

Effect handler oriented programming

Sam Lindley

The University of Edinburgh

SPLV 2022

What is an effect?

Effects

Programs as black boxes (Church-Turing model)?



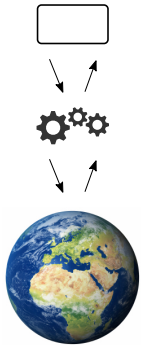
Effects

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**

Growing industrial interest (c.f. resumable exceptions, monads, delimited control)

GitHub

`semantic`

Code analysis library (> 25 million repositories)




React

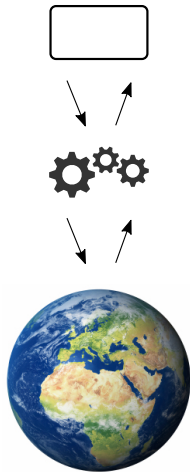
JavaScript UI library (> 2 million websites)



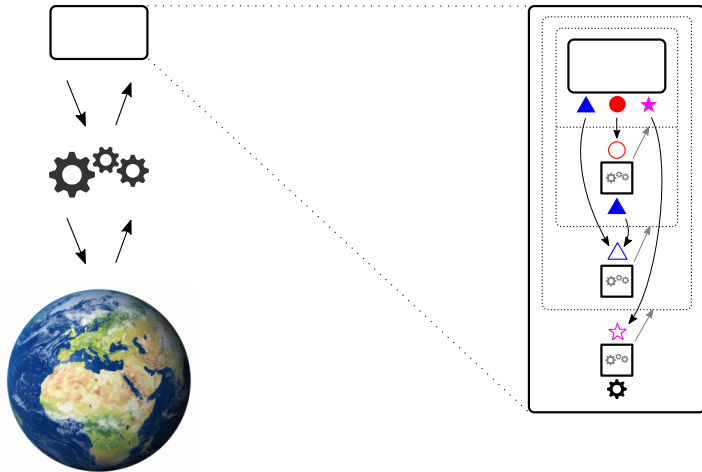

Pyro

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

{choose : 1 \rightarrow Bool, fail : 1 \rightarrow 0}

Example 1: choice and failure

Effect signature

$$\{\text{choose} : 1 \rightarrow \text{Bool}, \text{fail} : 1 \rightarrow 0\}$$

Drunk coin tossing

$\text{toss} : 1 \rightarrow \text{Toss!}(E \uplus \{\text{choose} : 1 \rightarrow \text{Bool}\})$
 $\text{toss} () = \text{if } \text{choose} () \text{ then Heads else Tails}$

$\text{drunkToss} : 1 \rightarrow \text{Toss!}(E \uplus \{\text{choose} : 1 \rightarrow \text{Bool}, \text{fail} : 1 \rightarrow 0\})$
 $\text{drunkToss} () = \text{if } \text{choose} () \text{ then}$
 $\text{if } \text{choose} () \text{ then Heads else Tails}$
 else
 $\text{absurd}(\text{fail}())$

$\text{drunkTosses} : \text{Nat} \rightarrow \text{List Toss!}(E \uplus \{\text{choose} : 1 \rightarrow \text{Bool}, \text{fail} : 1 \rightarrow 0\})$
 $\text{drunkTosses } n = \text{if } n = 0 \text{ then } []$
 $\text{else } \text{drunkToss} () :: \text{drunkTosses} (n - 1)$

Example 1: choice and failure

Handlers

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

$\text{maybeFail} =$ — exception handler

return $x \mapsto \text{Just } x$

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

Example 1: choice and failure

Handlers

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

$\text{maybeFail} =$ — exception handler

return $x \mapsto \text{Just } x$

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

handle 42 **with** maybeFail $\Rightarrow \text{Just } 42$

handle (**absurd** ($\text{fail} ()$)) **with** maybeFail $\Rightarrow \text{Nothing}$

Example 1: choice and failure

Handlers

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

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

return $x \mapsto \text{Just } x$

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

handle 42

with $\text{maybeFail} \Rightarrow \text{Just } 42$

handle (**absurd** ($\text{fail} ()$)) **with** $\text{maybeFail} \Rightarrow \text{Nothing}$

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

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

return $x \mapsto x$

$\langle \text{choose} () \rightarrow r \rangle \mapsto r \text{ tt}$

Example 1: choice and failure

Handlers

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

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

return $x \mapsto \text{Just } x$

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

handle 42 **with** $\text{maybeFail} \Rightarrow \text{Just } 42$

handle (**absurd** ($\text{fail} ()$)) **with** $\text{maybeFail} \Rightarrow \text{Nothing}$

