Effect Handlers

A New Approach to Computational Effects

Maciej Piróg

University of Wrocław, Poland

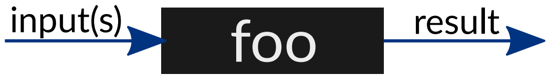
Lambda Days 2020

This talk

- Effect handlers: what and why?
- Live coding in our new experimental implementation

Why do we ❤ FP?

We compose our programs out of



all the way – from the tiniest bits to the big components.

Why do we ❤ FP?

We compose our programs out of

$$\begin{tikzcd}
{} \arrow[r, "\text{input(s)}"] & \boxed{\text{foo}} \arrow[r, "\text{result}"] & {}
\end{tikzcd}$$

all the way – from the tiniest bits to the big components.

Well... Yeah... Hmm... But...

Computational effects

- ♥ Input/output
- ♥ Exceptions
- ♥ Mutable state
- ♥ Backtracking
- ♥ Logging
- ♥ Concurrency
- ♥ Memoisation
- ♥ Control (`call/cc`, ...)
- ♥ Random value generation
- ♥ Fresh identifier generation
- ♥ ...

God-given effects

(as in most CBV languages, like OCaml, Scheme, F#, Erlang)

Good things:

★ Out of the box

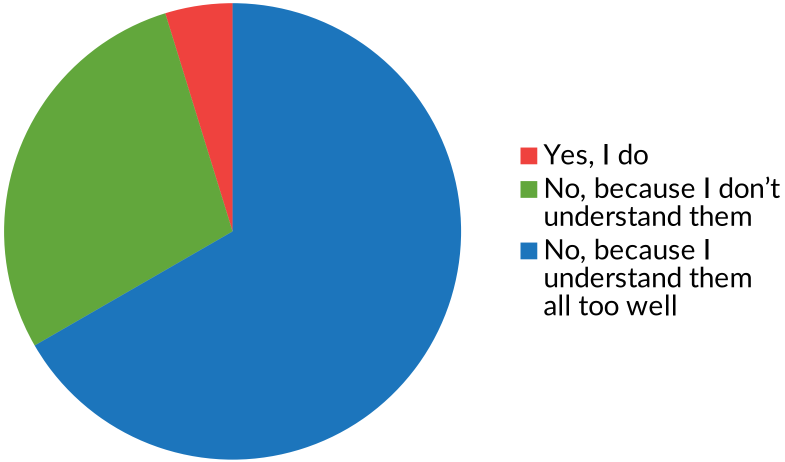
Bad things:

🚫 Only a predefined set of effects (backtracking search?)

🚫 Not tracked in the type system at all ($\mathit{Unit} \Rightarrow \mathit{Unit}$)

🚫 Fixed interaction between effects (transactional state?)

Do you enjoy programming with monads?



data source: No source, it's a joke (but is it?)

Monads

(as in Haskell)

Good things:

★ User-defined, fit-for-purpose effects

★ Effects tracked in types

Bad things:

🚫 Monadic (= kind-of imperative) style of programming

🚫 Modularity issues (transformer stack!)

Effect Handlers

(as in Eff, Frank, Koka, Links, Helium)

✦ Not that new on the theoretical side (Plotkin, Power, 2000s...)

Good things:

★ User-defined, fit-for-purpose effects

★ Effects tracked in types

★ Direct style of programming (refactoring!)

★ Easy custom interaction between effects

Bad things:

🚫 Still rather experimental as a programming construct

Operations and handlers

Effect signatures:

A bunch of (typed) operations, e.g.,

- $\mathit{throw} : \mathit{Unit} \Rightarrow a$ for exceptions
- $\mathit{put} : S \Rightarrow \mathit{Unit}$ and $\mathit{get} : \mathit{Unit} \Rightarrow S$ for state
- $\mathit{flip} : \mathit{Unit} \Rightarrow \mathit{Bool}$ for nondeterminism

Handlers:

Define how to proceed when an operation is encountered

- E.g., when *throw* is encountered, discard the entire computation within the handler, and replace it with a default value.

The Helium language

Homepage & sources:

```
https://bitbucket.org/pl-uwr/helium
```

Docker:

```
docker run -it pluwr/helium repl
```

Implements some theory from:
Binders by Day, Labels by Night: Effect Instances via Lexically Scoped Handlers
by D. Biernacki, M. Piróg, P. Polesiuk, and F. Sieczkowski (POPL 2020)

The Helium language

- ★ Effect handlers
- ★ Effect instances via lexical scoping
- ★ Advanced type-and-effect system
- ★ Effect polymorphism (without row types!)
- ★ Effect abstraction
- ★ Strong OneML-style module system

🚫 Reserch-level software (poor docs, hardly any tooling or libraring)

Example A

...in which the handler takes control over the situation

Example B

...in which the handler resumes the computation

Example C

...in which the handler resumes the computation many times

Thank you!

`https://bitbucket.org/pl-uwr/helium`

```
docker run -it pluwr/helium repl
```