Effect handler oriented programming

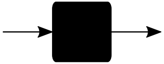
Sam Lindley

Heriot-Watt University and The University of Edinburgh

5th October 2020

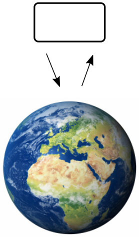
Effects

Programs as black boxes (Church-Turing model)?



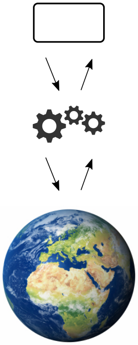
Effects

Programs must interact with their environment



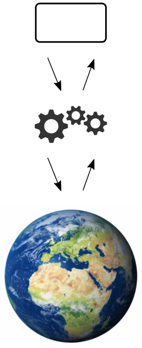
Effects

Programs must interact with their environment



Effects

Programs must interact with their environment

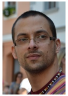


Effects are pervasive

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

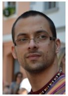
Typically ad hoc and hard-wired

Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009

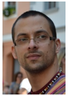
Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009

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

Effect handlers

 Gordon Plotkin  Matija Pretnar

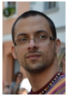
Handlers of algebraic effects, ESOP 2009

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

Give programmer direct access to **environment**

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

Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009

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

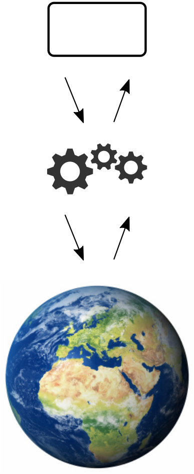
Give programmer direct access to **environment**

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

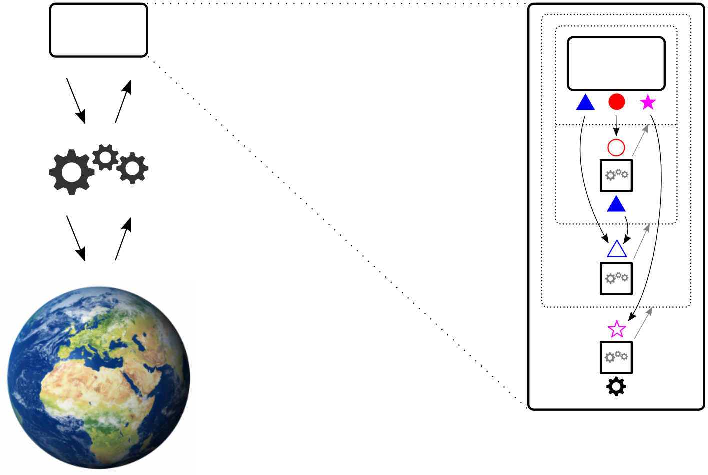
Growing industrial interest

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

Effect handlers as composable user-defined operating systems



Effect handlers as composable user-defined operating systems



Example 1: choice and failure

Effect signature

$$\{\text{choose} : 1 \Rightarrow \text{Bool}, \quad \text{fail} : a.1 \Rightarrow a\}$$

Example 1: choice and failure

Effect signature

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

Drunk coin tossing

$$\mathrm{toss}\,() = \textbf{if } \mathsf{choose}\,()\ \textbf{then}\ \mathrm{Heads}\ \textbf{else}\ \mathrm{Tails}$$

$$\begin{aligned}\mathrm{drunkToss}\,() = {} & \textbf{if } \mathsf{choose}\,()\ \textbf{then} \\ & \quad \textbf{if } \mathsf{choose}\,()\ \textbf{then}\ \mathrm{Heads}\ \textbf{else}\ \mathrm{Tails} \\ & \textbf{else} \\ & \quad \mathsf{fail}\,()\end{aligned}$$

$$\begin{aligned}\mathrm{drunkTosses}\, n = {} & \textbf{if } n = 0\ \textbf{then}\ [] \\ & \textbf{else}\ \mathrm{drunkToss}\,() :: \mathrm{drunkTosses}\,(n-1)\end{aligned}$$

Example 1: choice and failure

Handlers

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

Example 1: choice and failure

Handlers

maybeFail = — exception handler

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

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

Example 1: choice and failure

Handlers

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

$$\begin{aligned} &\mathbf{handle}\ \ 42\ \ \mathbf{with}\ \text{maybeFail} \Longrightarrow \text{Just}\, 42 \\ &\mathbf{handle}\ \textcolor{blue}{\text{fail}}\, ()\ \mathbf{with}\ \text{maybeFail} \Longrightarrow \text{Nothing} \end{aligned}$$

$$\begin{aligned} &\text{trueChoice} = \qquad \textcolor{red}{\text{— linear handler}} \\ &\quad \mathbf{return}\, x \qquad\qquad \mapsto x \\ &\quad \langle \textcolor{blue}{\text{choose}}\, () \to r \rangle \ \mapsto r\ \text{True} \end{aligned}$$