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

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

return $x \mapsto x$

$\langle \text{choose} () \rightarrow r \rangle \mapsto r \text{ tt}$

handle 42 **with** $\text{trueChoice} \Rightarrow 42$

handle $\text{toss} ()$ **with** $\text{trueChoice} \Rightarrow \text{Heads}$

Example 1: choice and failure

Handlers

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

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

$\text{return } x \mapsto \text{Just } x$

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

$\text{handle } 42 \text{ with maybeFail} \Rightarrow \text{Just } 42$

$\text{handle } (\text{absurd } (\text{fail} ())) \text{ with maybeFail} \Rightarrow \text{Nothing}$

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

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

$\text{return } x \mapsto x$

$\langle \text{choose} () \rightarrow r \rangle \mapsto r \text{ tt}$

$\text{handle } 42 \text{ with trueChoice} \Rightarrow 42$

$\text{handle } \text{toss} () \text{ with trueChoice} \Rightarrow \text{Heads}$

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

$\text{allChoices} =$ — **non-linear handler**

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

$\langle \text{choose} () \rightarrow r \rangle \mapsto r \text{ tt} ++ r \text{ ff}$

Example 1: choice and failure

Handlers

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

$\text{maybeFail} =$ — exception handler

$\text{return } x \mapsto \text{Just } x$

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

$\text{handle } 42 \text{ with maybeFail} \Rightarrow \text{Just } 42$

$\text{handle } (\text{absurd } (\text{fail} ())) \text{ with maybeFail} \Rightarrow \text{Nothing}$

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

$\text{trueChoice} =$ — linear handler

$\text{return } x \mapsto x$

$\langle \text{choose} () \rightarrow r \rangle \mapsto r \text{ tt}$

$\text{handle } 42 \text{ with trueChoice} \Rightarrow 42$

$\text{handle } \text{toss} () \text{ with trueChoice} \Rightarrow \text{Heads}$

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

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

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

$\langle \text{choose} () \rightarrow r \rangle \mapsto r \text{ tt} ++ r \text{ ff}$

$\text{handle } 42 \text{ with allChoices} \Rightarrow [42]$

$\text{handle } \text{toss} () \text{ with allChoices} \Rightarrow [\text{Heads}, \text{Tails}]$

Example 1: choice and failure

Handler composition

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

Example 1: choice and failure

Handler composition

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

Example 1: choice and failure

Handler composition

handle (**handle** drunkTosses 2 **with** maybeFail) **with** allChoices : List (Maybe (List Toss)) \implies
[Just [Heads, Heads], Just [Heads, Tails], Nothing,
Just [Tails, Heads], Just [Tails, Tails], Nothing,
Nothing]

Example 1: choice and failure

Handler composition

handle (**handle** drunkTosses 2 **with** maybeFail) **with** allChoices : List (Maybe (List Toss)) \implies
[Just [Heads, Heads], Just [Heads, Tails], Nothing,
Just [Tails, Heads], Just [Tails, Tails], Nothing,
Nothing]

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

Example 1: choice and failure

Handler composition

handle (**handle** drunkTosses 2 **with** maybeFail) **with** allChoices : List (Maybe (List Toss)) \implies
[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)) \implies

Example 1: choice and failure

Handler composition

handle (handle drunkTosses 2 with maybeFail) with allChoices : List (Maybe (List Toss)) \implies
[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)) \implies
Nothing

Operational semantics (deep handlers)

Reduction rules

let $x = V$ **in** $N \rightsquigarrow N[V/x]$

handle V **with** $H \rightsquigarrow N[V/x]$

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

where

$$\begin{aligned} H = \text{return } x & \quad \mapsto N \\ \langle \text{op}_1 p \rightarrow r \rangle & \quad \mapsto N_{\text{op}_1} \\ & \quad \dots \\ \langle \text{op}_k p \rightarrow r \rangle & \quad \mapsto N_{\text{op}_k} \end{aligned}$$

Evaluation contexts

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

Typing rules (deep handlers)

Effects

$$E ::= \emptyset \mid E \uplus \{\text{op} : A \rightarrow B\}$$

Computations

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

Operations

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

Handlers

$$\frac{\Gamma \vdash M : C \quad \Gamma \vdash H : C \Rightarrow D}{\Gamma \vdash \text{handle } M \text{ with } H : D}$$

$$\frac{\Gamma, x : A \vdash N : D \quad [\text{op}_i : A_i \rightarrow B_i \in E]_i \quad [\Gamma, p : A_i, r : B_i \rightarrow D \vdash N_i : D]_i}{\Gamma \vdash \text{return } x \mapsto N \quad (\langle \text{op}_i p \rightarrow r \rangle \mapsto N_i)_i : A!E \Rightarrow D}$$

