Effect handler oriented programming

Sam Lindley

The University of Edinburgh

FoPSS 2026

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
MET	https://github.com/4y8/modal-effect-types
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/

Resources

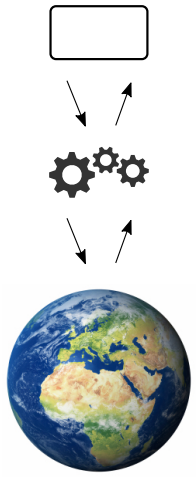
Jeremy Yallop's effects bibliography
https://github.com/yallop/effects-bibliography

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

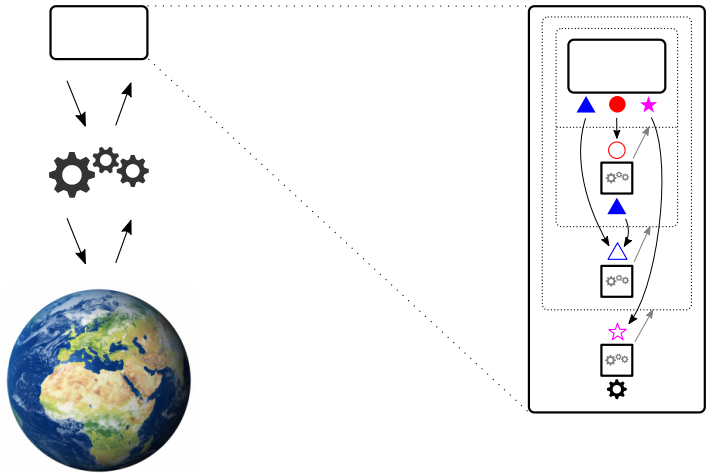
Daniel Hillerström's PhD thesis
"Foundations for programming and implementing effect handlers", 2022

EHOP web page
https://effect-handlers.org/

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`

Effect handlers in space

- Parsimoni's SpaceOS deployed in SpaceX Transporter-13 payload
- Unikernel operating system built on OCaml 5
- Makes essential use of the EIO library

Example 1: choice and failure

Drunk coin tossing

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

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

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

Example 1: choice and failure

Handlers

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

$\mathsf{maybeFail} =$ — exception handler

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

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

Example 1: choice and failure

Handlers

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

$\mathsf{maybeFail} =$ — exception handler

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

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

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

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

Example 1: choice and failure

Handlers

$$\mathsf{maybeFail} : A \,!\, (\{\mathsf{fail} : a.1 \twoheadrightarrow a\} \uplus E) \Rightarrow \mathsf{Maybe}\ A \,!\, E$$
$$\mathsf{maybeFail} = \quad \text{— exception handler}$$
$$\quad \mathbf{return}\ x \ \mapsto \mathsf{Just}\ x \qquad\qquad \mathbf{handle}\ \ 42\ \ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Just}\ 42$$
$$\quad \langle \mathsf{fail}\ () \rangle \ \mapsto \mathsf{Nothing} \qquad\qquad \mathbf{handle}\ \mathsf{fail}\ ()\ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Nothing}$$

$$\mathsf{trueChoice} : A \,!\, (\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\} \uplus E) \Rightarrow A \,!\, E$$
$$\mathsf{trueChoice} = \quad \text{— linear handler}$$
$$\quad \mathbf{return}\ x \qquad\quad \mapsto x$$
$$\quad \langle \mathsf{choose}\ () \to r \rangle \ \mapsto r\ \mathsf{tt}$$

Example 1: choice and failure

Handlers

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

$\textbf{return}\ x$	$\mapsto \mathsf{Just}\ x$	$\textbf{handle}\ \ 42\ \ \textbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Just}\ 42$
$\langle \mathsf{fail}\ () \rangle$	$\mapsto \mathsf{Nothing}$	$\textbf{handle}\ \mathsf{fail}\ ()\ \textbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Nothing}$

$\mathsf{trueChoice} : A \,!\, (\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\} \uplus E) \Rightarrow A \,!\, E$
$\mathsf{trueChoice} =$ — linear handler

