $\{Op_i : a_i \to b_i\}$

Definition

Abstract computation An abstract computation is composed from abstract operations. Computations have type

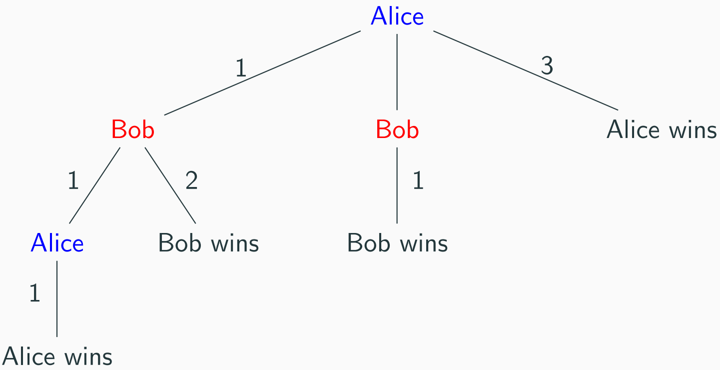
Nim: A game with sticks

Set-up

- Two players: Alice and Bob; Alice always starts.
- One heap of n sticks.
- Turn-based. Each player take between 1-3 sticks.
- The one, who takes the last stick, wins.

We'll demonstrate how to encode strategic behaviour, compute game data, and cheat using handlers.

Game tree generated by `mtGen` with $n = 3$



References

- Philip Wadler.
  The essence of functional programming.
  In *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '92, pages 1–14, New York, NY, USA, 1992. ACM.
- Philip Wadler and Peter Thiemann.
  The marriage of effects and monads.
  ACM Trans. Comput. Log., 4(1):1–32, 2003.
- Bryan O'Sullivan, John Goerzen, and Don Stewart.
  Real World Haskell.
  O'Reilly Media, Inc., 1st edition, 2008.
- Gordon D. Plotkin and Matija Pretnar.
  Handling algebraic effects.
  Logical Methods in Computer Science, 9(4), 2013.
- Philip Wadler, Sam Lindley, Garrett Morris, et al.
  Links: Linking theory to practice for the web.