Example 2: generators

Effect signature

{send : Nat \rightarrow 1}

Example 2: generators

Effect signature

$$\{\text{send} : \text{Nat} \rightarrow 1\}$$

A simple generator

$$\begin{aligned} \text{nats} &: \text{Nat} \rightarrow 1!(E \uplus \{\text{send} : \text{Nat} \rightarrow 1\}) \\ \text{nats } n &= \text{send } n; \text{nats } (n + 1) \end{aligned}$$

Example 2: generators

Effect signature

$$\{\text{send} : \text{Nat} \rightarrow 1\}$$

A simple generator

$$\begin{aligned} \text{nats} &: \text{Nat} \rightarrow 1!(E \uplus \{\text{send} : \text{Nat} \rightarrow 1\}) \\ \text{nats } n &= \text{send } n; \text{nats } (n + 1) \end{aligned}$$

Handler — a function that returns a handler

$$\begin{aligned} \text{until} &: \text{Nat} \rightarrow 1!(E \uplus \{\text{send} : \text{Nat} \rightarrow 1\}) \Rightarrow \text{List Nat!}E \\ \text{until } stop &= \\ &\text{return } () \quad \mapsto [] \\ &\langle \text{send } n \rightarrow r \rangle \mapsto \text{if } n < stop \text{ then } n :: r () \\ &\quad \text{else } [] \end{aligned}$$

Example 2: generators

Effect signature

$$\{\text{send} : \text{Nat} \rightarrow 1\}$$

A simple generator

$$\begin{aligned} \text{nats} &: \text{Nat} \rightarrow 1!(E \uplus \{\text{send} : \text{Nat} \rightarrow 1\}) \\ \text{nats } n &= \text{send } n; \text{nats } (n + 1) \end{aligned}$$

Handler — a function that returns a handler

$$\text{until} : \text{Nat} \rightarrow 1!(E \uplus \{\text{send} : \text{Nat} \rightarrow 1\}) \Rightarrow \text{List Nat!}E$$

until *stop* =

$$\text{return } () \quad \mapsto []$$
$$\langle \text{send } n \rightarrow r \rangle \mapsto \text{if } n < \text{stop} \text{ then } n :: r () \\ \text{else } []$$
$$\text{handle nats } 0 \text{ with until } 8 \implies [0, 1, 2, 3, 4, 5, 6, 7]$$

Example 3: lightweight threads

Effect signature

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

Example 3: lightweight threads

Effect signature

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

Two cooperative lightweight threads

`tA () = print ("A1 "); yield (); print ("A2 ")`

`tB () = print ("B1 "); yield (); print ("B2 ")`

Example 3.1: lightweight threads (deep handlers)

Types

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

Res $E = 1 \rightarrow \text{List} (\text{Res } E) \rightarrow 1!E$

Handler

$\text{coop} : 1!(\text{Thread } E) \Rightarrow (\text{List} (\text{Res } E) \rightarrow 1!E)$

$\text{coop} = \text{return } () \quad \mapsto \lambda rs. \text{case } rs \text{ of } [] \quad \mapsto ()$
 $\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad (r :: rs) \mapsto r () rs$
 $\langle \text{yield } () \rightarrow s \rangle \mapsto \lambda rs. \text{case } rs \text{ of } [] \quad \mapsto s () []$
 $\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad (r :: rs) \mapsto r () (rs ++ [s])$

$\text{lift} : \text{Thread } E \rightarrow \text{Res } E$

$\text{lift } t = \lambda(). \text{handle } t() \text{ with } \text{coop}$

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

$\text{cooperate } ts = \text{lift id } () (\text{map lift } ts)$

Example 3.1: lightweight threads (deep handlers)

Types

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

Res $E = 1 \rightarrow \text{List} (\text{Res } E) \rightarrow 1!E$

Handler

$\text{coop} : 1!(\text{Thread } E) \Rightarrow (\text{List} (\text{Res } E) \rightarrow 1!E)$

$\text{coop} = \mathbf{return} () \quad \mapsto \lambda rs. \mathbf{case} \text{ } rs \text{ of } [] \quad \mapsto ()$
 $\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad (r :: rs) \mapsto r () \text{ } rs$
 $\langle \text{yield} () \rightarrow s \rangle \mapsto \lambda rs. \mathbf{case} \text{ } rs \text{ of } [] \quad \mapsto s () []$
 $\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad (r :: rs) \mapsto r () (rs ++ [s])$