$\textbf{return}\ x$	$\mapsto x$	$\textbf{handle}\ \ 42\ \ \textbf{with}\ \mathsf{trueChoice} \Longrightarrow 42$
$\langle \mathsf{choose}\ () \to r \rangle$	$\mapsto r\ \mathsf{tt}$	$\textbf{handle}\ \mathsf{toss}\ ()\ \textbf{with}\ \mathsf{trueChoice} \Longrightarrow \mathsf{Heads}$

Example 1: choice and failure

Handlers

$$\begin{array}{l}
\mathsf{maybeFail} : A \,!\, (\{\mathsf{fail} : a.1 \twoheadrightarrow a\} \uplus E) \Rightarrow \mathsf{Maybe}\ A \,!\, E \\
\mathsf{maybeFail} = \quad \text{— exception handler} \\
\quad \mathbf{return}\ x \ \mapsto \mathsf{Just}\ x \qquad\qquad \mathbf{handle}\ \ 42\ \ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Just}\ 42 \\
\quad \langle \mathsf{fail}\ () \rangle \ \mapsto \mathsf{Nothing} \qquad\qquad \mathbf{handle}\ \mathsf{fail}\ ()\ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Nothing}
\end{array}$$

$$\begin{array}{l}
\mathsf{trueChoice} : A \,!\, (\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\} \uplus E) \Rightarrow A \,!\, E \\
\mathsf{trueChoice} = \quad \text{— linear handler} \\
\quad \mathbf{return}\ x \qquad\ \mapsto x \qquad\qquad \mathbf{handle}\ \ 42\ \ \mathbf{with}\ \mathsf{trueChoice} \Longrightarrow 42 \\
\quad \langle \mathsf{choose}\ () \to r \rangle \ \mapsto r\ \mathsf{tt} \qquad\qquad \mathbf{handle}\ \mathsf{toss}\ ()\ \mathbf{with}\ \mathsf{trueChoice} \Longrightarrow \mathsf{Heads}
\end{array}$$

$$\begin{array}{l}
\mathsf{allChoices} : A \,!\, (\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\} \uplus E) \Rightarrow \mathsf{List}\ A \,!\, E \\
\mathsf{allChoices} = \quad \text{— multishot handler} \\
\quad \mathbf{return}\ x \qquad\ \mapsto [x] \\
\quad \langle \mathsf{choose}\ () \to r \rangle \ \mapsto r\ \mathsf{tt} +\!\!+ r\ \mathsf{ff}
\end{array}$$

Example 1: choice and failure

Handlers

$$\begin{array}{l}
\mathsf{maybeFail} : A \,!\, (\{\mathsf{fail} : a.1 \twoheadrightarrow a\} \uplus E) \Rightarrow \mathsf{Maybe}\ A \,!\, E \\
\mathsf{maybeFail} = \quad \text{— exception handler} \\
\quad \mathbf{return}\ x \ \mapsto \mathsf{Just}\ x \qquad\qquad \mathbf{handle}\ \ 42\ \ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Just}\ 42 \\
\quad \langle \mathsf{fail}\ () \rangle \ \mapsto \mathsf{Nothing} \qquad\qquad \mathbf{handle}\ \mathsf{fail}\ ()\ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Nothing}
\end{array}$$

$$\begin{array}{l}
\mathsf{trueChoice} : A \,!\, (\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\} \uplus E) \Rightarrow A \,!\, E \\
\mathsf{trueChoice} = \quad \text{— linear handler} \\
\quad \mathbf{return}\ x \qquad\ \mapsto x \qquad\qquad \mathbf{handle}\ \ 42\ \ \mathbf{with}\ \mathsf{trueChoice} \Longrightarrow 42 \\
\quad \langle \mathsf{choose}\ () \to r \rangle \ \mapsto r\ \mathsf{tt} \qquad\qquad \mathbf{handle}\ \mathsf{toss}\ ()\ \mathbf{with}\ \mathsf{trueChoice} \Longrightarrow \mathsf{Heads}
\end{array}$$

