Effect handler oriented programming

Sam Lindley

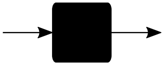
The University of Edinburgh

SPLV 2022

What is an effect?

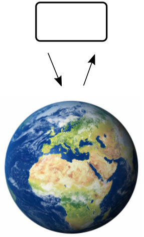
Effects

Programs as black boxes (Church-Turing model)?



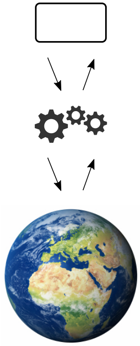
Effects

Programs must interact with their environment



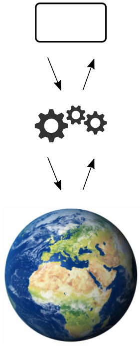
Effects

Programs must interact with their environment



Effects

Programs must interact with their environment

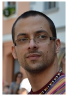


Effects are pervasive

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

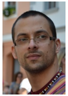
Typically ad hoc and hard-wired

Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009

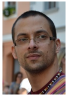
Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009

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

Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009

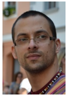
Composable and **customisable** user-defined interpretation of effects in general

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

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

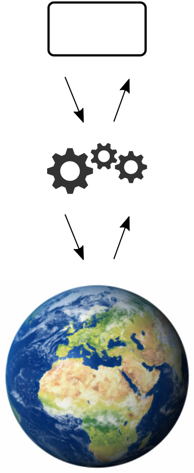
Give programmer direct access to **context**

(c.f. resumable exceptions, monads, delimited control)

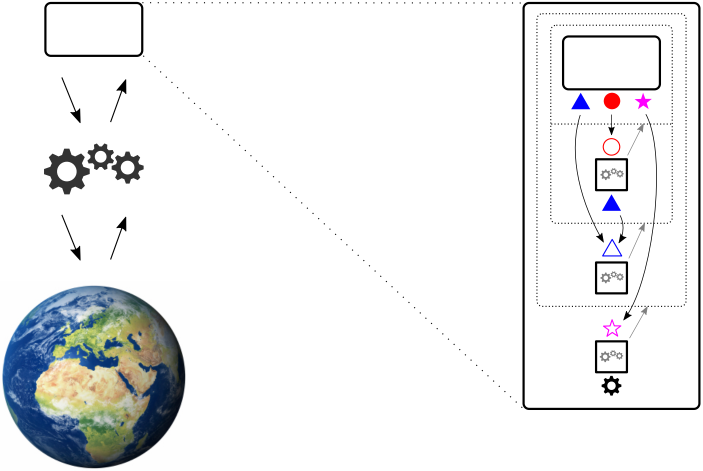
Growing industrial interest

GitHub	`semantic`	Code analysis library ($>$ 25 million repositories)
f	React	JavaScript UI library ($>$ 2 million websites)
Uber	Pyro	Statistical inference (10% ad spend saving)

Effect handlers as composable user-defined operating systems



Effect handlers as composable user-defined operating systems



Example 1: choice and failure

Effect signature

$$\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}, \quad \mathsf{fail} : 1 \twoheadrightarrow 0\}$$

Example 1: choice and failure

Effect signature

$$\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}, \quad \mathsf{fail} : 1 \twoheadrightarrow 0\}$$

Drunk coin tossing

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

$$\begin{array}{l}
\mathsf{drunkToss} : 1 \to \mathsf{Toss}!(E \uplus \{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}, \quad \mathsf{fail} : 1 \twoheadrightarrow 0\}) \\
\mathsf{drunkToss}\,() = \mathbf{if}\ \mathsf{choose}\,()\ \mathbf{then} \\
\qquad\qquad\qquad \mathbf{if}\ \mathsf{choose}\,()\ \mathbf{then}\ \mathsf{Heads}\ \mathbf{else}\ \mathsf{Tails} \\
\qquad\qquad\quad \mathbf{else} \\
\qquad\qquad\qquad \mathbf{absurd}\,(\mathsf{fail}\,())
\end{array}$$

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

Example 1: choice and failure

Handlers

$$\text{maybeFail} : A!(E \uplus \{\text{fail} : 1 \twoheadrightarrow 0\}) \Rightarrow \text{Maybe } A!E$$

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

$$\begin{aligned} &\textbf{return}\, x &&\mapsto \text{Just}\, x \\ &\langle \text{fail}\, () \rangle &&\mapsto \text{Nothing} \end{aligned}$$

Example 1: choice and failure

Handlers

$$\mathsf{maybeFail} : A!(E \uplus \{\mathsf{fail} : 1 \twoheadrightarrow 0\}) \Rightarrow \mathsf{Maybe}\ A!E$$