$\text{lift} : \text{Thread } E \rightarrow \text{Res } E$

$\text{lift } t = \lambda(). \mathbf{handle} \text{ } t() \text{ with } \text{coop}$

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

$\text{cooperate } ts = \text{lift id } () (\text{map lift } ts)$

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

A1 B1 A2 B2

Example 3.2: lightweight threads (parameterised handlers)

Types

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

Res $E = \text{List} (\text{Res } E) \rightarrow 1 \rightarrow 1!E$

Handler — parameterised handler

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

$\text{coop} ([]) =$
 $\text{return } () \quad \mapsto ()$
 $\langle \text{yield } () \rightarrow r' \rangle \mapsto r' [] ()$

$\text{coop} (r :: rs) =$
 $\text{return } () \quad \mapsto r rs ()$
 $\langle \text{yield } () \rightarrow r' \rangle \mapsto r (rs ++ [r']) ()$

$\text{lift} : \text{Thread } E \rightarrow \text{Res } E$

$\text{lift } t = \lambda rs (). \text{handle } t() \text{ with } \text{coop } rs$

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

$\text{cooperate } ts = \text{lift id } (\text{map lift } ts) ()$

Example 3.2: lightweight threads (parameterised handlers)

Types

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

Res $E = \text{List} (\text{Res } E) \rightarrow 1 \rightarrow 1!E$

Handler — parameterised handler

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

$\text{coop} ([]) =$

$\text{return } () \quad \mapsto ()$
 $\langle \text{yield } () \rightarrow r' \rangle \mapsto r' [] ()$

$\text{coop} (r :: rs) =$

$\text{return } () \quad \mapsto r rs ()$
 $\langle \text{yield } () \rightarrow r' \rangle \mapsto r (rs ++ [r']) ()$

$\text{lift} : \text{Thread } E \rightarrow \text{Res } E$

$\text{lift } t = \lambda rs (). \text{handle } t() \text{ with } \text{coop } rs$

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

$\text{cooperate } ts = \text{lift id } (\text{map lift } ts) ()$

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

A1 B1 A2 B2

Parameterised effect handlers

$$\frac{\Gamma \vdash M : C \quad \Gamma \vdash V : P \quad \Gamma \vdash H : P \rightarrow C \Rightarrow D}{\Gamma \vdash \mathbf{handle} M \mathbf{with} H V : D}$$

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

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

Parameterised effect handlers

$$\frac{\Gamma \vdash M : C \quad \Gamma \vdash V : P \quad \Gamma \vdash H : P \rightarrow C \Rightarrow D}{\Gamma \vdash \mathbf{handle} M \mathbf{with} H V : D}$$

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

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

Exercise: express parameterised handlers as deep handlers

Example 3.3: lightweight threads (shallow handlers)

Types

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

Handler — shallow handler

cooperate : List (Thread E) $\rightarrow 1!E$

cooperate [] = ()

cooperate ($r :: rs$) = **handle** $r()$ **with**

return () \mapsto cooperate (rs)

$\langle \text{yield} () \rightarrow s \rangle \mapsto$ cooperate ($rs \uparrow\uparrow [s]$)

Example 3.3: lightweight threads (shallow handlers)

Types

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

Res $E = \text{Thread } E$

Handler — shallow handler

cooperate : List (Thread E) $\rightarrow 1!E$

cooperate [] = ()

cooperate ($r :: rs$) = **handle** $r()$ **with**

return () \mapsto cooperate (rs)

$\langle \text{yield} () \rightarrow s \rangle \mapsto$ cooperate ($rs ++ [s]$)

cooperate [tA, tB] \Longrightarrow ()

A1 B1 A2 B2

Deep effect handlers

$$\frac{\Gamma, x : A \vdash N : D \quad [\text{op}_i : A_i \twoheadrightarrow B_i \in E]_i \quad [\Gamma, p : A_i, r : B_i \rightarrow D \vdash N_i : D]_i}{\Gamma \vdash \text{return } x \mapsto N \quad (\langle \text{op}_i \ p \rightarrow r \rangle \mapsto N_i)_i : A!E \Rightarrow D}$$

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

Deep effect handlers

$$\frac{\Gamma, x : A \vdash N : D \quad [\text{op}_i : A_i \twoheadrightarrow B_i \in E]_i \quad [\Gamma, p : A_i, r : B_i \rightarrow D \vdash N_i : D]_i}{\Gamma \vdash \text{return } x \mapsto N \quad (\langle \text{op}_i \ p \rightarrow r \rangle \mapsto N_i)_i : A!E \Rightarrow D}$$

