Effect handler oriented programming

Sam Lindley

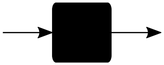
The University of Edinburgh

NTU Singapore, November 2024

What is an effect?

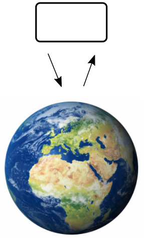
Effects

Programs as black boxes (Church-Turing model)?



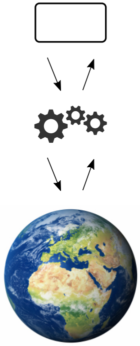
Effects

Programs must interact with their environment



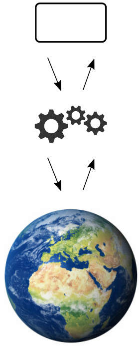
Effects

Programs must interact with their environment



Effects

Programs must interact with their environment

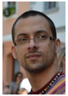


Effects are pervasive

- input/output
  user interaction
- concurrency
  web applications
- distribution
  cloud computing
- exceptions
  fault tolerance
- choice
  backtracking search

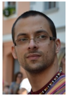
Typically ad hoc and hard-wired

Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009 (and ETAPS 2022 test of time award)

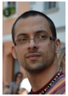
Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009 (and ETAPS 2022 test of time award)

Monads $\longrightarrow$ Algebraic Effects $\longrightarrow$ Effect Handlers

Effect handlers

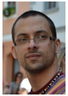
 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009 (and ETAPS 2022 test of time award)

Monads $\longrightarrow$ Algebraic Effects $\longrightarrow$ Effect Handlers

Composable and **customisable** user-defined interpretation of effects in general

Effect handlers

 Gordon Plotkin  Matija Pretnar

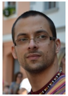
Handlers of algebraic effects, ESOP 2009 (and ETAPS 2022 test of time award)

Monads $\longrightarrow$ Algebraic Effects $\longrightarrow$ Effect Handlers

Composable and **customisable** user-defined interpretation of effects in general

Give programmer direct access to **context**
(c.f. resumable exceptions, monads, delimited control)

Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009 (and ETAPS 2022 test of time award)

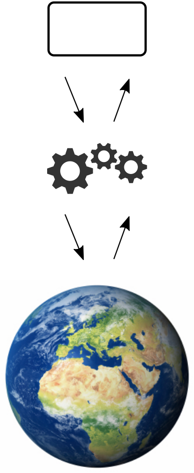
Monads $\longrightarrow$ Algebraic Effects $\longrightarrow$ Effect Handlers

Composable and **customisable** user-defined interpretation of effects in general

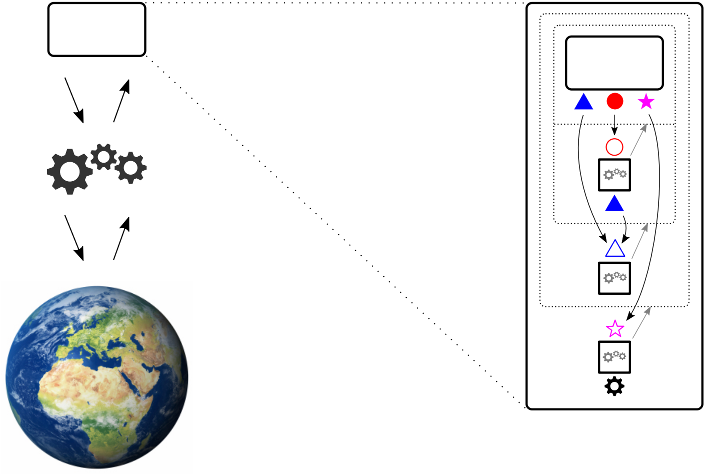
Give programmer direct access to **context**
(c.f. resumable exceptions, monads, delimited control)

Effect handlers in practice:
OCaml 5, GitHub (Semantic), Meta (React), Uber (Pyro), Wasm (WasmFX), ...

Effect handlers as composable user-defined operating systems



Effect handlers as composable user-defined operating systems



Effect handlers for operating systems

EIO — effects-based direct-style concurrent I/O stack for OCaml

`https://github.com/ocaml-multicore/eio`

Composing UNIX with effect handlers

Foundations for programming and implementing effect handlers, Chapter 2

Daniel Hillerström, PhD thesis, The University of Edinburgh, 2022

`https://www.dhil.net/research/papers/thesis.pdf`

Example 1: choice and failure

Effect signature

$$\{\mathsf{choose} : 1 \twoheadrightarrow \mathrm{Bool}, \quad \mathsf{fail} : a.1 \twoheadrightarrow a\}$$

Example 1: choice and failure

Effect signature

$$\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}, \quad \mathsf{fail} : a.1 \twoheadrightarrow a\}$$

Coin tossing

$$\begin{aligned}
&\mathsf{toss} : 1 \to \mathsf{Toss}!(E \uplus \{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\}) \\
&\mathsf{toss}\,() = \textbf{if } \mathsf{choose}\,() \textbf{ then } \mathsf{Heads} \textbf{ else } \mathsf{Tails}
\end{aligned}$$

Example 1: choice and failure

Effect signature

$$\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}, \quad \mathsf{fail} : a.1 \twoheadrightarrow a\}$$

Coin tossing

$$
\begin{aligned}
&\mathsf{toss} : 1 \to \mathsf{Toss}!(E \uplus \{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\}) \\
&\mathsf{toss}\,() = \mathbf{if}\ \mathsf{choose}\,()\ \mathbf{then}\ \mathsf{Heads}\ \mathbf{else}\ \mathsf{Tails}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{drunkToss} : 1 \to \mathsf{Toss}!(E \uplus \{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}, \quad \mathsf{fail} : a.1 \twoheadrightarrow a\}) \\
&\mathsf{drunkToss}\,() = \mathbf{if}\ \mathsf{choose}\,()\ \mathbf{then} \\
&\qquad\qquad\qquad \mathbf{if}\ \mathsf{choose}\,()\ \mathbf{then}\ \mathsf{Heads}\ \mathbf{else}\ \mathsf{Tails} \\
&\qquad\qquad\quad\ \mathbf{else} \\
&\qquad\qquad\qquad \mathsf{fail}\,()
\end{aligned}
$$