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

$$\begin{array}{lll} \mathbf{return}\, x & \mapsto & \mathsf{Just}\, x \\ \langle \mathsf{fail}\, () \rangle & \mapsto & \mathsf{Nothing} \end{array}$$

$$\begin{array}{lll} \mathbf{handle}\ 42\ \mathbf{with}\ \mathsf{maybeFail} & \Longrightarrow & \mathsf{Just}\, 42 \\ \mathbf{handle}\ (\mathbf{absurd}\, (\mathsf{fail}\, ()))\ \mathbf{with}\ \mathsf{maybeFail} & \Longrightarrow & \mathsf{Nothing} \end{array}$$

Example 1: choice and failure

Handlers

$$\text{maybeFail} : A!(E \uplus \{\text{fail} : 1 \twoheadrightarrow 0\}) \Rightarrow \text{Maybe } A!E$$

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

$$\begin{array}{lll} \textbf{return}\, x & \mapsto & \text{Just}\, x \\ \langle \text{fail}\, () \rangle & \mapsto & \text{Nothing} \end{array}$$

$$\textbf{handle}\ 42\ \textbf{with}\ \text{maybeFail} \Longrightarrow \text{Just}\, 42$$

$$\textbf{handle}\ (\textbf{absurd}\, (\text{fail}\, ()))\ \textbf{with}\ \text{maybeFail} \Longrightarrow \text{Nothing}$$

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

$$\text{trueChoice} = \quad \text{— linear handler}$$

$$\begin{array}{lll} \textbf{return}\, x & \mapsto & x \\ \langle \text{choose}\, () \rightarrow r \rangle & \mapsto & r\, \text{tt} \end{array}$$

Example 1: choice and failure

Handlers

$\mathsf{maybeFail} : A!(E \uplus \{\mathsf{fail} : 1 \twoheadrightarrow 0\}) \Rightarrow \mathsf{Maybe}\ A!E$

$\mathsf{maybeFail} =$ — exception handler

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

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

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

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

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

$\mathsf{trueChoice} =$ — linear handler

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

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

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

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

Example 1: choice and failure

Handlers

$\mathsf{maybeFail} : A!(E \uplus \{\mathsf{fail} : 1 \twoheadrightarrow 0\}) \Rightarrow \mathsf{Maybe}\ A!E$

$\mathsf{maybeFail} =$ — exception handler

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

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

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

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

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

$\mathsf{trueChoice} =$ — linear handler

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

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

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

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

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

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

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

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

Example 1: choice and failure

Handlers

$\mathrm{maybeFail} : A!(E \uplus \{\mathrm{fail} : 1 \twoheadrightarrow 0\}) \Rightarrow \mathrm{Maybe}\ A!E$

$\mathrm{maybeFail} =$ — exception handler

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

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

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

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

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

$\mathrm{trueChoice} =$ — linear handler

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

$\langle \mathrm{choose}\ () \to r\rangle \mapsto r\ \mathrm{tt}$

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

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

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

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

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

$\langle \mathrm{choose}\ () \to r\rangle \mapsto r\ \mathrm{tt} +\!\!+\ r\ \mathrm{ff}$

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

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

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

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

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

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

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

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)) $\Longrightarrow$
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

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

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

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

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

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 — a function that returns a handler

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

Example 2: generators

Effect signature

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

A simple generator

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

Handler — a function that returns a handler

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

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

Example 3: lightweight threads

Effect signature

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

Example 3: lightweight threads

Effect signature

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

Two cooperative lightweight threads

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

Example 3.1: lightweight threads (deep handlers)

Types

$$\mathsf{Thread}\ E = 1 \to 1!(E \uplus \{\mathsf{yield} : 1 \twoheadrightarrow 1\}) \qquad \mathsf{Res}\ E = 1 \to \mathsf{List}\ (\mathsf{Res}\ E) \to 1!E$$

Handler

$$\mathsf{coop} : 1!(\mathsf{Thread}\ E) \Rightarrow (\mathsf{List}\ (\mathsf{Res}\ E) \to 1!E)$$

$$\begin{aligned}
\mathsf{coop} = \mathbf{return}\,() &\mapsto \lambda rs.\mathbf{case}\ rs\ \mathbf{of}\ [\,] \mapsto () \\
&\qquad\qquad\qquad\quad (r :: rs) \mapsto r\ ()\ rs \\
\langle \mathsf{yield}\,() \to s \rangle &\mapsto \lambda rs.\mathbf{case}\ rs\ \mathbf{of}\ [\,] \mapsto s\ ()\ [\,] \\
&\qquad\qquad\qquad\quad (r :: rs) \mapsto r\ ()\ (rs +\!\!+ [s])
\end{aligned}$$

$$\begin{aligned}
&\mathsf{lift} : \mathsf{Thread}\ E \to \mathsf{Res}\ E \\
&\mathsf{lift}\ t = \lambda().\mathbf{handle}\ t()\ \mathbf{with}\ \mathsf{coop}
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{cooperate} : \mathsf{List}\ (\mathsf{Thread}\ E) \to 1!E \\
&\mathsf{cooperate}\ ts = \mathsf{lift}\ \mathsf{id}\ ()\ (\mathsf{map}\ \mathsf{lift}\ ts)
\end{aligned}$$