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

The body of the resumption r reinvokes the handler

Deep effect handlers

$$\frac{\Gamma, x : A \vdash N : D \quad [\text{op}_i : A_i \twoheadrightarrow B_i \in E]_i \quad [\Gamma, p : A_i, r : B_i \rightarrow D \vdash N_i : D]_i}{\Gamma \vdash \mathbf{return} \ x \mapsto N \quad (\langle \text{op}_i \ p \rightarrow r \rangle \mapsto N_i)_i : A!E \Rightarrow D}$$

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

The body of the resumption r reinvokes the handler

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

Shallow effect handlers

$$\frac{\Gamma, x : A \vdash N : D \quad [\text{op}_i : A_i \Rightarrow B_i \in E]_i \quad [\Gamma, p : A_i, r : B_i \rightarrow A!E \vdash N_i : D]_i}{\Gamma \vdash \text{return } x \mapsto N \quad (\langle \text{op}_i p \rightarrow r \rangle \mapsto N_i)_i : A!E \Rightarrow D}$$

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

Shallow effect handlers

$$\frac{\Gamma, x : A \vdash N : D \quad [\text{op}_i : A_i \Rightarrow B_i \in E]_i \quad [\Gamma, p : A_i, r : B_i \rightarrow A!E \vdash N_i : D]_i}{\Gamma \vdash \text{return } x \mapsto N \quad (\langle \text{op}_i p \rightarrow r \rangle \mapsto N_i)_i : A!E \Rightarrow D}$$

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

The body of the resumption r does not reinvoke the handler

Shallow effect handlers

$$\frac{\Gamma, x : A \vdash N : D \quad [\text{op}_i : A_i \Rightarrow B_i \in E]_i \quad [\Gamma, p : A_i, r : B_i \rightarrow A!E \vdash N_i : D]_i}{\Gamma \vdash \text{return } x \mapsto N \quad (\langle \text{op}_i p \rightarrow r \rangle \mapsto N_i)_i : A!E \Rightarrow D}$$

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

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

$$\frac{\Gamma, x : A \vdash N : D \quad [\text{op}_i : A_i \Rightarrow B_i \in E]_i \quad [\Gamma, p : A_i, r : B_i \rightarrow A!E \vdash N_i : D]_i}{\Gamma \vdash \text{return } x \mapsto N \quad (\langle \text{op}_i p \rightarrow r \rangle \mapsto N_i)_i : A!E \Rightarrow D}$$

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

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

$$\text{CoU } E = E \uplus \{\text{yield} : 1 \rightarrow 1, \text{ufork} : 1 \rightarrow \text{Bool}\}$$

Example 5: lightweight threads with UNIX-style fork

Effect signature

$$\text{CoU } E = E \uplus \{\text{yield} : 1 \rightarrow 1, \text{ufork} : 1 \rightarrow \text{Bool}\}$$

A single cooperative program

`main : 1 → CoU E!1`

```
main () = print "M1 "; if ufork () then print "A1 "; yield (); print "A2 "  
         else print "M2 "; if ufork () then print "B1 "; yield (); print "B2 " else print "M3 "
```

Example 5: lightweight threads with UNIX-style fork

Types

Thread $E = 1 \rightarrow \text{CoU } E!1$

Res $E = \text{List (Res } E) \rightarrow 1 \rightarrow 1!E$

Parameterised handler

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

$\text{coop} ([]) =$

return () \mapsto ()

$\langle \text{yield} () \rightarrow r' \rangle \mapsto r' [] ()$

$\langle \text{ufork} () \rightarrow r' \rangle \mapsto r' [\lambda rs ().r' rs \text{ ff}]$
tt

$\text{coop} (r :: rs) =$

return () $\mapsto r rs ()$

$\langle \text{yield} () \rightarrow r' \rangle \mapsto r (rs ++ [r']) ()$

$\langle \text{ufork} () \rightarrow r' \rangle \mapsto r' (r :: rs ++ [\lambda rs ().r' rs \text{ ff}])$
tt

Example 5: lightweight threads with UNIX-style fork

Types

Thread $E = 1 \rightarrow \text{CoU } E!1$

Res $E = \text{List (Res } E) \rightarrow 1 \rightarrow 1!E$

Parameterised handler

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

$\text{coop} ([])$ =

return () \mapsto ()

$\langle \text{yield} () \rightarrow r' \rangle \mapsto r' [] ()$

$\langle \text{ufork} () \rightarrow r' \rangle \mapsto r' [\lambda rs ().r' rs \text{ ff}]$
tt