Example 1: choice and failure

Effect signature

$$\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}, \quad \mathsf{fail} : a.1 \twoheadrightarrow a\}$$

Coin tossing

$$
\begin{aligned}
&\mathsf{toss} : 1 \to \mathsf{Toss}!(E \uplus \{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\}) \\
&\mathsf{toss}\,() = \mathbf{if}\ \mathsf{choose}\,()\ \mathbf{then}\ \mathsf{Heads}\ \mathbf{else}\ \mathsf{Tails}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{drunkToss} : 1 \to \mathsf{Toss}!(E \uplus \{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}, \quad \mathsf{fail} : a.1 \twoheadrightarrow a\}) \\
&\mathsf{drunkToss}\,() = \mathbf{if}\ \mathsf{choose}\,()\ \mathbf{then} \\
&\qquad\qquad\qquad\quad \mathbf{if}\ \mathsf{choose}\,()\ \mathbf{then}\ \mathsf{Heads}\ \mathbf{else}\ \mathsf{Tails} \\
&\qquad\qquad\qquad \mathbf{else} \\
&\qquad\qquad\qquad\quad \mathsf{fail}\,()
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{drunkTosses} : \mathsf{Nat} \to \mathsf{List}\ \mathsf{Toss}!(E \uplus \{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}, \quad \mathsf{fail} : a.1 \twoheadrightarrow a\}) \\
&\mathsf{drunkTosses}\, n = \mathbf{if}\ n = 0\ \mathbf{then}\ [] \\
&\qquad\qquad\qquad\quad \mathbf{else}\ \mathsf{drunkToss}\,() :: \mathsf{drunkTosses}\,(n-1)
\end{aligned}
$$

Example 1: choice and failure

Handlers

$$\text{maybeFail} : A!(E \uplus \{\text{fail} : a.1 \twoheadrightarrow a\}) \Rightarrow \text{Maybe}\ A!E$$

$$\text{maybeFail} = \quad \text{— exception handler}$$

$$\quad \textbf{return}\, x \mapsto \text{Just}\, x$$

$$\quad \langle \text{fail}\,()\rangle \mapsto \text{Nothing}$$

Example 1: choice and failure

Handlers

$$\mathsf{maybeFail} : A!(E \uplus \{\mathsf{fail} : a.1 \twoheadrightarrow a\}) \Rightarrow \mathsf{Maybe}\ A!E$$

$$\mathsf{maybeFail} = \quad \text{— exception handler}$$

$$\mathbf{return}\, x \mapsto \mathsf{Just}\, x$$

$$\langle \mathsf{fail}\, ()\rangle \mapsto \mathsf{Nothing}$$

$$\mathbf{handle}\ 42\ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Just}\, 42$$

$$\mathbf{handle}\ \mathsf{fail}\, ()\ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Nothing}$$

Example 1: choice and failure

Handlers

$$\mathsf{maybeFail} : A!(E \uplus \{\mathsf{fail} : a.1 \twoheadrightarrow a\}) \Rightarrow \mathsf{Maybe}\ A!E$$

$\mathsf{maybeFail} =$ — exception handler

$\quad \textbf{return}\ x \mapsto \mathsf{Just}\ x$

$\quad \langle \mathsf{fail}\ ()\rangle \mapsto \mathsf{Nothing}$

$\textbf{handle}\ 42\ \textbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Just}\ 42$

$\textbf{handle}\ \mathsf{fail}\ ()\ \textbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Nothing}$

$$\mathsf{trueChoice} : A!(E \uplus \{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\}) \Rightarrow A!E$$

$\mathsf{trueChoice} =$ — linear handler

$\quad \textbf{return}\ x \mapsto x$

$\quad \langle \mathsf{choose}\ () \to r\rangle \mapsto r\ \mathsf{true}$

Example 1: choice and failure

Handlers

$\mathrm{maybeFail} : A!(E \uplus \{\mathrm{fail} : a.1 \twoheadrightarrow a\}) \Rightarrow \mathrm{Maybe}\ A!E$

$\mathrm{maybeFail} =$ — exception handler

$\quad \mathbf{return}\, x \mapsto \mathrm{Just}\, x$

$\quad \langle \mathrm{fail}\, () \rangle \mapsto \mathrm{Nothing}$

$\mathbf{handle}\ 42\ \mathbf{with}\ \mathrm{maybeFail} \Longrightarrow \mathrm{Just}\, 42$

$\mathbf{handle}\ \mathrm{fail}\, ()\ \mathbf{with}\ \mathrm{maybeFail} \Longrightarrow \mathrm{Nothing}$

$\mathrm{trueChoice} : A!(E \uplus \{\mathrm{choose} : 1 \twoheadrightarrow \mathrm{Bool}\}) \Rightarrow A!E$

$\mathrm{trueChoice} =$ — linear handler

$\quad \mathbf{return}\, x \mapsto x$

$\quad \langle \mathrm{choose}\, () \to r \rangle \mapsto r\, \mathrm{true}$

$\mathbf{handle}\ 42\ \mathbf{with}\ \mathrm{trueChoice} \Longrightarrow 42$

$\mathbf{handle}\ \mathrm{toss}\, ()\ \mathbf{with}\ \mathrm{trueChoice} \Longrightarrow \mathrm{Heads}$

Example 1: choice and failure

Handlers

$\mathsf{maybeFail} : A!(E \uplus \{\mathsf{fail} : a.1 \twoheadrightarrow a\}) \Rightarrow \mathsf{Maybe}\ A!E$

$\mathsf{maybeFail} =$ — exception handler

$\quad \mathbf{return}\ x \mapsto \mathsf{Just}\ x$

$\quad \langle \mathsf{fail}\ ()\rangle \mapsto \mathsf{Nothing}$

$\mathbf{handle}\ 42\ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Just}\ 42$