Example 3.1: lightweight threads (deep handlers)

Types

$$\mathsf{Thread}\ E = 1 \to 1!(E \uplus \{\mathsf{yield} : 1 \twoheadrightarrow 1\}) \qquad \mathsf{Res}\ E = 1 \to \mathsf{List}\ (\mathsf{Res}\ E) \to 1!E$$

Handler

$$\mathsf{coop} : 1!(\mathsf{Thread}\ E) \Rightarrow (\mathsf{List}\ (\mathsf{Res}\ E) \to 1!E)$$

$$\begin{aligned}
\mathsf{coop} = \mathbf{return}\ () &\mapsto \lambda rs.\mathbf{case}\ rs\ \mathbf{of}\ [\,] \mapsto () \\
&\qquad\qquad\qquad\quad (r :: rs) \mapsto r\ ()\ rs \\
\langle \mathsf{yield}\ () \to s \rangle &\mapsto \lambda rs.\mathbf{case}\ rs\ \mathbf{of}\ [\,] \mapsto s\ ()\ [\,] \\
&\qquad\qquad\qquad\quad (r :: rs) \mapsto r\ ()\ (rs +\!\!+ [s])
\end{aligned}$$

$$\begin{aligned}
&\mathsf{lift} : \mathsf{Thread}\ E \to \mathsf{Res}\ E \\
&\mathsf{lift}\ t = \lambda().\mathbf{handle}\ t()\ \mathbf{with}\ \mathsf{coop}
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{cooperate} : \mathsf{List}\ (\mathsf{Thread}\ E) \to 1!E \\
&\mathsf{cooperate}\ ts = \mathsf{lift}\ \mathsf{id}\ ()\ (\mathsf{map}\ \mathsf{lift}\ ts)
\end{aligned}$$

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

A1 B1 A2 B2

Example 3.2: lightweight threads (parameterised handlers)

Types

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

Handler — parameterised handler

$$\text{coop} : \text{List } (\text{Res } E) \to 1!(E \uplus \{\text{yield} : 1 \twoheadrightarrow 1\}) \Rightarrow 1!E$$

$$\begin{array}{ll}
\text{coop}\,([]) = & \\
\quad \mathbf{return}\,() & \mapsto () \\
\quad \langle \text{yield}\,() \to r' \rangle & \mapsto r'\,[]\,()
\end{array}
\qquad
\begin{array}{ll}
\text{coop}\,(r :: rs) = & \\
\quad \mathbf{return}\,() & \mapsto r\ rs\,() \\
\quad \langle \text{yield}\,() \to r' \rangle & \mapsto r\,(rs +\!\!+ [r'])\,()
\end{array}$$

$$\begin{array}{l}
\text{lift} : \text{Thread } E \to \text{Res } E \\
\text{lift } t = \lambda rs\ ().\mathbf{handle}\ t()\ \mathbf{with}\ \text{coop } rs
\end{array}
\qquad
\begin{array}{l}
\text{cooperate} : \text{List } (\text{Thread } E) \to 1!E \\
\text{cooperate } ts = \text{lift id } (\text{map lift } ts)\ ()
\end{array}$$

Example 3.2: lightweight threads (parameterised handlers)

Types

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

Handler — parameterised handler

$$\text{coop} : \text{List } (\text{Res } E) \to 1!(E \uplus \{\text{yield} : 1 \twoheadrightarrow 1\}) \Rightarrow 1!E$$

$$\begin{array}{ll} \text{coop}\,([]) = & \\ \quad \mathbf{return}\,() & \mapsto () \\ \quad \langle \text{yield}\,() \to r' \rangle & \mapsto r'\,[]\,() \end{array} \qquad \begin{array}{ll} \text{coop}\,(r :: rs) = & \\ \quad \mathbf{return}\,() & \mapsto r\;rs\,() \\ \quad \langle \text{yield}\,() \to r' \rangle & \mapsto r\,(rs +\!\!+ [r'])\,() \end{array}$$

$$\begin{array}{l} \text{lift} : \text{Thread } E \to \text{Res } E \\ \text{lift } t = \lambda rs\;().\mathbf{handle}\; t()\;\mathbf{with}\;\text{coop } rs \end{array} \qquad \begin{array}{l} \text{cooperate} : \text{List } (\text{Thread } E) \to 1!E \\ \text{cooperate } ts = \text{lift id } (\text{map lift } ts)\;() \end{array}$$

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

A1 B1 A2 B2

Parameterised effect handlers

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

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

$$\textbf{handle } \mathcal{E}[\mathsf{op}\ V] \textbf{ with } H \leadsto N_{\mathsf{op}}[V/p,\ (\lambda x\ q.\textbf{handle } \mathcal{E}[x] \textbf{ with } H\ q)/r], \quad \mathsf{op} \# \mathcal{E}$$