$\text{coop} (r :: rs)$ =

return () $\mapsto r rs ()$

$\langle \text{yield} () \rightarrow r' \rangle \mapsto r (rs ++ [r']) ()$

$\langle \text{ufork} () \rightarrow r' \rangle \mapsto r' (r :: rs ++ [\lambda rs ().r' rs \text{ ff}])$
tt

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

M1 A1 M2 B1 A2 M3 B2

Example 5: lightweight threads with UNIX-style fork

Types

Thread $E = 1 \rightarrow \text{CoU } E!1$

Res $E = \text{List (Res } E) \rightarrow 1 \rightarrow 1!E$

Parameterised handler

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

$\text{coop} ([]) =$

return () $\mapsto ()$

$\langle \text{yield } () \rightarrow r' \rangle \mapsto r' [] ()$

$\langle \text{ufork } () \rightarrow r' \rangle \mapsto r' [\lambda rs ().r' rs \text{ tt}]$
ff

$\text{coop} (r :: rs) =$

return () $\mapsto r rs ()$

$\langle \text{yield } () \rightarrow r' \rangle \mapsto r (rs ++ [r']) ()$

$\langle \text{ufork } () \rightarrow r' \rangle \mapsto r' (r :: rs ++ [\lambda rs ().r' rs \text{ tt}])$
ff

Example 5: lightweight threads with UNIX-style fork

Types

Thread $E = 1 \rightarrow \text{CoU } E!1$

Res $E = \text{List } (\text{Res } E) \rightarrow 1 \rightarrow 1!E$

Parameterised handler

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

$\text{coop} ([]) =$

return () \mapsto ()

$\langle \text{yield } () \rightarrow r' \rangle \mapsto r' [] ()$

$\langle \text{ufork } () \rightarrow r' \rangle \mapsto r' [\lambda rs ().r' rs tt]$
ff

$\text{coop} (r :: rs) =$

return () $\mapsto r rs ()$

$\langle \text{yield } () \rightarrow r' \rangle \mapsto r (rs ++ [r']) ()$

$\langle \text{ufork } () \rightarrow r' \rangle \mapsto r' (r :: rs ++ [\lambda rs ().r' rs tt])$
ff

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

M1 M2 M3 A1 B1 A2 B2

Example 6: lightweight threads with higher-order fork

Effect signature — recursive effect signature

$$\text{Co } E = E \uplus \{\text{yield} : 1 \twoheadrightarrow 1, \text{fork} : (1 \rightarrow 1! \text{Co } E) \twoheadrightarrow 1\}$$

Example 6: lightweight threads with higher-order fork

Effect signature — recursive effect signature

$$\text{Co } E = E \uplus \{\text{yield} : 1 \rightarrow 1, \text{fork} : (1 \rightarrow 1! \text{Co } E) \rightarrow 1\}$$

A single cooperative program

```
main : 1 → 1!Co E
```

```
main () = print "M1 "; fork (λ().print "A1 "; yield (); print "A2 ");  
         print "M2 "; fork (λ().print "B1 "; yield (); print "B2 "); print "M3 "
```

Example 6: lightweight threads with higher-order fork

Types

Thread $E = 1 \rightarrow 1!Co E$

Res $E = List (Res E) \rightarrow 1 \rightarrow 1!E$

Parameterised handler

$coop : List (Res E) \rightarrow 1!Co E \Rightarrow 1!E$

$coop ([]) =$

return () \mapsto ()

$\langle yield () \rightarrow r' \rangle \mapsto r' [] ()$

$\langle fork t \rightarrow r' \rangle \mapsto lift t [r'] ()$

$coop (r :: rs) =$

return () $\mapsto r rs ()$

$\langle yield () \rightarrow r' \rangle \mapsto r (rs ++ [r']) ()$

$\langle fork t \rightarrow r' \rangle \mapsto lift t (r :: rs ++ [r']) ()$

Example 6: lightweight threads with higher-order fork

Types

Thread $E = 1 \rightarrow 1!Co E$

Res $E = List (Res E) \rightarrow 1 \rightarrow 1!E$

Parameterised handler

$coop : List (Res E) \rightarrow 1!Co E \Rightarrow 1!E$

$coop ([]) =$

return () \mapsto ()

$\langle yield () \rightarrow r' \rangle \mapsto r' [] ()$

$\langle fork t \rightarrow r' \rangle \mapsto lift t [r'] ()$

$coop (r :: rs) =$

return () $\mapsto r rs ()$

$\langle yield () \rightarrow r' \rangle \mapsto r (rs ++ [r']) ()$

$\langle fork t \rightarrow r' \rangle \mapsto lift t (r :: rs ++ [r']) ()$

$cooperate [main] \Longrightarrow ()$

M1 A1 M2 B1 A2 M3 B2

Example 6: lightweight threads with higher-order fork

Types

Thread $E = 1 \rightarrow 1!Co E$

Res $E = List (Res E) \rightarrow 1 \rightarrow 1!E$

Parameterised handler

$coop : List (Res E) \rightarrow 1!Co E \Rightarrow 1!E$

$coop ([])$ =

return () \mapsto ()

