Effect handler oriented programming

Sam Lindley

The University of Edinburgh

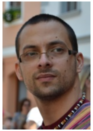
SPLV 2024

Effect handler oriented programming languages

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

Resources

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

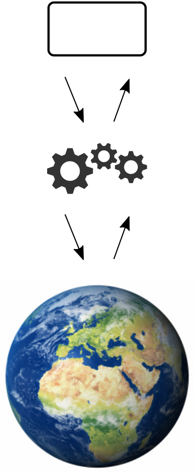


Matija Pretnar's tutorial
"An introduction to algebraic effects and handlers", MFPS 2015

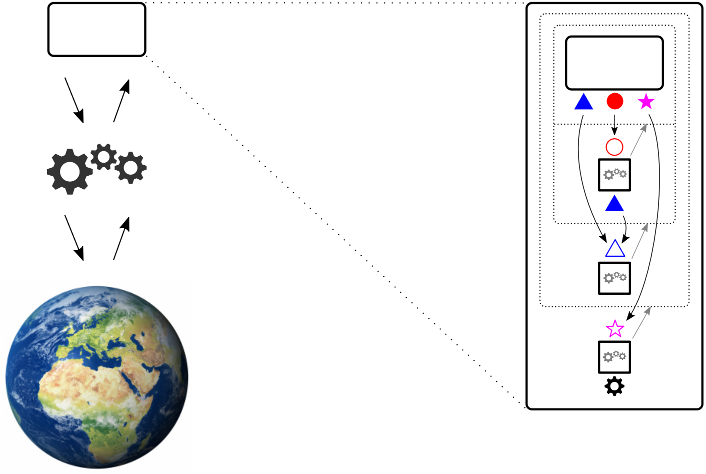
Andrej Bauer's tutorial
"What is algebraic about algebraic effects and handlers?", OPLSS 2018

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

Effect handlers as composable user-defined operating systems



Effect handlers as composable user-defined operating systems



Effect handlers for operating systems

EIO — effects-based direct-style concurrent I/O stack for OCaml
`https://github.com/ocaml-multicore/eio`

Composing UNIX with effect handlers
Foundations for programming and implementing effect handlers, Chapter 2
Daniel Hillerström, PhD thesis, The University of Edinburgh, 2022
`https://www.dhil.net/research/papers/thesis.pdf`

Example 1: choice and failure

Drunk coin tossing

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

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

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

Example 1: choice and failure

Handlers

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

$\mathsf{maybeFail} =$ — exception handler

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

Example 1: choice and failure

Handlers

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

$\text{maybeFail} =$ — exception handler

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

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

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

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

Example 1: choice and failure

Handlers

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

$\mathsf{maybeFail} =$ — exception handler

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

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

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

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

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

$\mathsf{trueChoice} =$ — linear handler

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

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

Example 1: choice and failure

Handlers

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

$\text{maybeFail} =$ — exception handler

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

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

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

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

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

$\text{trueChoice} =$ — linear handler

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

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

$\textbf{handle } 42 \textbf{ with } \text{trueChoice} \Longrightarrow 42$

$\textbf{handle } \text{toss}\,() \textbf{ with } \text{trueChoice} \Longrightarrow \text{Heads}$

Example 1: choice and failure

Handlers

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

$$\begin{aligned}&\mathsf{maybeFail} = \quad \text{— exception handler}\\ &\quad \mathbf{return}\, x \mapsto \mathsf{Just}\, x\\ &\quad \langle \mathsf{fail}\, ()\rangle \mapsto \mathsf{Nothing}\end{aligned}$$

$$\mathbf{handle}\ 42\ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Just}\, 42$$
$$\mathbf{handle}\ \mathsf{fail}\, ()\ \mathbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Nothing}$$

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

$$\begin{aligned}&\mathsf{trueChoice} = \quad \text{— linear handler}\\ &\quad \mathbf{return}\, x \mapsto x\\ &\quad \langle \mathsf{choose}\, () \rightarrow r\rangle \mapsto r\, \mathsf{tt}\end{aligned}$$

$$\mathbf{handle}\ 42\ \mathbf{with}\ \mathsf{trueChoice} \Longrightarrow 42$$
$$\mathbf{handle}\ \mathsf{toss}\, ()\ \mathbf{with}\ \mathsf{trueChoice} \Longrightarrow \mathsf{Heads}$$