$\mathbf{handle}\ \mathsf{fail}\ ()\ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Nothing}$

$\mathsf{trueChoice} : A!(E \uplus \{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\}) \Rightarrow A!E$

$\mathsf{trueChoice} =$ — linear handler

$\quad \mathbf{return}\ x \mapsto x$

$\quad \langle \mathsf{choose}\ () \to r\rangle \mapsto r\ \mathsf{true}$

$\mathbf{handle}\ 42\ \mathbf{with}\ \mathsf{trueChoice} \Longrightarrow 42$

$\mathbf{handle}\ \mathsf{toss}\ ()\ \mathbf{with}\ \mathsf{trueChoice} \Longrightarrow \mathsf{Heads}$

$\mathsf{allChoices} : A!(E \uplus \{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\}) \Rightarrow \mathsf{List}\ A!E$

$\mathsf{allChoices} =$ — non-linear handler

$\quad \mathbf{return}\ x \mapsto [x]$

$\quad \langle \mathsf{choose}\ () \to r\rangle \mapsto r\ \mathsf{true} \mathbin{+\!\!+} r\ \mathsf{false}$

Example 1: choice and failure

Handlers

$$\mathsf{maybeFail} : A!(E \uplus \{\mathsf{fail} : a.1 \twoheadrightarrow a\}) \Rightarrow \mathsf{Maybe}\ A!E$$

$\mathsf{maybeFail} =$ — exception handler

$\quad \mathbf{return}\ x \mapsto \mathsf{Just}\ x$

$\quad \langle \mathsf{fail}\ () \rangle \mapsto \mathsf{Nothing}$

$\mathbf{handle}\ 42\ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Just}\ 42$

$\mathbf{handle}\ \mathsf{fail}\ ()\ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Nothing}$

$$\mathsf{trueChoice} : A!(E \uplus \{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\}) \Rightarrow A!E$$

$\mathsf{trueChoice} =$ — linear handler

$\quad \mathbf{return}\ x \mapsto x$

$\quad \langle \mathsf{choose}\ () \rightarrow r \rangle \mapsto r\ \mathsf{true}$

$\mathbf{handle}\ 42\ \mathbf{with}\ \mathsf{trueChoice} \Longrightarrow 42$

$\mathbf{handle}\ \mathsf{toss}\ ()\ \mathbf{with}\ \mathsf{trueChoice} \Longrightarrow \mathsf{Heads}$

$$\mathsf{allChoices} : A!(E \uplus \{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\}) \Rightarrow \mathsf{List}\ A!E$$

$\mathsf{allChoices} =$ — non-linear handler

$\quad \mathbf{return}\ x \mapsto [x]$

$\quad \langle \mathsf{choose}\ () \rightarrow r \rangle \mapsto r\ \mathsf{true} +\!\!+\ r\ \mathsf{false}$

$\mathbf{handle}\ 42\ \mathbf{with}\ \mathsf{allChoices} \Longrightarrow [42]$

$\mathbf{handle}\ \mathsf{toss}\ ()\ \mathbf{with}\ \mathsf{allChoices} \Longrightarrow [\mathsf{Heads}, \mathsf{Tails}]$

Example 1: choice and failure

Handler composition

handle (**handle** drunkTosses 2 **with** maybeFail) **with** allChoices

Example 1: choice and failure

Handler composition

$\textbf{handle}\,(\textbf{handle}\,\mathsf{drunkTosses}\,2\,\textbf{with}\,\mathsf{maybeFail})\,\textbf{with}\,\mathsf{allChoices} : \mathsf{List}\,(\mathsf{Maybe}\,(\mathsf{List}\,\mathsf{Toss})) \Longrightarrow$

Example 1: choice and failure

Handler composition

$$\textbf{handle}\,(\textbf{handle}\;\mathsf{drunkTosses}\;2\;\textbf{with}\;\mathsf{maybeFail})\;\textbf{with}\;\mathsf{allChoices} : \mathsf{List}\,(\mathsf{Maybe}\,(\mathsf{List}\;\mathsf{Toss})) \Longrightarrow$$

$$\begin{array}{lll} [\mathsf{Just}\,[\mathsf{Heads}, \mathsf{Heads}], & \mathsf{Just}\,[\mathsf{Heads}, \mathsf{Tails}], & \mathsf{Nothing}, \\ \;\mathsf{Just}\,[\mathsf{Tails}, \mathsf{Heads}], & \mathsf{Just}\,[\mathsf{Tails}, \mathsf{Tails}], & \mathsf{Nothing}, \\ \;\mathsf{Nothing}] & & \end{array}$$

Example 1: choice and failure

Handler composition

$$\textbf{handle}\,(\textbf{handle}\ \text{drunkTosses}\ 2\ \textbf{with}\ \text{maybeFail})\ \textbf{with}\ \text{allChoices} : \text{List}\,(\text{Maybe}\,(\text{List Toss})) \Longrightarrow$$

$$\begin{aligned}[&\text{Just}\,[\text{Heads}, \text{Heads}],\ \text{Just}\,[\text{Heads}, \text{Tails}],\ \text{Nothing},\\ &\text{Just}\,[\text{Tails}, \text{Heads}],\ \text{Just}\,[\text{Tails}, \text{Tails}],\ \text{Nothing},\\ &\text{Nothing}]\end{aligned}$$

$$\textbf{handle}\,(\textbf{handle}\ \text{drunkTosses}\ 2\ \textbf{with}\ \text{allChoices})\ \textbf{with}\ \text{maybeFail}$$

Example 1: choice and failure

Handler composition

handle (**handle** drunkTosses 2 **with** maybeFail) **with** allChoices : List (Maybe (List Toss)) $\Longrightarrow$

[Just [Heads, Heads], Just [Heads, Tails], Nothing,
Just [Tails, Heads], Just [Tails, Tails], Nothing,
Nothing]

handle (**handle** drunkTosses 2 **with** allChoices) **with** maybeFail : Maybe (List (List Toss)) $\Longrightarrow$