$\langle yield () \rightarrow r' \rangle \mapsto r' [] ()$

$\langle fork t \rightarrow r' \rangle \mapsto r' [lift t] ()$

$coop (r :: rs)$ =

return () $\mapsto r rs ()$

$\langle yield () \rightarrow r' \rangle \mapsto r (rs ++ [r']) ()$

$\langle fork t \rightarrow r' \rangle \mapsto r' (r :: rs ++ [lift t]) ()$

Example 6: lightweight threads with higher-order fork

Types

Thread $E = 1 \rightarrow 1!Co E$

Res $E = List (Res E) \rightarrow 1 \rightarrow 1!E$

Parameterised handler

$coop : List (Res E) \rightarrow 1!Co E \Rightarrow 1!E$

$coop ([])$ =

return () \mapsto ()

$\langle yield () \rightarrow r' \rangle \mapsto r' [] ()$

$\langle fork t \rightarrow r' \rangle \mapsto r' [lift t] ()$

$coop (r :: rs)$ =

return () $\mapsto r rs ()$

$\langle yield () \rightarrow r' \rangle \mapsto r (rs ++ [r']) ()$

$\langle fork t \rightarrow r' \rangle \mapsto r' (r :: rs ++ [lift t]) ()$

cooperate [main] \Longrightarrow ()

M1 M2 M3 A1 B1 A2 B2

Example 6: pipes

Effect signatures

Sender = {`send` : Nat \rightarrow 1}

Receiver = {`receive` : 1 \rightarrow Nat}

Example 6: pipes

Effect signatures

Sender = {`send` : Nat \rightarrow 1}

Receiver = {`receive` : 1 \rightarrow Nat}

A producer and a consumer

nats : Nat \rightarrow 1!($E \uplus$ Sender)

nats n = `send` n ; nats ($n + 1$)

grabANat : 1 \rightarrow Nat!($E \uplus$ Receiver)

grabANat () = `receive` ()

Example 6: pipes

Effect signatures

Sender = {**send** : Nat \rightarrow 1}

Receiver = {**receive** : 1 \rightarrow Nat}

A producer and a consumer

nats : Nat \rightarrow 1!($E \uplus$ Sender)

nats n = **send** n ; nats ($n + 1$)

grabANat : 1 \rightarrow Nat!($E \uplus$ Receiver)

grabANat () = **receive** ()

Pipes and copipes as shallow handlers

pipe p c = **handle** c () **with**

return x $\mapsto x$

\langle **receive** () $\rightarrow r$ $\rangle \mapsto$ copipe r p

copipe c p = **handle** p () **with**

return x $\mapsto x$

\langle **send** $n \rightarrow r$ $\rangle \mapsto$ pipe r ($\lambda().c$ n)

Example 6: pipes

Effect signatures

Sender = {**send** : Nat \rightarrow 1}

Receiver = {**receive** : 1 \rightarrow Nat}

A producer and a consumer

nats : Nat \rightarrow 1!($E \uplus$ Sender)

nats n = **send** n ; nats ($n + 1$)

grabANat : 1 \rightarrow Nat!($E \uplus$ Receiver)

grabANat () = **receive** ()

Pipes and copipes as shallow handlers

pipe p c = **handle** c () **with**

return x $\mapsto x$

\langle **receive** () $\rightarrow r$ $\rangle \mapsto$ copipe r p

copipe c p = **handle** p () **with**

return x $\mapsto x$

\langle **send** $n \rightarrow r$ $\rangle \mapsto$ pipe r ($\lambda()$. c n)

pipe ($\lambda()$.nats 0) grabANat \rightsquigarrow^+ copipe ($\lambda x.x$) ($\lambda()$.nats 0)

\rightsquigarrow^+ pipe ($\lambda()$.nats 1) ($\lambda()$.0) \rightsquigarrow^+ 0

Example 6: pipes

Effect signatures

Sender = {**send** : Nat \rightarrow 1}

Receiver = {**receive** : 1 \rightarrow Nat}

A producer and a consumer

nats : Nat \rightarrow 1!($E \uplus$ Sender)

nats n = **send** n ; nats ($n + 1$)

grabANat : 1 \rightarrow Nat!($E \uplus$ Receiver)

grabANat () = **receive** ()