$$\begin{array}{l}
\mathsf{allChoices} : A \,!\, (\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}\} \uplus E) \Rightarrow \mathsf{List}\ A \,!\, E \\
\mathsf{allChoices} = \quad \text{— multishot handler} \\
\quad \mathbf{return}\ x \qquad\ \mapsto [x] \qquad\qquad \mathbf{handle}\ \ 42\ \ \mathbf{with}\ \mathsf{allChoices} \Longrightarrow [42] \\
\quad \langle \mathsf{choose}\ () \to r \rangle \ \mapsto r\ \mathsf{tt} +\!\!+ r\ \mathsf{ff} \qquad\qquad \mathbf{handle}\ \mathsf{toss}\ ()\ \mathbf{with}\ \mathsf{allChoices} \Longrightarrow [\mathsf{Heads}, \mathsf{Tails}]
\end{array}$$

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))

$\Longrightarrow$

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]

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

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 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 Toss)

$\Longrightarrow$

Nothing

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

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

Example 2: generators

Effect signature

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

A simple generator

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

Handler

$$\begin{aligned}&\mathsf{until} : \mathsf{Nat} \to (1 \to 1 \,!\, (\{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\} \uplus E)) \to \mathsf{List}\ \mathsf{Nat} \,!\, E\\&\mathsf{until}\, \mathit{stop}\, m =\\&\quad \textbf{handle}\ m\,()\ \textbf{with} \quad \text{— affine handler}\\&\qquad \textbf{return}\,() \quad \mapsto []\\&\qquad \langle \mathsf{send}\, i \to r \rangle \ \mapsto \textbf{if}\ i < \mathit{stop}\ \textbf{then}\ i :: r\,()\ \textbf{else}\ []\end{aligned}$$

Example 2: generators

Effect signature

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

A simple generator

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

Handler

$$\begin{aligned}&\mathsf{until} : \mathsf{Nat} \to (1 \to 1 \,!\, (\{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\} \uplus E)) \to \mathsf{List}\ \mathsf{Nat} \,!\, E\\&\mathsf{until}\ \mathit{stop}\ m =\\&\quad \textbf{handle}\ m\,()\ \textbf{with} \quad \text{— affine handler}\\&\qquad \textbf{return}\,() \quad \mapsto []\\&\qquad \langle \mathsf{send}\ i \to r \rangle \ \mapsto \textbf{if}\ i < \mathit{stop}\ \textbf{then}\ i :: r\,()\ \textbf{else}\ []\end{aligned}$$

$$\mathsf{until}\, 8\, (\lambda().\mathsf{nats}\, 0) \implies [0,1,2,3,4,5,6,7]$$

Example 3: state

Effect signature

$$\{\mathsf{put} : \mathsf{Nat} \twoheadrightarrow 1, \quad \mathsf{get} : 1 \twoheadrightarrow \mathsf{Nat}\}$$

Example 3: state

Effect signature

$$\{\mathsf{put} : \mathsf{Nat} \twoheadrightarrow 1, \quad \mathsf{get} : 1 \twoheadrightarrow \mathsf{Nat}\}$$

Standard state handler

$$\mathsf{state} : A \,!\, (\{\mathsf{put} : \mathsf{Nat} \twoheadrightarrow 1, \quad \mathsf{get} : 1 \twoheadrightarrow \mathsf{Nat}\} \uplus E) \Rightarrow (\mathsf{Nat} \to (A \times \mathsf{Nat} \,!\, E)) \,!\, E$$

$\mathsf{state} =$ — closure handler

- $\mathbf{return}\, v \mapsto \lambda s.(v, s)$
- $\langle \mathsf{get}\, () \to r \rangle \mapsto \lambda s.r\, s\, s$
- $\langle \mathsf{put}\, s' \to r \rangle \mapsto \lambda s.r\, ()\, s'$

Example 3: state

Effect signature

$$\{\mathsf{put} : \mathsf{Nat} \twoheadrightarrow 1, \quad \mathsf{get} : 1 \twoheadrightarrow \mathsf{Nat}\}$$