Parameterised effect handlers

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

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

$$\textbf{handle}\ \mathcal{E}[\mathsf{op}\ V]\ \textbf{with}\ H \leadsto N_{\mathsf{op}}[V/p,\ (\lambda x\, q.\textbf{handle}\ \mathcal{E}[x]\ \textbf{with}\ H\ q)/r], \quad \mathsf{op} \# \mathcal{E}$$

Exercise: express parameterised handlers as deep handlers

Example 3.3: lightweight threads (shallow handlers)

Types

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

Handler — shallow handler

$$\text{cooperate} : \text{List } (\text{Thread } E) \to 1!E$$

$$\text{cooperate}\,[\,] = () \qquad\qquad \begin{array}{l} \text{cooperate}\,(r :: rs) = \textbf{handle}\; r()\;\textbf{with} \\ \quad \begin{array}{ll} \textbf{return}\,() & \mapsto \text{cooperate}\,(rs) \\ \langle \text{yield}\,() \to s\rangle & \mapsto \text{cooperate}\,(rs +\!\!+ [s]) \end{array} \end{array}$$

Example 3.3: lightweight threads (shallow handlers)

Types

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

Handler — shallow handler

$$\text{cooperate} : \text{List}\,(\text{Thread } E) \to 1!E$$

$$\text{cooperate}\,[] = () \qquad \begin{aligned} &\text{cooperate}\,(r :: rs) = \textbf{handle}\; r\,()\;\textbf{with} \\ &\quad \textbf{return}\,() \qquad\quad \mapsto \text{cooperate}\,(rs) \\ &\quad \langle \text{yield}\,() \to s \rangle \mapsto \text{cooperate}\,(rs +\!\!+ [s]) \end{aligned}$$

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

A1 B1 A2 B2

Deep effect handlers

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

$$\mathbf{handle}\ \mathcal{E}[\mathsf{op}\ V]\ \mathbf{with}\ H \leadsto N_{\mathsf{op}}[V/p,\ (\lambda x.\mathbf{handle}\ \mathcal{E}[x]\ \mathbf{with}\ H)/r], \quad \mathsf{op} \# \mathcal{E}$$

Deep effect handlers

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

$$\textbf{handle}\ \mathcal{E}[\mathsf{op}\ V]\ \textbf{with}\ H \rightsquigarrow N_{\mathsf{op}}[V/p,\ (\lambda x.\textbf{handle}\ \mathcal{E}[x]\ \textbf{with}\ H)/r], \quad \mathsf{op} \# \mathcal{E}$$

The body of the resumption r reinvokes the handler

Deep effect handlers

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

$$\textbf{handle}\ \mathcal{E}[\mathsf{op}\ V]\ \textbf{with}\ H \rightsquigarrow N_{\mathsf{op}}[V/p,\ (\lambda x.\textbf{handle}\ \mathcal{E}[x]\ \textbf{with}\ H)/r], \quad \mathsf{op} \# \mathcal{E}$$

The body of the resumption r reinvokes the handler

A deep handler performs a fold (catamorphism) on a computation tree

Shallow effect handlers

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

$$\textbf{handle}\ \mathcal{E}[\mathsf{op}\ V]\ \textbf{with}\ H \ \rightsquigarrow N_{\mathsf{op}}[V/p, (\lambda x.\mathcal{E}[x])/r], \quad \mathsf{op} \# \mathcal{E}$$

Shallow effect handlers

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

$$\textbf{handle}\ \mathcal{E}[\mathsf{op}\ V]\ \textbf{with}\ H \ \rightsquigarrow\ N_{\mathsf{op}}[V/p, (\lambda x.\mathcal{E}[x])/r], \qquad \mathsf{op} \# \mathcal{E}$$

The body of the resumption r does not reinvoke the handler

Shallow effect handlers

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

$$\textbf{handle}\ \mathcal{E}[\mathsf{op}\ V]\ \textbf{with}\ H \ \rightsquigarrow N_{\mathsf{op}}[V/p, (\lambda x.\mathcal{E}[x])/r], \quad \mathsf{op} \# \mathcal{E}$$

The body of the resumption r does not reinvoke the handler

A shallow handler performs a case-split on a computation tree

Shallow effect handlers

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

$$\textbf{handle}\ \mathcal{E}[\mathsf{op}\ V]\ \textbf{with}\ H \ \rightsquigarrow\ N_{\mathsf{op}}[V/p, (\lambda x.\mathcal{E}[x])/r], \quad \mathsf{op} \# \mathcal{E}$$