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

$$\begin{aligned}&\mathsf{allChoices} = \quad \text{— non-linear handler}\\ &\quad \mathbf{return}\, x \mapsto [x]\\ &\quad \langle \mathsf{choose}\, () \rightarrow r\rangle \mapsto r\, \mathsf{tt} +\!\!+\, r\, \mathsf{ff}\end{aligned}$$

Example 1: choice and failure

Handlers

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

$$\begin{aligned} \mathsf{maybeFail} = \quad & \text{— exception handler} \\ \quad \mathbf{return}\ x & \mapsto \mathsf{Just}\ x \\ \quad \langle \mathsf{fail}\ ()\rangle & \mapsto \mathsf{Nothing} \end{aligned}$$

$$\begin{aligned} \mathbf{handle}\ 42\ \mathbf{with}\ \mathsf{maybeFail} & \Longrightarrow \mathsf{Just}\ 42 \\ \mathbf{handle}\ \mathsf{fail}\ ()\ \mathbf{with}\ \mathsf{maybeFail} & \Longrightarrow \mathsf{Nothing} \end{aligned}$$

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

$$\begin{aligned} \mathsf{trueChoice} = \quad & \text{— linear handler} \\ \quad \mathbf{return}\ x & \mapsto x \\ \quad \langle \mathsf{choose}\ () \to r\rangle & \mapsto r\ \mathsf{tt} \end{aligned}$$

$$\begin{aligned} \mathbf{handle}\ 42\ \mathbf{with}\ \mathsf{trueChoice} & \Longrightarrow 42 \\ \mathbf{handle}\ \mathsf{toss}\ ()\ \mathbf{with}\ \mathsf{trueChoice} & \Longrightarrow \mathsf{Heads} \end{aligned}$$

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

$$\begin{aligned} \mathsf{allChoices} = \quad & \text{— non-linear handler} \\ \quad \mathbf{return}\ x & \mapsto [x] \\ \quad \langle \mathsf{choose}\ () \to r\rangle & \mapsto r\ \mathsf{tt} +\!\!+\ r\ \mathsf{ff} \end{aligned}$$

$$\begin{aligned} \mathbf{handle}\ 42\ \mathbf{with}\ \mathsf{allChoices} & \Longrightarrow [42] \\ \mathbf{handle}\ \mathsf{toss}\ ()\ \mathbf{with}\ \mathsf{allChoices} & \Longrightarrow [\mathsf{Heads}, \mathsf{Tails}] \end{aligned}$$

Example 1: choice and failure

Handler composition

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

Example 1: choice and failure

Handler composition

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

Example 1: choice and failure

Handler composition

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

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

Example 1: choice and failure

Handler composition

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

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

Example 2: generators

Effect signature

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

A simple generator

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

Handler — parameterised handler

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

Example 2: generators

Effect signature

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

A simple generator

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

Handler — parameterised handler

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

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

Operational semantics (parameterised handlers)

Reduction rules

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

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

Evaluation contexts

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

Operational semantics (parameterised handlers)

Reduction rules

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

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

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

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

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

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

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

Example 3: cooperative concurrency (parameterised handler)

Effect signature

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

Example 3: cooperative concurrency (parameterised handler)

Effect signature

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

Two cooperative lightweight threads

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

Example 3: cooperative concurrency (parameterised handler)

Types

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

Handler

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

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

Example 3: cooperative concurrency (parameterised handler)

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

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

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

Helpers

$$\begin{aligned} &\text{coopWith} : \text{Thread } E \to \text{Res } E \\ &\text{coopWith}\ t = \lambda rs.\lambda().\textbf{handle}\ t\,()\ \textbf{with}\ \text{coop}\ rs \end{aligned}$$

$$\begin{aligned} &\text{cooperate} : \text{List}\,(\text{Thread } E) \to 1!E \\ &\text{cooperate}\ ts = \text{coopWith}\,\text{id}\,(\text{map coopWith}\ ts)\,() \end{aligned}$$

Example 3: cooperative concurrency (parameterised handler)

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

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

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

Helpers

$$\text{coopWith} : \text{Thread}\ E \to \text{Res}\ E$$
$$\text{coopWith}\ t = \lambda rs.\lambda().\textbf{handle}\ t\,()\ \textbf{with}\ \text{coop}\ rs$$

$$\text{cooperate} : \text{List}\,(\text{Thread}\ E) \to 1!E$$
$$\text{cooperate}\ ts = \text{coopWith}\,\text{id}\,(\text{map coopWith}\ ts)\,()$$