Standard state handler

$$\begin{array}{l}
\mathsf{state}' : \mathsf{Nat} \to A \,!\, (\{\mathsf{put} : \mathsf{Nat} \twoheadrightarrow 1, \quad \mathsf{get} : 1 \twoheadrightarrow \mathsf{Nat}\} \uplus E) \Rightarrow A \times \mathsf{Nat} \,!\, E \\
\mathsf{state}'\,(s) = \quad \text{— parameterised handler} \\
\quad \mathbf{return}\, v \quad \mapsto (v, s) \\
\quad \langle \mathsf{get}\,() \to r \rangle \mapsto r\, s\, s \\
\quad \langle \mathsf{put}\, s' \to r \rangle \mapsto r\,()\, s'
\end{array}$$

$$\mathbf{handle}\ \mathsf{put}\,(1 + \mathsf{get}\,())\ \mathbf{with}\ \mathsf{state}'\,(42) \Longrightarrow$$

Example 3: state

Effect signature

$$\{\mathsf{put} : \mathsf{Nat} \twoheadrightarrow 1, \quad \mathsf{get} : 1 \twoheadrightarrow \mathsf{Nat}\}$$

Standard state handler

$$\begin{array}{l}
\mathsf{state}' : \mathsf{Nat} \to A \,!\, (\{\mathsf{put} : \mathsf{Nat} \twoheadrightarrow 1, \quad \mathsf{get} : 1 \twoheadrightarrow \mathsf{Nat}\} \uplus E) \Rightarrow A \times \mathsf{Nat} \,!\, E \\
\mathsf{state}'\,(s) = \quad \text{— parameterised handler} \\
\quad \mathbf{return}\, v \quad \mapsto (v, s) \\
\quad \langle \mathsf{get}\,() \to r \rangle \mapsto r\, s\, s \\
\quad \langle \mathsf{put}\, s' \to r \rangle \mapsto r\,()\, s'
\end{array}$$

$$\begin{array}{l}
\mathbf{handle}\ \mathsf{put}\,(1 + \mathsf{get}\,())\ \mathbf{with}\ \mathsf{state}'\,(42) \Longrightarrow \\
\quad ((), 43)
\end{array}$$

Operational semantics (parameterised handlers)

Reduction rules

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

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

Evaluation contexts

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

Operational semantics (parameterised handlers)

Reduction rules

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

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

Evaluation contexts

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

Exercise: express parameterised handlers as deep handlers

Typing rules (parameterised handlers)

Effects

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

Computations

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

Operations

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

Handlers

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

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

Example 4: cooperative lightweight threads

Effect signature — higher-order recursive effect signature

$$\begin{aligned}\mathsf{Co} = \{&\mathsf{yield} : 1 \twoheadrightarrow 1,\\ &\mathsf{fork} : \mathsf{Thread} \twoheadrightarrow 1\}\\ \mathsf{Thread} = 1 &\to 1 \,!\, \mathsf{Co}\end{aligned}$$

Example 4: cooperative lightweight threads

Effect signature — higher-order recursive effect signature

$$\begin{aligned}\mathsf{Co} &= \{\mathsf{yield} : 1 \twoheadrightarrow 1,\\ &\qquad \mathsf{fork} : \mathsf{Thread} \twoheadrightarrow 1\}\\ \mathsf{Thread} &= 1 \to 1 \,!\, \mathsf{Co}\end{aligned}$$

A single cooperative program

```
main : Thread
main () = print ("M1 "); fork (fun () → print ("A1 "); yield (); print ("A2 "));
          print ("M2 "); fork (fun () → print ("B1 "); yield (); print ("B2 "));
          print ("M3 ")
```

Example 4: cooperative lightweight threads

Handler

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

Example 4: cooperative lightweight threads

Handler

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

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

M1 A1 M2 A2 B1 M3 B2

Example 4: cooperative lightweight threads

Handler

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

Example 4: cooperative lightweight threads

Handler

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

Example 4: cooperative lightweight threads

Handler

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

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

M1 M2 M3 A1 B1 A2 B2

Operational semantics (shallow handlers)