Example 1: choice and failure

Handler composition

$\textbf{handle}\,(\textbf{handle}\ \mathsf{drunkTosses}\ 2\ \textbf{with}\ \mathsf{maybeFail})\ \textbf{with}\ \mathsf{allChoices} : \mathsf{List}\,(\mathsf{Maybe}\,(\mathsf{List}\ \mathsf{Toss})) \Longrightarrow$

$[\mathsf{Just}\ [\mathsf{Heads}, \mathsf{Heads}],\ \mathsf{Just}\ [\mathsf{Heads}, \mathsf{Tails}],\ \mathsf{Nothing},$
$\ \mathsf{Just}\ [\mathsf{Tails}, \mathsf{Heads}],\ \mathsf{Just}\ [\mathsf{Tails}, \mathsf{Tails}],\ \mathsf{Nothing},$
$\ \mathsf{Nothing}]$

$\textbf{handle}\,(\textbf{handle}\ \mathsf{drunkTosses}\ 2\ \textbf{with}\ \mathsf{allChoices})\ \textbf{with}\ \mathsf{maybeFail} : \mathsf{Maybe}\,(\mathsf{List}\,(\mathsf{List}\ \mathsf{Toss})) \Longrightarrow$

$\mathsf{Nothing}$

Operational semantics (deep handlers)

Reduction rules

$$
\begin{aligned}
\textbf{let}\ x = V\ \textbf{in}\ N &\rightsquigarrow N[V/x] \\
\textbf{handle}\ V\ \textbf{with}\ H &\rightsquigarrow N[V/x] \\
\textbf{handle}\ \mathcal{E}[\mathsf{op}\ V]\ \textbf{with}\ H &\rightsquigarrow N_{\mathsf{op}}[V/p,\ (\lambda x.\textbf{handle}\ \mathcal{E}[x]\ \textbf{with}\ H)/r], \quad \mathsf{op} \# \mathcal{E}
\end{aligned}
$$

where

$$
\begin{aligned}
\text{where } H = \textbf{return}\, x &\mapsto N \\
\langle \mathsf{op}_1\ p \to r \rangle &\mapsto N_{\mathsf{op}_1} \\
&\ldots \\
\langle \mathsf{op}_k\ p \to r \rangle &\mapsto N_{\mathsf{op}_k}
\end{aligned}
$$

Evaluation contexts

$$
\mathcal{E} ::= [\,] \mid \textbf{let}\ x = \mathcal{E}\ \textbf{in}\ N \mid \textbf{handle}\ \mathcal{E}\ \textbf{with}\ H
$$

Typing rules (deep handlers)

Effects

$$E ::= \emptyset \mid E \uplus \{\mathsf{op} : A \twoheadrightarrow B\}$$

Computations

$$C, D ::= A!E$$

Operations

$$\frac{\Gamma \vdash V : A}{\Gamma \vdash \mathsf{op}\ V : B!(E \uplus \{\mathsf{op} : A \twoheadrightarrow B\})}$$

Handlers

$$\frac{\Gamma \vdash M : C \qquad \Gamma \vdash H : C \Rightarrow D}{\Gamma \vdash \textbf{handle}\ M\ \textbf{with}\ H : D}$$

$$\frac{\Gamma, x : A \vdash N : D \qquad [\mathsf{op}_i : A_i \twoheadrightarrow B_i \in E]_i \qquad [\Gamma, p : A_i, r : B_i \to D \vdash N_i : D]_i}{\Gamma \vdash \begin{array}{l}\textbf{return}\ x \mapsto N \\ (\langle \mathsf{op}_i\ p \to r \rangle \mapsto N_i)_i\end{array} : A!E \Rightarrow D}$$

Typing rules (deep handlers)

Effects

$$E ::= \emptyset \mid E \uplus \{\mathsf{op} : A \twoheadrightarrow B\}$$

Computations

$$C, D ::= A!E$$

Operations

$$\frac{\Gamma \vdash V : A}{\Gamma \vdash \mathsf{op}\ V : B!(E \uplus \{\mathsf{op} : A \twoheadrightarrow B\})}$$

Handlers

$$\frac{\Gamma \vdash M : C \qquad \Gamma \vdash H : C \Rightarrow D}{\Gamma \vdash \textbf{handle}\ M\ \textbf{with}\ H : D}$$

$$\frac{\Gamma, x : A \vdash N : D \qquad [\mathsf{op}_i : A_i \twoheadrightarrow B_i \in E]_i \qquad [\Gamma, p : A_i, r : B_i \to D \vdash N_i : D]_i}{\Gamma \vdash \begin{array}{l}\textbf{return}\ x \mapsto N \\ (\langle \mathsf{op}_i\ p \to r\rangle \mapsto N_i)_i\end{array} : A!E \Rightarrow D}$$

Exercise: Adapt the typing rules to accommodate parametric operations

What is an effect handler?

What is an effect handler?

- A **modular** interpreter for effectful computations

What is an effect handler?

- A **modular** interpreter for effectful computations
- A generalisation of an exception handler
  - based on exceptional syntax [Benton and Kennedy, 2001]

$$\textbf{let}\ x = (\textbf{try}\ M\ \textbf{with}\ H)\ \textbf{in}\ N \qquad \text{conventional exception handlers}$$

$$\Downarrow$$

$$\textbf{try}\ M\ \textbf{as}\ x\ \textbf{in}\ N\ \textbf{otherwise}\ H \qquad \text{exceptional syntax}$$

$$\Downarrow$$

$$\textbf{handle}\ M\ \textbf{with}\ \{\textbf{return}\ x \mapsto N; H\} \qquad \text{effect handlers}$$

    success continuations aid composition, optimisation, and reasoning
  - resumable

What is an effect handler?