The body of the resumption r does not reinvoke the handler

A shallow handler performs a case-split on a computation tree

Exercise: express shallow handlers as deep handlers

Example 5: lightweight threads with UNIX-style fork

Effect signature

$$\mathsf{CoU}\ E = E \uplus \{\mathsf{yield} : 1 \twoheadrightarrow 1,\quad \mathsf{ufork} : 1 \twoheadrightarrow \mathsf{Bool}\}$$

Example 5: lightweight threads with UNIX-style fork

Effect signature

$$\mathsf{CoU}\ E = E \uplus \{\mathsf{yield} : 1 \twoheadrightarrow 1, \quad \mathsf{ufork} : 1 \twoheadrightarrow \mathsf{Bool}\}$$

A single cooperative program

$$\begin{aligned}
&\mathsf{main} : 1 \to \mathsf{CoU}\ E!1 \\
&\mathsf{main}\,() = \mathsf{print}\ \text{“M1 ”}; \textbf{if}\ \mathsf{ufork}\,()\ \textbf{then}\ \mathsf{print}\ \text{“A1 ”}; \mathsf{yield}\,(); \mathsf{print}\ \text{“A2 ”} \\
&\qquad\qquad \textbf{else}\ \mathsf{print}\ \text{“M2 ”}; \textbf{if}\ \mathsf{ufork}\,()\ \textbf{then}\ \mathsf{print}\ \text{“B1 ”}; \mathsf{yield}\,(); \mathsf{print}\ \text{“B2 ”}\ \textbf{else}\ \mathsf{print}\ \text{“M3 ”}
\end{aligned}$$

Example 5: lightweight threads with UNIX-style fork

Types

$$\mathsf{Thread}\ E = 1 \to \mathsf{CoU}\ E!1 \qquad\qquad \mathsf{Res}\ E = \mathsf{List}\,(\mathsf{Res}\ E) \to 1 \to 1!E$$

Parameterised handler

$$\mathsf{coop} : \mathsf{List}\,(\mathsf{Res}\ E) \to \mathsf{CoU}\ E!1 \Rrightarrow 1!E$$

$$\begin{array}{ll}
\mathsf{coop}\,([]) = & \\
\quad \mathbf{return}\,() & \mapsto ()\\
\quad \langle \mathsf{yield}\,() \to r' \rangle & \mapsto r'\,[]\,()\\
\quad \langle \mathsf{ufork}\,() \to r' \rangle & \mapsto r'\,[\lambda rs\,().r'\ rs\,\mathsf{ff}]\\
 & \qquad \mathsf{tt}
\end{array}$$

$$\begin{array}{ll}
\mathsf{coop}\,(r :: rs) = & \\
\quad \mathbf{return}\,() & \mapsto r\ rs\,()\\
\quad \langle \mathsf{yield}\,() \to r' \rangle & \mapsto r\,(rs \mathbin{+\!\!+} [r'])\,()\\
\quad \langle \mathsf{ufork}\,() \to r' \rangle & \mapsto r'\,(r :: rs \mathbin{+\!\!+} [\lambda rs\,().r'\ rs\,\mathsf{ff}])\\
 & \qquad \mathsf{tt}
\end{array}$$

Example 5: lightweight threads with UNIX-style fork

Types

$$\text{Thread } E = 1 \to \text{CoU } E!1 \qquad\qquad \text{Res } E = \text{List } (\text{Res } E) \to 1 \to 1!E$$

Parameterised handler

$$\text{coop} : \text{List } (\text{Res } E) \to \text{CoU } E!1 \Rightarrow 1!E$$

$$\begin{aligned}
&\text{coop } ([]) = \\
&\quad \mathbf{return}\,() && \mapsto () \\
&\quad \langle \text{yield}\,() \to r' \rangle && \mapsto r'\,[]\,() \\
&\quad \langle \text{ufork}\,() \to r' \rangle && \mapsto r'\,[\lambda rs\,().r'\,rs\,\text{ff}] \\
&&& \qquad \text{tt}
\end{aligned}
\qquad
\begin{aligned}
&\text{coop } (r :: rs) = \\
&\quad \mathbf{return}\,() && \mapsto r\,rs\,() \\
&\quad \langle \text{yield}\,() \to r' \rangle && \mapsto r\,(rs +\!\!+ [r'])\,() \\
&\quad \langle \text{ufork}\,() \to r' \rangle && \mapsto r'\,(r :: rs +\!\!+ [\lambda rs\,().r'\,rs\,\text{ff}]) \\
&&& \qquad \text{tt}
\end{aligned}$$

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

M1 A1 M2 B1 A2 M3 B2

Example 5: lightweight threads with UNIX-style fork

Types

$$\text{Thread } E = 1 \to \text{CoU } E!1 \qquad\qquad \text{Res } E = \text{List}\,(\text{Res } E) \to 1 \to 1!E$$