Reduction rules

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

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

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

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

Computations

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

Operations

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

Handlers

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

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

Example 5: cooperative lightweight threads with UNIX-style fork

Effect signature — first-order non-recursive effect signature

$$\mathsf{UCo} = \{\mathsf{yield} : 1 \twoheadrightarrow 1,\ \mathsf{ufork} : 1 \twoheadrightarrow \mathsf{Bool}\}$$

$$\mathsf{UThread} = 1 \to 1 \,!\, \mathsf{UCo}$$

Example 5: cooperative lightweight threads with UNIX-style fork

Effect signature — first-order non-recursive effect signature

$$\begin{aligned}\mathsf{UCo} &= \{\mathsf{yield} : 1 \twoheadrightarrow 1,\\ &\qquad \mathsf{ufork} : 1 \twoheadrightarrow \mathsf{Bool}\}\\ \mathsf{UThread} &= 1 \to 1 \,!\, \mathsf{UCo}\end{aligned}$$

A single cooperative program

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

Example 5: cooperative lightweight threads with UNIX-style fork

Handler

$$
\begin{array}{l}
\mathsf{ucooperate} : \mathsf{List}\,(\mathsf{UThread}) \to 1 \\
\mathsf{ucooperate}\,[] = () \\
\mathsf{ucooperate}\,(t :: ts) = \\
\quad \mathbf{handle}^{\dagger}\ t()\ \mathbf{with} \quad \text{— multishot shallow handler} \\
\qquad \mathbf{return}\,() \mapsto \mathsf{ucooperate}\,(ts) \\
\qquad \langle \mathsf{yield}\,() \to t \rangle \mapsto \mathsf{ucooperate}\,(ts +\!\!+ [t]) \\
\qquad \langle \mathsf{ufork}\,() \to r \rangle \mapsto \mathsf{ucooperate}\,(ts +\!\!+ [r\,\mathsf{tt}, r\,\mathsf{ff}])
\end{array}
$$

Example 5: cooperative lightweight threads with UNIX-style fork

Handler

$$
\begin{array}{l}
\mathsf{ucooperate} : \mathsf{List}\,(\mathsf{UThread}) \to 1 \\
\mathsf{ucooperate}\,[] = () \\
\mathsf{ucooperate}\,(t :: ts) = \\
\quad \mathbf{handle}^{\dagger}\ t()\ \mathbf{with} \quad \text{— multishot shallow handler} \\
\qquad \mathbf{return}\,() \mapsto \mathsf{ucooperate}\,(ts) \\
\qquad \langle \mathsf{yield}\,() \to t \rangle \mapsto \mathsf{ucooperate}\,(ts +\!\!+ [t]) \\
\qquad \langle \mathsf{ufork}\,() \to r \rangle \mapsto \mathsf{ucooperate}\,(ts +\!\!+ [r\,\mathsf{tt}, r\,\mathsf{ff}])
\end{array}
$$

$$
\mathsf{ucooperate}\,[\mathsf{main}] \implies ()
$$

M1 A1 M2 A2 B1 M3 B2

Example 5: cooperative lightweight threads with UNIX-style fork

Handler

$$
\begin{array}{l}
\mathsf{ucooperate} : \mathsf{List}\,(\mathsf{UThread}) \to 1 \\
\mathsf{ucooperate}\,[] = () \\
\mathsf{ucooperate}\,(t :: ts) = \\
\quad \mathbf{handle}^{\dagger}\ t()\ \mathbf{with} \quad \text{— multishot shallow handler} \\
\qquad \mathbf{return}\,() \mapsto \mathsf{ucooperate}\,(ts) \\
\qquad \langle \mathsf{yield}\,() \to t \rangle \mapsto \mathsf{ucooperate}\,(ts +\!\!+ [t]) \\
\qquad \langle \mathsf{ufork}\,() \to r \rangle \mapsto \mathsf{ucooperate}\,(ts +\!\!+ [r\,\mathsf{tt}, r\,\mathsf{ff}])
\end{array}
$$