- A **modular** interpreter for effectful computations
- A generalisation of an exception handler
  - based on exceptional syntax [Benton and Kennedy, 2001]

$$\begin{array}{cl} \textbf{let}\ x = (\textbf{try}\ M\ \textbf{with}\ H)\ \textbf{in}\ N & \text{conventional exception handlers} \\ \Downarrow & \\ \textbf{try}\ M\ \textbf{as}\ x\ \textbf{in}\ N\ \textbf{otherwise}\ H & \text{exceptional syntax} \\ \Downarrow & \\ \textbf{handle}\ M\ \textbf{with}\ \{\textbf{return}\ x \mapsto N; H\} & \text{effect handlers} \end{array}$$

  success continuations aid composition, optimisation, and reasoning
  - resumable
- A morphism between (free) algebras

What is an effect handler?

- A **modular** interpreter for effectful computations
- A generalisation of an exception handler
  - based on exceptional syntax [Benton and Kennedy, 2001]

$\mathbf{let}\ x = (\mathbf{try}\ M\ \mathbf{with}\ H)\ \mathbf{in}\ N$	conventional exception handlers
$\Downarrow$	
$\mathbf{try}\ M\ \mathbf{as}\ x\ \mathbf{in}\ N\ \mathbf{otherwise}\ H$	exceptional syntax
$\Downarrow$	
$\mathbf{handle}\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N; H\}$	effect handlers

  success continuations aid composition, optimisation, and reasoning
  - resumable
- A morphism between (free) algebras
- A fold (catamorphism) over a command-response tree

What is an effect handler?

- A **modular** interpreter for effectful computations
- A generalisation of an exception handler
  - based on exceptional syntax [Benton and Kennedy, 2001]

$\mathbf{let}\ x = (\mathbf{try}\ M\ \mathbf{with}\ H)\ \mathbf{in}\ N$	conventional exception handlers
$\Downarrow$	
$\mathbf{try}\ M\ \mathbf{as}\ x\ \mathbf{in}\ N\ \mathbf{otherwise}\ H$	exceptional syntax
$\Downarrow$	
$\mathbf{handle}\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N; H\}$	effect handlers

  success continuations aid composition, optimisation, and reasoning
  - resumable
- A morphism between (free) algebras
- A fold (catamorphism) over a command-response tree
- A structured delimited control operator

What is an effect handler?

- A **modular** interpreter for effectful computations
- A generalisation of an exception handler
  - based on exceptional syntax [Benton and Kennedy, 2001]

$$\begin{array}{cl} \textbf{let}\ x = (\textbf{try}\ M\ \textbf{with}\ H)\ \textbf{in}\ N & \text{conventional exception handlers} \\ \Downarrow & \\ \textbf{try}\ M\ \textbf{as}\ x\ \textbf{in}\ N\ \textbf{otherwise}\ H & \text{exceptional syntax} \\ \Downarrow & \\ \textbf{handle}\ M\ \textbf{with}\ \{\textbf{return}\ x \mapsto N; H\} & \text{effect handlers} \end{array}$$

    success continuations aid composition, optimisation, and reasoning
  - resumable
- A morphism between (free) algebras
- A fold (catamorphism) over a command-response tree
- A structured delimited control operator
- A composable user-defined operating system

Example 2: generators

Effect signature

$$\{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}$$

Example 2: generators

Effect signature

$$\{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}$$

A simple generator

$$\begin{aligned}
&\mathsf{nats} : \mathsf{Nat} \to 1!(E \uplus \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}) \\
&\mathsf{nats}\, n = \mathsf{send}\, n; \mathsf{nats}\,(n + 1)
\end{aligned}$$

Example 2: generators

Effect signature

$$\{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}$$

A simple generator

$$\begin{aligned}
&\mathsf{nats} : \mathsf{Nat} \to 1!(E \uplus \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}) \\
&\mathsf{nats}\, n = \mathsf{send}\, n; \mathsf{nats}\,(n+1)
\end{aligned}$$

Handler — function that returns a handler

$$\begin{aligned}
&\mathsf{until} : \mathsf{Nat} \to 1!(E \uplus \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}) \Rightarrow \mathsf{List}\ \mathsf{Nat}!E \\
&\mathsf{until}\, \mathit{stop} = \\
&\quad \mathbf{return}\,() \qquad\quad \mapsto [] \\
&\quad \langle \mathsf{send}\, n \to r \rangle \ \mapsto \mathbf{if}\ n < \mathit{stop}\ \mathbf{then}\ n :: r\,() \\
&\qquad\qquad\qquad\qquad\quad \mathbf{else}\ []
\end{aligned}$$

Example 2: generators

Effect signature

$$\{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}$$

A simple generator

$$\begin{aligned}
&\mathsf{nats} : \mathsf{Nat} \to 1!(E \uplus \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}) \\
&\mathsf{nats}\, n = \mathsf{send}\; n; \mathsf{nats}\,(n+1)
\end{aligned}$$

Handler — function that returns a handler

$$\begin{aligned}
&\mathsf{until} : \mathsf{Nat} \to 1!(E \uplus \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}) \Rightarrow \mathsf{List\ Nat}!E \\
&\mathsf{until}\, \mathit{stop} = \\
&\quad \mathbf{return}\,() \qquad\quad \mapsto [] \\
&\quad \langle \mathsf{send}\; n \to r \rangle \;\; \mapsto \mathbf{if}\; n < \mathit{stop}\; \mathbf{then}\; n :: r\,() \\
&\qquad\qquad\qquad\qquad\quad\;\, \mathbf{else}\; []
\end{aligned}$$

$$\mathbf{handle}\; \mathsf{nats}\, 0\; \mathbf{with}\; \mathsf{until}\, 8 \Longrightarrow [0, 1, 2, 3, 4, 5, 6, 7]$$

Example 3: cooperative concurrency

Effect signature

$$\{\textsf{yield} : 1 \twoheadrightarrow 1\}$$

Example 3: cooperative concurrency

Effect signature

$$\{\mathsf{yield} : 1 \twoheadrightarrow 1\}$$

Two cooperative lightweight threads

$$\begin{aligned} \mathsf{tA}\,() &= \mathsf{print}\,(\text{“A1 ”}); \mathsf{yield}\,(); \mathsf{print}\,(\text{“A2 ”}) \\ \mathsf{tB}\,() &= \mathsf{print}\,(\text{“B1 ”}); \mathsf{yield}\,(); \mathsf{print}\,(\text{“B2 ”}) \end{aligned}$$