Pipes and copipes as shallow handlers

pipe p c = **handle** c () **with**

return x $\mapsto x$

\langle **receive** () $\rightarrow r$ $\rangle \mapsto$ copipe r p

copipe c p = **handle** p () **with**

return x $\mapsto x$

\langle **send** $n \rightarrow r$ $\rangle \mapsto$ pipe r ($\lambda()$. c n)

pipe ($\lambda()$.nats 0) grabANat \rightsquigarrow^+ copipe ($\lambda x.x$) ($\lambda()$.nats 0)

\rightsquigarrow^+ pipe ($\lambda()$.nats 1) ($\lambda()$.0) \rightsquigarrow^+ 0

Exercise: implement pipes using parameterised handlers

Built-in effects

Console I/O

$$\text{Console} = \{\text{inch} : 1 \rightarrow \text{char}$$
$$\quad \text{ouch} : \text{char} \rightarrow 1\}$$
$$\text{print } s = \text{map}(\lambda c. \text{ouch } c) s; ()$$

Generative state

$$\text{GenState} = \{\text{new} : a. \quad a \rightarrow \text{Ref } a,$$
$$\quad \text{write} : a. (\text{Ref } a \times a) \rightarrow 1,$$
$$\quad \text{read} : a. \quad \text{Ref } a \rightarrow a\}$$

Example 7: actors

Process ids

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

Effect signature

Actor $a = \{$
 $\text{self} \quad : \quad 1 \twoheadrightarrow \text{Pid } a,$
 $\text{spawn} : b. (1 \rightarrow 1! \text{Actor } b) \twoheadrightarrow \text{Pid } b,$
 $\text{send} \quad : b. \quad (b \times \text{Pid } b) \twoheadrightarrow 1,$
 $\text{recv} \quad : \quad 1 \twoheadrightarrow a \}$

Example 7: actors

Process ids

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

Effect signature

$$\text{Actor } a = \left\{ \begin{array}{ll} \text{self} & : \quad 1 \twoheadrightarrow \text{Pid } a, \\ \text{spawn} & : b. (1 \rightarrow 1! \text{Actor } b) \twoheadrightarrow \text{Pid } b, \\ \text{send} & : b. \quad (b \times \text{Pid } b) \twoheadrightarrow 1, \\ \text{recv} & : \quad 1 \twoheadrightarrow a \end{array} \right\}$$

An actor chain

$\text{spawnMany} : \text{Pid String} \rightarrow \text{Int} \rightarrow 1!(E \uplus \text{Actor String})$

$\text{spawnMany } p 0 = \text{send} (\text{"ping!"}, p)$

$\text{spawnMany } p n = \text{spawnMany} (\text{spawn } (\lambda(). \text{let } s = \text{recv} () \text{ in print "."; send } (s, p))) (n - 1)$

$\text{chain} : \text{Int} \rightarrow 1!(E \uplus \text{Actor String} \uplus \text{Console})$

$\text{chain } n = \text{spawnMany} (\text{self } ()) n; \text{let } s = \text{recv} () \text{ in print } s$

Example 7: actors — via lightweight threads

$\text{act} : \text{Pid } a \rightarrow 1!(E \uplus \text{Actor } a) \Rightarrow 1!\text{Co } (E \uplus \text{GenState})$

$\text{act } \text{mine} = \text{return } () \quad \mapsto ()$
 $\langle \text{self } () \rightarrow r \rangle \quad \mapsto r \text{ mine mine}$
 $\langle \text{spawn } \text{you} \rightarrow r \rangle \quad \mapsto \text{let } \text{yours} = \text{new } [] \text{ in}$
 $\quad \text{fork } (\lambda().\text{act } \text{yours } (\text{you } ())), r \text{ mine yours}$
 $\langle \text{send } (m, \text{yours}) \rightarrow r \rangle \mapsto \text{let } \text{ms} = \text{read } \text{yours} \text{ in}$
 $\quad \text{write } (\text{yours}, \text{ms} ++ [m]); r \text{ mine } ()$
 $\langle \text{recv } () \rightarrow r \rangle \quad \mapsto \text{letrec } \text{recvWhenReady } () =$
 $\quad \text{case read mine of}$
 $\quad \quad [] \quad \quad \mapsto \text{yield } (); \text{recvWhenReady } ()$
 $\quad \quad (m :: \text{ms}) \mapsto \text{write } (\text{mine}, \text{ms}); r \text{ mine } m$
 $\quad \text{in } \text{recvWhenReady } ()$

cooperate [handle chain 64 with act (new [])] \Rightarrow ()
.....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/ocaml-labs/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