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

A1 B1 A2 B2

Example 4: cooperative concurrency (shallow handler)

Types

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

Handler

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

Example 4: cooperative concurrency (shallow handler)

Types

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

Handler

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

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

A1 B1 A2 B2

Operational semantics (shallow handlers)

Reduction rules

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

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

Evaluation contexts

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

Typing rules (shallow handlers)

Effects

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

Computations

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

Operations

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

Handlers

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

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

Example 5: cooperative concurrency with UNIX-style fork

Effect signature

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

Example 5: cooperative concurrency with UNIX-style fork

Effect signature

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

A single cooperative program

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

Example 5: cooperative concurrency with UNIX-style fork

Types

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

Parameterised handler

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

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

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

Example 5: cooperative concurrency with UNIX-style fork

Types

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

Parameterised handler

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

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

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

M1 A1 M2 B1 A2 M3 B2

Example 5: cooperative concurrency with UNIX-style fork

Types

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

Parameterised handler

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

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

Example 5: cooperative concurrency with UNIX-style fork

Types

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

Parameterised handler

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

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

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

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

M1 M2 M3 A1 B1 A2 B2

Example 6: cooperative concurrency 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: cooperative concurrency with higher-order fork

Effect signature — recursive effect signature

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

A single cooperative program

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

Example 6: cooperative concurrency 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

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

$$\begin{aligned}
&\text{coop}\,([]) = \\
&\quad \textbf{return}\,() \mapsto () \\
&\quad \langle \text{yield}\,() \to r' \rangle \mapsto r'\,[]\,() \\
&\quad \langle \text{fork}\; t \to r' \rangle \mapsto \text{coopWith}\; t\,[r']\,()
\end{aligned}$$

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

Example 6: cooperative concurrency 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

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

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

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

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

M1 A1 M2 B1 A2 M3 B2

Example 6: cooperative concurrency 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

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

$$\begin{aligned}
&\text{coop}\,([]) = \\
&\quad \textbf{return}\,() \mapsto () \\
&\quad \langle \text{yield}\,() \to r' \rangle \mapsto r'\,[]\,() \\
&\quad \langle \text{fork}\; t \to r' \rangle \mapsto r'\,[\text{coopWith}\; t]\,()
\end{aligned}$$

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

Example 6: cooperative concurrency 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

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

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

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

M1 M2 M3 A1 B1 A2 B2

Example 7: pipes

Effect signatures

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

Example 7: 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 7: 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

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

Pipes and copipes as shallow handlers

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

Example 7: 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

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

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

Example 7: pipes

Effect signatures

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

A producer and a consumer

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

Pipes and copipes as shallow handlers

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

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

Exercise: implement pipes using parameterised handlers

Built-in effects

Console I/O

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

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

Generative state

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

Example 8: 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 8: actors

Process ids

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

Effect signature

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

An actor chain

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

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

Example 8: actors — via cooperative concurrency

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

Example 8: actors — via cooperative concurrency

$$\text{cooperate}\,[\textbf{handle}\ \text{chain}\ 64\ \textbf{with}\ \text{act}\,(\text{new}\,[\,])] \implies ()$$

....ping!

Example 8: actors — via cooperative concurrency

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

....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 effect handlers

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

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

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

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

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

The body of the resumption r reinvokes the handler

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

Shallow effect handlers

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

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

Shallow effect handlers

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

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

The body of the resumption r does not reinvoke the handler

Shallow effect handlers

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

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

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

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

Always choose true

Deep

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

Shallow

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

Deep vs shallow

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

Choose true and false

Deep

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

Shallow

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

Deep vs shallow

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

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

$$\begin{aligned}
\mathcal{S}[\![\textbf{handle}\ M\ \textbf{with}\ H]\!] &= \textbf{letrec}\ h\ f = \textbf{handle}^{\dagger}\ f\ ()\ \textbf{with}\ \mathcal{S}[\![H]\!]h\ \textbf{in} \\
&\quad\ h\ (\lambda().\mathcal{S}[\![M]\!])
\end{aligned}$$

$$\mathcal{S}[\![H]\!]h = \mathcal{S}[\![H^{\mathrm{ret}}]\!]h \uplus \mathcal{S}[\![H^{\mathrm{ops}}]\!]h$$

$$\begin{aligned}
\mathcal{S}[\![\{\textbf{return}\ x \mapsto N\}]\!]h &= \{\textbf{return}\ x \mapsto \mathcal{S}[\![N]\!]\} \\
\mathcal{S}[\![\{\mathsf{op}_i\ p\ r \mapsto N_i\}_i]\!]h &= \{\mathsf{op}_i\ p\ r \mapsto \textbf{let}\ r = \textbf{return}\ \lambda x.h\,(\lambda().r\,x)\ \textbf{in}\ \mathcal{S}[\![N_i]\!]\}_i
\end{aligned}$$