Example 3: cooperative concurrency

Types

$$\text{Thread}\ E = 1 \to 1!(E \uplus \{\text{yield} : 1 \twoheadrightarrow 1\})$$

Handler — recursive function containing a shallow handler

$$\begin{aligned}
&\text{cooperate} : \text{List}\ (\text{Thread}\ E) \to 1!E \\
&\text{cooperate}\ [] = () \\
&\text{cooperate}\ (t :: ts) = \\
&\quad \textbf{handle}^{\dagger}\ t()\ \textbf{with} \\
&\qquad \textbf{return}\ () \qquad\quad \mapsto \text{cooperate}\ (ts) \\
&\qquad \langle \text{yield}\ () \to t \rangle \mapsto \text{cooperate}\ (ts +\!\!+ [t])
\end{aligned}$$

Example 3: cooperative concurrency

Types

$$\text{Thread } E = 1 \to 1!(E \uplus \{\text{yield} : 1 \twoheadrightarrow 1\})$$

Handler — recursive function containing a shallow handler

$$\begin{aligned}
&\text{cooperate} : \text{List}\,(\text{Thread } E) \to 1!E \\
&\text{cooperate}\,[] = () \\
&\text{cooperate}\,(t :: ts) = \\
&\quad \mathbf{handle}^{\dagger}\; t()\; \mathbf{with} \\
&\qquad \mathbf{return}\,() \qquad\quad \mapsto \text{cooperate}\,(ts) \\
&\qquad \langle \text{yield}\,() \to t \rangle \mapsto \text{cooperate}\,(ts +\!\!+ [t])
\end{aligned}$$

$$\text{cooperate}\,[tA, tB] \Longrightarrow ()$$

A1 B1 A2 B2

Operational semantics (shallow handlers)

Reduction rules

$$
\begin{aligned}
\textbf{let}\ x = V\ \textbf{in}\ N &\rightsquigarrow N[V/x] \\
\textbf{handle}^{\dagger}\ V\ \textbf{with}\ H &\rightsquigarrow N[V/x] \\
\textbf{handle}^{\dagger}\ \mathcal{E}[\mathsf{op}\ V]\ \textbf{with}\ H &\rightsquigarrow N_{\mathsf{op}}[V/p, (\lambda x.\mathcal{E}[x])/r], \quad \mathsf{op} \# \mathcal{E}
\end{aligned}
$$

$$
\begin{aligned}
\text{where}\ H = \textbf{return}\, x &\mapsto N \\
\langle \mathsf{op}_1\ p \to r \rangle &\mapsto N_{\mathsf{op}_1} \\
&\dots \\
\langle \mathsf{op}_k\ p \to r \rangle &\mapsto N_{\mathsf{op}_k}
\end{aligned}
$$

Evaluation contexts

$$
\mathcal{E} ::= [\ ] \mid \textbf{let}\ x = \mathcal{E}\ \textbf{in}\ N \mid \textbf{handle}^{\dagger}\ \mathcal{E}\ \textbf{with}\ H
$$

Typing rules (shallow handlers)

Effects

$$E ::= \emptyset \mid E \uplus \{\mathsf{op} : A \twoheadrightarrow B\}$$

Computations

$$C, D ::= A!E$$

Operations

$$\frac{\Gamma \vdash V : A}{\Gamma \vdash \mathsf{op}\ V : B!(E \uplus \{\mathsf{op} : A \twoheadrightarrow B\})}$$

Handlers

$$\frac{\Gamma \vdash M : C \qquad \Gamma \vdash H : C \Rightarrow^{\dagger} D}{\Gamma \vdash \textbf{handle}^{\dagger}\ M\ \textbf{with}\ H : D}$$

$$\frac{\Gamma, x : A \vdash N : D \qquad [\mathsf{op}_i : A_i \twoheadrightarrow B_i \in E]_i \qquad [\Gamma, p : A_i, r : B_i \to A!E \vdash N_i : D]_i}{\Gamma \vdash \begin{array}{l}\textbf{return}\ x \mapsto N \\ (\langle \mathsf{op}_i\ p \to r\rangle \mapsto N_i)_i\end{array} : A!E \Rightarrow^{\dagger} D}$$

Example 4: cooperative concurrency with higher-order fork

Effect signature — recursive effect signature

$$\mathsf{Coop}\ E = E \uplus \{\mathsf{yield} : 1 \twoheadrightarrow 1,\quad \mathsf{fork} : (1 \to 1!\mathsf{Coop}\ E) \twoheadrightarrow 1\}$$

Example 4: cooperative concurrency with higher-order fork

Effect signature — recursive effect signature

$$\text{Coop}\ E = E \uplus \{\text{yield} : 1 \twoheadrightarrow 1,\quad \text{fork} : (1 \to 1!\text{Coop}\ E) \twoheadrightarrow 1\}$$

A single cooperative program

$$\begin{aligned}
&\text{main} : 1 \to 1!\text{Coop}\ E \\
&\text{main}\,() = \text{print “M1 ”}; \text{fork}\,(\lambda().\text{print “A1 ”}; \text{yield}\,(); \text{print “A2 ”}); \\
&\qquad\qquad\ \text{print “M2 ”}; \text{fork}\,(\lambda().\text{print “B1 ”}; \text{yield}\,(); \text{print “B2 ”}); \\
&\qquad\qquad\ \text{print “M3 ”}
\end{aligned}$$

Example 4: cooperative concurrency with higher-order fork

Types

$$\text{Thread } E = 1 \to 1!\text{Coop } E$$

Handler

$$
\begin{array}{l}
\text{cooperate} : \text{List}\,(\text{Thread } E) \to 1!E \\
\text{cooperate}\,[] = () \\
\text{cooperate}\,(t :: ts) = \\
\quad \textbf{handle}^{\dagger}\; t()\; \textbf{with} \\
\quad\quad \begin{array}{lcl}
\textbf{return}\,() & \mapsto & \text{cooperate}\,(ts) \\
\langle \text{yield}\,() \to t \rangle & \mapsto & \text{cooperate}\,(ts +\!\!+ [t]) \\
\langle \text{fork}\; t \to r \rangle & \mapsto & \text{cooperate}\,(t :: ts +\!\!+ [r])
\end{array}
\end{array}
$$