Parameterised handler

$$\text{coop} : \text{List}\,(\text{Res } E) \to \text{CoU } E!1 \Rightarrow 1!E$$

$$\begin{array}{ll}
\text{coop}\,([]) = & \\
\quad \textbf{return}\,() & \mapsto () \\
\quad \langle \text{yield}\,() \to r' \rangle & \mapsto r'\,[]\,() \\
\quad \langle \text{ufork}\,() \to r' \rangle & \mapsto r'\,[\lambda rs\,().r'\,rs\,\text{tt}\,] \\
 & \qquad \text{ff}
\end{array}
\qquad
\begin{array}{ll}
\text{coop}\,(r :: rs) = & \\
\quad \textbf{return}\,() & \mapsto r\,rs\,() \\
\quad \langle \text{yield}\,() \to r' \rangle & \mapsto r\,(rs +\!\!+ [r'])\,() \\
\quad \langle \text{ufork}\,() \to r' \rangle & \mapsto r'\,(r :: rs +\!\!+ [\lambda rs\,().r'\,rs\,\text{tt}\,]) \\
 & \qquad \text{ff}
\end{array}$$

Example 5: lightweight threads with UNIX-style fork

Types

$$\mathsf{Thread}\ E = 1 \to \mathsf{CoU}\ E!1 \qquad\qquad \mathsf{Res}\ E = \mathsf{List}\ (\mathsf{Res}\ E) \to 1 \to 1!E$$

Parameterised handler

$$\mathsf{coop} : \mathsf{List}\ (\mathsf{Res}\ E) \to \mathsf{CoU}\ E!1 \Rightarrow 1!E$$

$$\begin{array}{ll}
\mathsf{coop}\,([]) = & \\
\quad \mathbf{return}\,() & \mapsto ()\\
\quad \langle \mathsf{yield}\,() \to r' \rangle & \mapsto r'\,[]\,()\\
\quad \langle \mathsf{ufork}\,() \to r' \rangle & \mapsto r'\,[\lambda rs\,().r'\,rs\,\mathsf{tt}\,]\\
 & \qquad\quad \mathsf{ff}
\end{array}$$

$$\begin{array}{ll}
\mathsf{coop}\,(r :: rs) = & \\
\quad \mathbf{return}\,() & \mapsto r\;rs\,()\\
\quad \langle \mathsf{yield}\,() \to r' \rangle & \mapsto r\,(rs +\!\!+ [r'])\,()\\
\quad \langle \mathsf{ufork}\,() \to r' \rangle & \mapsto r'\,(r :: rs +\!\!+ [\lambda rs\,().r'\,rs\,\mathsf{tt}\,])\\
 & \qquad\quad \mathsf{ff}
\end{array}$$

$$\mathsf{cooperate}\,[\mathsf{main}] \Longrightarrow ()$$

M1 M2 M3 A1 B1 A2 B2

Example 6: lightweight threads with higher-order fork

Effect signature — recursive effect signature

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

Example 6: lightweight threads with higher-order fork

Effect signature — recursive effect signature

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

A single cooperative program

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

Example 6: lightweight threads with higher-order fork

Types

$$\text{Thread } E = 1 \to 1!\text{Co } E \qquad \text{Res } E = \text{List } (\text{Res } E) \to 1 \to 1!E$$

Parameterised handler

$$\text{coop} : \text{List } (\text{Res } E) \to 1!\text{Co } E \Rightarrow 1!E$$

$$\begin{array}{lll}
\text{coop}\,([]) = & & \\
\quad \mathbf{return}\,() & \mapsto & () \\
\quad \langle \text{yield}\,() \to r' \rangle & \mapsto & r'\,[]\,() \\
\quad \langle \text{fork}\, t \to r' \rangle & \mapsto & \text{lift}\, t\,[r']\,()
\end{array}
\qquad
\begin{array}{lll}
\text{coop}\,(r :: rs) = & & \\
\quad \mathbf{return}\,() & \mapsto & r\, rs\,() \\
\quad \langle \text{yield}\,() \to r' \rangle & \mapsto & r\,(rs +\!\!+ [r'])\,() \\
\quad \langle \text{fork}\, t \to r' \rangle & \mapsto & \text{lift}\, t\,(r :: rs +\!\!+ [r'])\,()
\end{array}$$

Example 6: lightweight threads with higher-order fork

Types

$$\text{Thread } E = 1 \to 1!\text{Co } E \qquad\qquad \text{Res } E = \text{List } (\text{Res } E) \to 1 \to 1!E$$

Parameterised handler

$$\text{coop} : \text{List } (\text{Res } E) \to 1!\text{Co } E \Rightarrow 1!E$$

$$\begin{array}{lll} \text{coop } ([\,]) = & & \\ \quad \textbf{return } () & \mapsto & () \\ \quad \langle \text{yield } () \to r' \rangle & \mapsto & r' \, [\,] \, () \\ \quad \langle \text{fork } t \to r' \rangle & \mapsto & \text{lift } t \, [r'] \, () \end{array} \qquad \begin{array}{lll} \text{coop } (r :: rs) = & & \\ \quad \textbf{return } () & \mapsto & r \; rs \, () \\ \quad \langle \text{yield } () \to r' \rangle & \mapsto & r \, (rs +\!\!+ [r']) \, () \\ \quad \langle \text{fork } t \to r' \rangle & \mapsto & \text{lift } t \, (r :: rs +\!\!+ [r']) \, () \end{array}$$