Example 5: cooperative lightweight threads with UNIX-style fork

Handler

$$
\begin{array}{l}
\mathsf{ucooperate} : \mathsf{List}\,(\mathsf{UThread}) \to 1 \\
\mathsf{ucooperate}\,[] = () \\
\mathsf{ucooperate}\,(t :: ts) = \\
\quad \mathbf{handle}^{\dagger}\ t()\ \mathbf{with} \quad \text{— multishot shallow handler} \\
\qquad \mathbf{return}\,() \mapsto \mathsf{ucooperate}\,(ts) \\
\qquad \langle \mathsf{yield}\,() \to t \rangle \mapsto \mathsf{ucooperate}\,(ts +\!\!+ [t]) \\
\qquad \langle \mathsf{ufork}\,() \to r \rangle \mapsto \mathsf{ucooperate}\,(ts +\!\!+ [r\,\mathsf{ff}, r\,\mathsf{tt}])
\end{array}
$$

Example 5: cooperative lightweight threads with UNIX-style fork

Handler

$$
\begin{array}{l}
\mathsf{ucooperate} : \mathsf{List}\,(\mathsf{UThread}) \to 1 \\
\mathsf{ucooperate}\,[] = () \\
\mathsf{ucooperate}\,(t :: ts) = \\
\quad \mathbf{handle}^{\dagger}\ t()\ \mathbf{with} \quad \text{— multishot shallow handler} \\
\qquad \mathbf{return}\,() \qquad\quad \mapsto \mathsf{ucooperate}\,(ts) \\
\qquad \langle \mathsf{yield}\,() \to t \rangle \ \mapsto \mathsf{ucooperate}\,(ts +\!\!+ [t]) \\
\qquad \langle \mathsf{ufork}\,() \to r \rangle \ \mapsto \mathsf{ucooperate}\,(ts +\!\!+ [r\,\mathsf{ff}, r\,\mathsf{tt}])
\end{array}
$$

$$\mathsf{ucooperate}\,[\mathsf{main}] \implies ()$$

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

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

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

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

Pipes and copipes as shallow handlers

$$\mathsf{pipe}\, p\, c = \mathbf{handle}^{\dagger}\ c\,()\ \mathbf{with}$$
$$\mathbf{return}\, x \mapsto x$$
$$\langle \mathsf{receive}\,() \to r \rangle \mapsto \mathsf{copipe}\, r\, p$$

$$\mathsf{copipe}\, c\, p = \mathbf{handle}^{\dagger}\ p\,()\ \mathbf{with}$$
$$\mathbf{return}\, x \mapsto x$$
$$\langle \mathsf{send}\, n \to r \rangle \mapsto \mathsf{pipe}\, r\, (\lambda().c\, n)$$

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

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

Pipes and copipes as shallow handlers

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

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

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

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

Pipes and copipes as shallow handlers

$$\mathsf{pipe}\, p\, c = \mathbf{handle}^{\dagger}\ c\, ()\ \mathbf{with}$$
$$\mathbf{return}\, x \mapsto x$$
$$\langle \mathsf{receive}\, () \to r \rangle \mapsto \mathsf{copipe}\, r\, p$$

$$\mathsf{copipe}\, c\, p = \mathbf{handle}^{\dagger}\ p\, ()\ \mathbf{with}$$
$$\mathbf{return}\, x \mapsto x$$
$$\langle \mathsf{send}\, n \to r \rangle \mapsto \mathsf{pipe}\, r\, (\lambda().c\, n)$$

$$\mathsf{pipe}\, (\lambda().\mathsf{nats}\, 0)\, \mathsf{grabANat} \rightsquigarrow^{+} \mathsf{copipe}\, (\lambda x.x)\, (\lambda().\mathsf{nats}\, 0)$$
$$\rightsquigarrow^{+} \mathsf{pipe}\, (\lambda().\mathsf{nats}\, 1)\, (\lambda().0) \rightsquigarrow^{+} 0$$

Exercise: implement pipes using parameterised handlers

Built-in effects

Console I/O

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

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

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 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

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

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

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

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

Example 7: actors — via cooperative concurrency

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

Example 7: actors — via cooperative concurrency

cooperate [**handle** chain 64 **with** act (new [])] $\Longrightarrow$ ()

.....ping!