Example 4: cooperative concurrency with higher-order fork

Types

$$\text{Thread}\ E = 1 \to 1!\text{Coop}\ E$$

Handler

$$
\begin{array}{l}
\text{cooperate} : \text{List}\,(\text{Thread}\ E) \to 1!E \\
\text{cooperate}\,[] = () \\
\text{cooperate}\,(t :: ts) = \\
\quad \textbf{handle}^{\dagger}\ t()\ \textbf{with} \\
\qquad \textbf{return}\,() \qquad \mapsto \text{cooperate}\,(ts) \\
\qquad \langle \text{yield}\,() \to t \rangle \mapsto \text{cooperate}\,(ts +\!\!+ [t]) \\
\qquad \langle \text{fork}\ t \to r \rangle \mapsto \text{cooperate}\,(t :: ts +\!\!+ [r])
\end{array}
$$

$$\text{cooperate}\,[\text{main}] \implies ()$$

M1 A1 M2 B1 A2 M3 B2

Example 4: cooperative concurrency with higher-order fork

Types

$$\text{Thread } E = 1 \to 1!\text{Coop } E$$

Handler

$$
\begin{array}{l}
\text{cooperate} : \text{List}\,(\text{Thread } E) \to 1!E \\
\text{cooperate}\,[] = () \\
\text{cooperate}\,(t :: ts) = \\
\quad \textbf{handle}^{\dagger}\; t()\; \textbf{with} \\
\quad\quad \begin{array}{ll}
\textbf{return}\,() & \mapsto \text{cooperate}\,(ts) \\
\langle \text{yield}\,() \to t \rangle & \mapsto \text{cooperate}\,(ts +\!\!+ [t]) \\
\langle \text{fork}\; t \to r \rangle & \mapsto \text{cooperate}\,(r :: ts +\!\!+ [t])
\end{array}
\end{array}
$$

Example 4: cooperative concurrency with higher-order fork

Types

$$\text{Thread}\ E = 1 \to 1!\text{Coop}\ E$$

Handler

$$
\begin{array}{l}
\text{cooperate} : \text{List}\,(\text{Thread}\ E) \to 1!E \\
\text{cooperate}\,[] = () \\
\text{cooperate}\,(t :: ts) = \\
\quad \textbf{handle}^{\dagger}\ t()\ \textbf{with} \\
\qquad \textbf{return}\,() \qquad\quad \mapsto \text{cooperate}\,(ts) \\
\qquad \langle \text{yield}\,() \to t \rangle \mapsto \text{cooperate}\,(ts +\!\!+ [t]) \\
\qquad \langle \text{fork}\ t \to r \rangle \quad \mapsto \text{cooperate}\,(r :: ts +\!\!+ [t])
\end{array}
$$

$$\text{cooperate}\,[\text{main}] \implies ()$$

M1 M2 M3 A1 B1 A2 B2

Built-in effects

Output

$$\mathsf{Output} = \{\mathsf{print} : \mathsf{String} \twoheadrightarrow 1\}$$

Generative state

$$\begin{aligned}\mathsf{GenState} = \{&\mathsf{new} &: a.\ & a \twoheadrightarrow \mathsf{Ref}\ a,\\ &\mathsf{write} &: a.\ & (\mathsf{Ref}\ a \times a) \twoheadrightarrow 1,\\ &\mathsf{read} &: a.\ & \mathsf{Ref}\ a \twoheadrightarrow a\}\end{aligned}$$

Example 5: actors

Process ids

$$\mathsf{Pid}\ a = \mathsf{Ref}\ (\mathsf{List}\ a)$$

Effect signature

$$\begin{aligned}\mathsf{Actor}\ a = \{\mathsf{self} &: & 1 &\twoheadrightarrow \mathsf{Pid}\ a,\\ \mathsf{spawn} &: b.\ & (1 \to 1!\mathsf{Actor}\ b) &\twoheadrightarrow \mathsf{Pid}\ b,\\ \mathsf{send} &: b.\ & (b \times \mathsf{Pid}\ b) &\twoheadrightarrow 1,\\ \mathsf{recv} &: & 1 &\twoheadrightarrow a\}\end{aligned}$$

Example 5: actors

Process ids

$$\mathsf{Pid}\ a = \mathsf{Ref}\ (\mathsf{List}\ a)$$

Effect signature

$$\begin{aligned}\mathsf{Actor}\ a = \{\mathsf{self} &: & 1 &\twoheadrightarrow \mathsf{Pid}\ a,\\ \mathsf{spawn} &: b.\ & (1 \to 1!\mathsf{Actor}\ b) &\twoheadrightarrow \mathsf{Pid}\ b,\\ \mathsf{send} &: b.\ & (b \times \mathsf{Pid}\ b) &\twoheadrightarrow 1,\\ \mathsf{recv} &: & 1 &\twoheadrightarrow a\}\end{aligned}$$

An actor chain

$$\begin{aligned}&\mathsf{spawnMany} : \mathsf{Pid\,String} \to \mathsf{Int} \to 1!(E \uplus \mathsf{Actor\,String})\\ &\mathsf{spawnMany}\ p\ 0 = \mathsf{send}\ (\text{“ping!”}, p)\\ &\mathsf{spawnMany}\ p\ n = \mathsf{spawnMany}\ (\mathsf{spawn}\ (\lambda().\mathbf{let}\ s = \mathsf{recv}\ ()\ \mathbf{in}\ \mathsf{print}\ \text{“.”}; \mathsf{send}\ (s, p)))\ (n-1)\end{aligned}$$

$$\begin{aligned}&\mathsf{chain} : \mathsf{Int} \to 1!(E \uplus \mathsf{Actor\,String} \uplus \mathsf{Output})\\ &\mathsf{chain}\ n = \mathsf{spawnMany}\ (\mathsf{self}\ ())\ n; \mathbf{let}\ s = \mathsf{recv}\ ()\ \mathbf{in}\ \mathsf{print}\ s\end{aligned}$$