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

M1 A1 M2 B1 A2 M3 B2

Example 6: lightweight threads with higher-order fork

Types

$$\text{Thread } E = 1 \to 1!\text{Co } E \qquad\qquad \text{Res } E = \text{List } (\text{Res } E) \to 1 \to 1!E$$

Parameterised handler

$$\text{coop} : \text{List } (\text{Res } E) \to 1!\text{Co } E \Rightarrow 1!E$$

$$\begin{array}{ll} \text{coop}\,([]) = & \text{coop}\,(r :: rs) = \\ \quad \textbf{return}\,() \mapsto () & \quad \textbf{return}\,() \mapsto r\; rs\,() \\ \quad \langle \text{yield}\,() \to r' \rangle \mapsto r'\,[]\,() & \quad \langle \text{yield}\,() \to r' \rangle \mapsto r\,(rs +\!\!+ [r'])\,() \\ \quad \langle \text{fork}\; t \to r' \rangle \mapsto r'\,[\text{lift}\; t]\,() & \quad \langle \text{fork}\; t \to r' \rangle \mapsto r'\,(r :: rs +\!\!+ [\text{lift}\; t])\,() \end{array}$$

Example 6: lightweight threads with higher-order fork

Types

$$\text{Thread } E = 1 \to 1!\text{Co } E \qquad \text{Res } E = \text{List } (\text{Res } E) \to 1 \to 1!E$$

Parameterised handler

$$\text{coop} : \text{List } (\text{Res } E) \to 1!\text{Co } E \Rightarrow 1!E$$

$$\begin{array}{ll}
\text{coop } ([]) = & \\
\quad \textbf{return } () & \mapsto () \\
\quad \langle \text{yield } () \to r' \rangle & \mapsto r' \, [] \, () \\
\quad \langle \text{fork } t \to r' \rangle & \mapsto r' \, [\text{lift } t] \, ()
\end{array}
\qquad
\begin{array}{ll}
\text{coop } (r :: rs) = & \\
\quad \textbf{return } () & \mapsto r \, rs \, () \\
\quad \langle \text{yield } () \to r' \rangle & \mapsto r \, (rs +\!\!+ [r']) \, () \\
\quad \langle \text{fork } t \to r' \rangle & \mapsto r' \, (r :: rs +\!\!+ [\text{lift } t]) \, ()
\end{array}$$

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

M1 M2 M3 A1 B1 A2 B2

Example 6: pipes

Effect signatures

$$\text{Sender} = \{\text{send} : \text{Nat} \twoheadrightarrow 1\} \qquad \text{Receiver} = \{\text{receive} : 1 \twoheadrightarrow \text{Nat}\}$$

Example 6: pipes

Effect signatures

$$\mathsf{Sender} = \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\} \qquad \mathsf{Receiver} = \{\mathsf{receive} : 1 \twoheadrightarrow \mathsf{Nat}\}$$

A producer and a consumer

$$\begin{array}{ll} \mathsf{nats} : \mathsf{Nat} \to 1!(E \uplus \mathsf{Sender}) & \mathsf{grabANat} : 1 \to \mathsf{Nat}!(E \uplus \mathsf{Receiver}) \\ \mathsf{nats}\, n = \mathsf{send}\ n; \mathsf{nats}\,(n+1) & \mathsf{grabANat}\,() = \mathsf{receive}\,() \end{array}$$

Example 6: pipes

Effect signatures

$$\mathsf{Sender} = \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\} \qquad \mathsf{Receiver} = \{\mathsf{receive} : 1 \twoheadrightarrow \mathsf{Nat}\}$$

A producer and a consumer

$$\begin{array}{ll} \mathsf{nats} : \mathsf{Nat} \to 1!(E \uplus \mathsf{Sender}) & \mathsf{grabANat} : 1 \to \mathsf{Nat}!(E \uplus \mathsf{Receiver}) \\ \mathsf{nats}\, n = \mathsf{send}\, n; \mathsf{nats}\,(n+1) & \mathsf{grabANat}\,() = \mathsf{receive}\,() \end{array}$$

Pipes and copipes as shallow handlers

$$\begin{array}{l} \mathsf{pipe}\, p\, c = \mathbf{handle}\; c\,()\; \mathbf{with} \\ \qquad \mathbf{return}\, x \mapsto x \\ \qquad \langle \mathsf{receive}\,() \to r\rangle \mapsto \mathsf{copipe}\, r\, p \end{array} \qquad \begin{array}{l} \mathsf{copipe}\, c\, p = \mathbf{handle}\; p\,()\; \mathbf{with} \\ \qquad \mathbf{return}\, x \mapsto x \\ \qquad \langle \mathsf{send}\, n \to r\rangle \mapsto \mathsf{pipe}\, r\,(\lambda().c\, n) \end{array}$$