Example 7: actors — via cooperative concurrency

$$\text{cooperate}\,[\textbf{handle}\ \text{chain}\,64\ \textbf{with}\ \text{act}\,(\text{new}\,[])] \Longrightarrow ()$$
.....ping!

Exercise: Compare three different implementations of actors:

- the one we've just seen (factored through a parameterised handler for cooperative concurrency)
- the same thing, but using shallow handlers
- a single monolithic handler using parameterised handlers

Deep vs shallow

Deep

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

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

The body of the resumption r **reinvokes** the handler

A deep handler performs a **fold** on a computation tree

Deep vs shallow

Shallow

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

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

The body of the resumption r **does not reinvoke** the handler

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

Deep vs shallow

$$\text{Choice } E = \{\text{choose} : 1 \twoheadrightarrow \text{Bool}\} \uplus E$$

Always choose true

Deep

$\text{trueChoice} : (1 \to A \,!\, \text{Choice } E) \to A \,!\, E$
$\text{trueChoice}\, m = \textbf{handle}\ m\,()\ \textbf{with}$
$\quad \textbf{return}\, x \quad \mapsto x$
$\quad \text{choose}\,()\, r \mapsto r\,\text{true}$

Shallow

$\text{trueChoice}^\dagger : (1 \to A \,!\, \text{Choice } E) \to A \,!\, E$
$\text{trueChoice}^\dagger\, m = \textbf{handle}^\dagger\ m\,()\ \textbf{with}$
$\quad \textbf{return}\, x \quad \mapsto x$
$\quad \text{choose}\,()\, r \mapsto \text{trueChoice}^\dagger\,(\lambda().r\,\text{true})$

Deep vs shallow

$$\text{Choice } E = \{\text{choose} : 1 \twoheadrightarrow \text{Bool}\} \uplus E$$

Choose true and false

Deep

$\text{allChoices} : (1 \to A \,!\, \text{Choice } E) \to \text{List } A \,!\, E$
$\text{allChoices } m = \textbf{handle } m\,() \textbf{ with}$
  $\textbf{return } x \quad \mapsto [x]$
  $\text{choose}\,()\, r \mapsto$
    $r\,\text{true} +\!\!+ r\,\text{false}$

Shallow

$\text{allChoices}^\dagger : (1 \to A \,!\, \text{Choice } E) \to \text{List } A \,!\, E$
$\text{allChoices}^\dagger\, m = \textbf{handle}^\dagger\, m\,() \textbf{ with}$
  $\textbf{return } x \quad \mapsto [x]$
  $\text{choose}\,()\, r \mapsto$
    $\text{allChoices}^\dagger\,(\lambda().r\,\text{true}) +\!\!+$
    $\text{allChoices}^\dagger\,(\lambda().r\,\text{false})$

Deep vs shallow

$$\text{Reader } S\ E = \{\text{get} : 1 \twoheadrightarrow S\} \uplus E$$

Read-only state

Deep

$$\begin{aligned}
&\text{reader} : (1 \to A \,!\, \text{Reader } S\ E) \to S \to A \,!\, E \\
&\text{reader } m = \textbf{handle } m\,() \textbf{ with} \\
&\quad \textbf{return } x \mapsto \lambda s.x \\
&\quad \text{get}\,()\,r \quad \mapsto \lambda s.r\,s\,s
\end{aligned}$$

Shallow

$$\begin{aligned}
&\text{reader}^\dagger : (1 \to A \,!\, \text{Reader } S\ E) \to S \to A \,!\, E \\
&\text{reader}^\dagger\ m\,s = \textbf{handle}^\dagger\ m\,() \textbf{ with} \\
&\quad \textbf{return } x \mapsto x \\
&\quad \text{get}\,()\,r \quad \mapsto \text{reader}^\dagger\ (\lambda().r\,s)\,s
\end{aligned}$$

Deep vs shallow

Deep

- can be more concise
- simpler to implement efficiently and without memory leaks

Shallow

- convenient for parameterisation
- convenient for implementing structural recursion schemes other than catamorphisms (e.g. **pipes**)