Example 5: actors — via cooperative concurrency

$$
\begin{array}{l}
\mathsf{act} : \mathsf{Pid}\, a \to (1 \to 1!(E \uplus \mathsf{Actor}\, a)) \to 1!\mathsf{Coop}\ (E \uplus \mathsf{GenState}) \\
\mathsf{act}\ \mathit{mine}\ t = \mathbf{handle}^{\dagger}\ t\,()\ \mathbf{with} \\
\quad \begin{array}{ll}
\mathbf{return}\,() & \mapsto () \\
\langle \mathsf{self}\,() \to r \rangle & \mapsto \mathsf{act}\ \mathit{mine}\,(\lambda().r\ \mathit{mine}) \\
\langle \mathsf{spawn}\ \mathit{you} \to r \rangle & \mapsto \mathbf{let}\ \mathit{yours} = \mathsf{new}\,[]\ \mathbf{in} \\
& \quad\ \ \mathsf{fork}\,(\lambda().\mathsf{act}\ \mathit{yours}\,(\mathit{you}\,()));\ \mathsf{act}\ \mathit{mine}\,(\lambda().r\ \mathit{yours}) \\
\langle \mathsf{send}\,(m, \mathit{yours}) \to r \rangle & \mapsto \mathbf{let}\ \mathit{ms} = \mathsf{read}\ \mathit{yours}\ \mathbf{in} \\
& \quad\ \ \mathsf{write}\,(\mathit{yours}, \mathit{ms} +\!\!+ [m]);\ \mathsf{act}\ \mathit{mine}\ r \\
\langle \mathsf{recv}\,() \to r \rangle & \mapsto \mathbf{letrec}\ \mathit{recvWhenReady}\ () = \\
& \qquad \mathbf{case}\ \mathsf{read}\ \mathit{mine}\ \mathbf{of} \\
& \qquad\quad \begin{array}{ll}
[] & \mapsto \mathsf{yield}\,();\ \mathit{recvWhenReady}\ () \\
(m :: \mathit{ms}) & \mapsto \mathsf{write}\,(\mathit{mine}, \mathit{ms});\ \mathsf{act}\ \mathit{mine}\,(\lambda().r\ m)
\end{array} \\
& \quad\ \ \mathbf{in}\ \mathit{recvWhenReady}\ ()
\end{array}
\end{array}
$$

Example 5: actors — via cooperative concurrency

$$\mathsf{cooperate}\,[\mathsf{act}\,(\mathsf{new}\,[])\,(\lambda().\mathsf{chain}\,64)] \implies$$

Example 5: actors — via cooperative concurrency

$$\text{cooperate}\,[\text{act}\,(\text{new}\,[])\,(\lambda().\text{chain}\,64)] \implies ()$$

....ping!

Effect handler oriented programming languages

Eff	https://www.eff-lang.org/
Effekt	https://effekt-lang.org/
Frank	https://github.com/frank-lang/frank
Helium	https://bitbucket.org/pl-uwr/helium
Links	https://www.links-lang.org/
Koka	https://github.com/koka-lang/koka
OCaml 5	https://github.com/ocamllabs/ocaml-multicore/wiki
Unison	https://www.unison-lang.org/

Expressiveness

Local transformations
(with Yannick Forster, Ohad Kammar, Matija Pretnar)
"On the expressive power of user-defined effects:
effect handlers, monadic reflection, delimited control", JFP 2019

Asymptotic complexity
(with Daniel Hillerström, John Longley)
"Asymptotic speedup via effect handlers", JFP 2024

Higher-order effects
(with Cristina Matache, Sean Moss, Sam Staton, Nicolas Wu, Zhixuan Yang)
"Scoped effects as parameterised algebraic theories", ESOP 2024

Effect handlers for imperative languages

C++
(with Dan Ghica, Maciej Piróg, Marcello Maroñas Bravo)
"High-level effect handlers in C++", OOPSLA 2022

WebAssembly
(with Arjun Guha, Daniel Hillerström, Daan Leijen, Luna Phipps-Costin, Matija Pretnar, Andreas Rossberg, KC Sivaramakrishnan)
"Continuing WebAssembly with effect handlers", OOPSLA 2023

C
(with Mario Alvarez-Picallo, Teodoro Freund, Dan Ghica)
"Effect handlers for C via coroutines", OOPSLA 2024

Effect type systems

Frank
(with Lukas Convent, Conor McBride, Craig McLaughlin)
"Doo Bee Doo Bee Doo", JFP 2020

Combining linear resources with effect handlers
(with Daniel Hillerström, J. Garrett Morris, Wenhao Tang)
"Soundly handling linearity", POPL 2024

Modal effect types
(with Stephen Dolan, Daniel Hillerström, Anton Lorenzen, Wenhao Tang, Leo White)
"Modal effect types", arXiv 2024

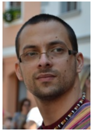
Related projects

EPOCH: Effectful Programming on Capability Hardware
(with Ian Stark, Brian Campbell, Wilmer Ricciotti)
funded by Edinburgh-Huawei joint lab

UCFX: Universal Composability with Effects and Handlers
(with Markulf Kohlweiss, Danel Ahman, Pooya Farshim, Sabine Oeschner, Jesse Sigal)
funded by Input Output Research Hub

Resources

Jeremy Yallop's effects bibliography
https://github.com/yallop/effects-bibliography



Matija Pretnar's tutorial
"An introduction to algebraic effects and handlers", MFPS 2015

Andrej Bauer's tutorial
"What is algebraic about algebraic effects and handlers?", OPLSS 2018

Daniel Hillerström's PhD thesis
"Foundations for programming and implementing effect handlers", 2022