Example 6: pipes

Effect signatures

$$\text{Sender} = \{\text{send} : \text{Nat} \twoheadrightarrow 1\} \qquad \text{Receiver} = \{\text{receive} : 1 \twoheadrightarrow \text{Nat}\}$$

A producer and a consumer

$$\begin{array}{ll} \text{nats} : \text{Nat} \to 1!(E \uplus \text{Sender}) & \text{grabANat} : 1 \to \text{Nat}!(E \uplus \text{Receiver}) \\ \text{nats}\, n = \text{send}\, n; \text{nats}\,(n+1) & \text{grabANat}\,() = \text{receive}\,() \end{array}$$

Pipes and copipes as shallow handlers

$$\begin{array}{l} \text{pipe}\, p\, c = \textbf{handle}\; c\,()\; \textbf{with} \\ \qquad \textbf{return}\, x \mapsto x \\ \qquad \langle \text{receive}\,() \to r \rangle \mapsto \text{copipe}\, r\, p \end{array} \qquad \begin{array}{l} \text{copipe}\, c\, p = \textbf{handle}\; p\,()\; \textbf{with} \\ \qquad \textbf{return}\, x \mapsto x \\ \qquad \langle \text{send}\, n \to r \rangle \mapsto \text{pipe}\, r\, (\lambda().c\, n) \end{array}$$

$$\begin{array}{rl} \text{pipe}\,(\lambda().\text{nats}\,0)\,\text{grabANat} & \rightsquigarrow^{+} \text{copipe}\,(\lambda x.x)\,(\lambda().\text{nats}\,0) \\ & \rightsquigarrow^{+} \text{pipe}\,(\lambda().\text{nats}\,1)\,(\lambda().0) \rightsquigarrow^{+} 0 \end{array}$$

Example 6: pipes

Effect signatures

$$\text{Sender} = \{\text{send} : \text{Nat} \twoheadrightarrow 1\} \qquad \text{Receiver} = \{\text{receive} : 1 \twoheadrightarrow \text{Nat}\}$$

A producer and a consumer

$$\begin{array}{ll} \text{nats} : \text{Nat} \to 1!(E \uplus \text{Sender}) & \text{grabANat} : 1 \to \text{Nat}!(E \uplus \text{Receiver}) \\ \text{nats}\, n = \text{send}\, n; \text{nats}\,(n+1) & \text{grabANat}\,() = \text{receive}\,() \end{array}$$

Pipes and copipes as shallow handlers

$$\begin{array}{ll} \text{pipe}\, p\, c = \textbf{handle}\ c\,()\ \textbf{with} & \text{copipe}\, c\, p = \textbf{handle}\ p\,()\ \textbf{with} \\ \quad \textbf{return}\, x \mapsto x & \quad \textbf{return}\, x \mapsto x \\ \quad \langle \text{receive}\,() \to r \rangle \mapsto \text{copipe}\, r\, p & \quad \langle \text{send}\, n \to r \rangle \mapsto \text{pipe}\, r\, (\lambda().c\, n) \end{array}$$

$$\begin{aligned} \text{pipe}\,(\lambda().\text{nats}\,0)\,\text{grabANat} &\rightsquigarrow^{+} \text{copipe}\,(\lambda x.x)\,(\lambda().\text{nats}\,0) \\ &\rightsquigarrow^{+} \text{pipe}\,(\lambda().\text{nats}\,1)\,(\lambda().0) \rightsquigarrow^{+} 0 \end{aligned}$$

Exercise: implement pipes using parameterised handlers

Built-in effects

Console I/O

$$\begin{aligned}\text{Console} = \{\textsf{inch} &: 1 \twoheadrightarrow \text{char}\\ \textsf{ouch} &: \text{char} \twoheadrightarrow 1\}\end{aligned}$$

$$\text{print}\, s = \text{map}\,(\lambda c.\textsf{ouch}\; c)\; s; ()$$

Generative state

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

Example 7: 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 7: actors

Process ids

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

Effect signature

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

An actor chain

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

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

Example 7: actors — via lightweight threads

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

Example 7: actors — via lightweight threads

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

$$\mathsf{cooperate}\ [\mathbf{handle}\ \mathsf{chain}\ 64\ \mathbf{with}\ \mathsf{act}\ (\mathsf{new}\ [])] \Longrightarrow ()$$

....ping!

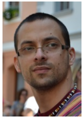
Effect handler oriented programming languages

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
Multicore OCaml	https://github.com/ocamllabs/ocaml-multicore/wiki

Resources

The EHOP project website: https://effect-handlers.org/

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