Deep as shallow

$$
\begin{aligned}
\mathcal{S}[\![\textbf{handle } M \textbf{ with } H]\!] &= \textbf{letrec } h\ f = \textbf{handle}^{\dagger}\ f\ () \textbf{ with } \mathcal{S}[\![H]\!]h \textbf{ in} \\
&\quad\ h\ (\lambda().\mathcal{S}[\![M]\!]) \\
\mathcal{S}[\![H]\!]h &= \mathcal{S}[\![H^{\mathrm{ret}}]\!]h \uplus \mathcal{S}[\![H^{\mathrm{ops}}]\!]h \\
\mathcal{S}[\![\{\textbf{return } x \mapsto N\}]\!]h &= \{\textbf{return } x \mapsto \mathcal{S}[\![N]\!]\} \\
\mathcal{S}[\![\{\mathsf{op}_i\ p\ r \mapsto N_i\}_i]\!]h &= \{\mathsf{op}_i\ p\ r \mapsto \textbf{let } r = \textbf{return } \lambda x.h\,(\lambda().r\,x) \textbf{ in } \mathcal{S}[\![N_i]\!]\}_i
\end{aligned}
$$

Exercise: prove an operational correspondence result for $\mathcal{S}[\![-]\!]$

Shallow as deep

$$\mathcal{D}[\![C \Rightarrow^{\dagger} D]\!] = \mathcal{D}[\![C]\!] \Rightarrow (1 \to \mathcal{D}[\![C]\!], 1 \to \mathcal{D}[\![D]\!])$$

Shallow as deep

$$
\begin{aligned}
\mathcal{D}[\![C \Rightarrow^{\dagger} D]\!] &= \mathcal{D}[\![C]\!] \Rightarrow (1 \to \mathcal{D}[\![C]\!], 1 \to \mathcal{D}[\![D]\!]) \\
\mathcal{D}[\![\textbf{handle}^{\dagger}\ M\ \textbf{with}\ H]\!] &= \textbf{let}\ z = \textbf{handle}\ \mathcal{D}[\![M]\!]\ \textbf{with}\ \mathcal{D}[\![H]\!]\ \textbf{in} \\
&\quad\ \textbf{let}\ (f, g) = z\ \textbf{in}\ g\,() \\
\mathcal{D}[\![H]\!] &= \mathcal{D}[\![H^{\mathrm{ret}}]\!] \uplus \mathcal{D}[\![H^{\mathrm{ops}}]\!] \\
\mathcal{D}[\![\{\textbf{return}\ x \mapsto N\}]\!] &= \{\textbf{return}\ x \mapsto \textbf{return}\ (\lambda().\textbf{return}\ x, \lambda().\mathcal{D}[\![N]\!])\} \\
\mathcal{D}[\![\{\mathsf{op}_i\ p\ r \mapsto N\}_i]\!] &= \{\mathsf{op}_i\ p\ r \mapsto \\
&\qquad \textbf{let}\ r = \lambda x.\textbf{let}\ z = r\,x\ \textbf{in let}\ (f, g) = z\ \textbf{in}\ f\,()\ \textbf{in} \\
&\qquad \textbf{return}\ (\lambda().\textbf{let}\ x = \mathsf{op}_i\ p\ \textbf{in}\ r\,x, \lambda().\mathcal{D}[\![N]\!])\}_i
\end{aligned}
$$

Exercise: prove an operational correspondence result for $\mathcal{D}[\![-]\!]$
(the result is weaker and requires more sophisticated techniques than for $\mathcal{S}[\![-]\!]$)

Sheep effect handlers — a hybrid of shallow and deep handlers

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

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

Like a shallow handler, the body of the resumption need not reinvoke the same handler

Like a deep handler, the body of the resumption must invoke *some* handler

Sheep effect handlers — a hybrid of shallow and deep handlers

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

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

Like a shallow handler, the body of the resumption need not reinvoke the same handler

Like a deep handler, the body of the resumption must invoke *some* handler

WasmFX, the "stack switching" extension to WebAssembly, is based on sheep handlers