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

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

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

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

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

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

Example 9: effect pollution

Effect signatures

$$\text{Reader} = \{\textsf{get} : 1 \twoheadrightarrow \text{Nat}\} \qquad \text{Failure} = \{\textsf{fail} : a.1 \twoheadrightarrow a\}$$

Example 9: effect pollution

Effect signatures

$$\text{Reader} = \{\text{get} : 1 \twoheadrightarrow \text{Nat}\} \qquad \text{Failure} = \{\text{fail} : a.1 \twoheadrightarrow a\}$$

Handlers

$$\text{reads} : \text{List Nat} \to A!(E \uplus \text{Reader}) \Rightarrow A!(E \uplus \text{Failure})$$

$$\begin{aligned}\text{reads}\,([]) = \;& \mathbf{return}\, x && \mapsto x \\ & \langle \text{get}\,() \to r\rangle && \mapsto \text{fail}\,()\end{aligned} \qquad \begin{aligned}\text{reads}\,(n :: ns) = \;& \mathbf{return}\, x && \mapsto x \\ & \langle \text{get}\,() \to r\rangle && \mapsto r\; ns\; n\end{aligned}$$

$$\begin{aligned}\text{maybeFail} &: A!(E \uplus \text{Failure}) \Rightarrow \text{Maybe}\; A!E \\ \text{maybeFail} &= \mathbf{return}\, x && \mapsto \text{Just}\, x \\ & \quad\; \langle \text{fail}\,() \to r\rangle && \mapsto \text{Nothing}\end{aligned}$$

Example 9: effect pollution

$$\mathsf{bad} : \mathsf{List\ Nat} \to (1 \to A!(E \uplus \mathsf{Reader} \uplus \mathsf{Failure})) \to \mathsf{Maybe}\ A!E$$
$$\mathsf{bad}\ ns\ t = \textbf{handle}\ (\textbf{handle}\ t\ ()\ \textbf{with}\ \mathsf{reads}\ ns)\ \textbf{with}\ \mathsf{maybeFail}$$

Example 9: effect pollution

$$\mathrm{bad} : \mathsf{List\ Nat} \to (1 \to A!(E \uplus \mathsf{Reader} \uplus \mathsf{Failure})) \to \mathsf{Maybe}\ A!E$$
$$\mathrm{bad}\ ns\ t = \mathbf{handle}\ (\mathbf{handle}\ t\ ()\ \mathbf{with}\ \mathrm{reads}\ ns)\ \mathbf{with}\ \mathrm{maybeFail}$$

$$\mathrm{bad}\ [1, 2]\ (\lambda().\mathrm{get}\ () + \mathrm{fail}\ ()) : \mathsf{Maybe\ Nat} \Longrightarrow \mathsf{Nothing}$$

Example 9: effect pollution

$$\begin{aligned}&\mathsf{bad} : \mathsf{List\ Nat} \to (1 \to A!(E \uplus \mathsf{Reader} \uplus \mathsf{Failure})) \to \mathsf{Maybe}\ A!E\\ &\mathsf{bad}\ ns\ t = \mathbf{handle}\ (\mathbf{handle}\ t\,()\ \mathbf{with}\ \mathsf{reads}\ ns)\ \mathbf{with}\ \mathsf{maybeFail}\end{aligned}$$

$$\mathsf{bad}\,[1,2]\,(\lambda().\mathsf{get}\,()+\mathsf{fail}\,()) : \mathsf{Maybe\ Nat} \Longrightarrow \mathsf{Nothing}$$

Exercise: Design a mechanism to allow the Failure effect to be encapsulated.

(The aim is to define

$$\mathsf{good} : \mathsf{List}\ A \to (1 \to \mathsf{Nat}!(E \uplus \mathsf{Reader})) \to \mathsf{Maybe}\ A!E$$

by composing reads and maybeFail such that

$$\mathsf{good}\,[1,2]\,(\lambda().\mathsf{get}\,()+\mathsf{fail}\,()) : \mathsf{Maybe\ Nat!Failure}$$

performs the fail operation.)