Example 1: choice and failure

Handlers

$$
\begin{array}{l}
\mathrm{maybeFail} = \qquad \textcolor{red}{\text{— exception handler}} \\
\quad \mathbf{return}\ x \ \mapsto \mathrm{Just}\ x \\
\quad \langle \mathrm{fail}\ () \rangle \ \mapsto \mathrm{Nothing}
\end{array}
$$

$$
\begin{array}{l}
\mathbf{handle}\ 42\ \mathbf{with}\ \mathrm{maybeFail} \Longrightarrow \mathrm{Just}\ 42 \\
\mathbf{handle}\ \mathrm{fail}\ ()\ \mathbf{with}\ \mathrm{maybeFail} \Longrightarrow \mathrm{Nothing}
\end{array}
$$

$$
\begin{array}{l}
\mathrm{trueChoice} = \qquad \textcolor{red}{\text{— linear handler}} \\
\quad \mathbf{return}\ x \qquad\qquad \mapsto x \\
\quad \langle \mathrm{choose}\ () \to r \rangle \ \mapsto r\ \mathrm{True}
\end{array}
$$

$$
\begin{array}{l}
\mathbf{handle}\ 42\ \mathbf{with}\ \mathrm{trueChoice} \Longrightarrow 42 \\
\mathbf{handle}\ \mathrm{toss}\ ()\ \mathbf{with}\ \mathrm{trueChoice} \Longrightarrow \mathrm{Heads}
\end{array}
$$

Example 1: choice and failure

Handlers

$\mathsf{maybeFail} =$ — exception handler

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

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

$\mathsf{trueChoice} =$ — linear handler

$$\begin{aligned} &\mathbf{return}\, x &&\mapsto x \\ &\langle \mathsf{choose}\,() \to r\rangle &&\mapsto r\,\mathsf{True} \end{aligned}$$

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

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

$$\begin{aligned} &\mathbf{return}\, x &&\mapsto [x] \\ &\langle \mathsf{choose}\,() \to r\rangle &&\mapsto r\,\mathsf{True} \mathbin{+\!\!+} r\,\mathsf{False} \end{aligned}$$

Example 1: choice and failure

Handlers

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

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

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

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

$$\begin{aligned} &\textbf{return}\, x \mapsto x \\ &\langle \mathsf{choose}\,() \to r \rangle \mapsto r\, \mathsf{True} \end{aligned}$$

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

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

$$\begin{aligned} &\textbf{return}\, x \mapsto [x] \\ &\langle \mathsf{choose}\,() \to r \rangle \mapsto r\, \mathsf{True} \mathbin{+\!\!+} r\, \mathsf{False} \end{aligned}$$

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

Example 1: choice and failure

Handlers

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

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

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

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

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

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

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

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

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

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

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

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

$\quad \langle \mathsf{choose}\,() \to r\rangle \mapsto r\ \mathsf{True} +\!\!+\ r\ \mathsf{False}$

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

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

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

Example 1: choice and failure

Handlers

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

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

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

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

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

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

$$\begin{array}{l}
\mathbf{handle}\ (\mathbf{handle}\ \mathsf{drunkTosses}\, 2\ \mathbf{with}\ \mathsf{maybeFail})\ \mathbf{with}\ \mathsf{allChoices} \Longrightarrow \\
\quad [\mathsf{Just}\, [\mathsf{Heads}, \mathsf{Heads}],\ \mathsf{Just}\, [\mathsf{Heads}, \mathsf{Tails}],\ \mathsf{Nothing}, \\
\quad\ \mathsf{Just}\, [\mathsf{Tails}, \mathsf{Heads}],\ \mathsf{Just}\, [\mathsf{Tails}, \mathsf{Tails}],\ \mathsf{Nothing}, \\
\quad\ \mathsf{Nothing}]
\end{array}$$

Example 1: choice and failure

Handlers

$$
\begin{array}{ll}
\mathrm{maybeFail} = & \text{— exception handler} \\
\quad \mathbf{return}\; x \mapsto \mathrm{Just}\; x & \mathbf{handle}\; 42\; \mathbf{with}\; \mathrm{maybeFail} \Longrightarrow \mathrm{Just}\; 42 \\
\quad \langle \mathrm{fail}\; () \rangle \mapsto \mathrm{Nothing} & \mathbf{handle}\; \mathrm{fail}\; ()\; \mathbf{with}\; \mathrm{maybeFail} \Longrightarrow \mathrm{Nothing}
\end{array}
$$

$$
\begin{array}{ll}
\mathrm{trueChoice} = & \text{— linear handler} \\
\quad \mathbf{return}\; x \mapsto x & \mathbf{handle}\; 42\; \mathbf{with}\; \mathrm{trueChoice} \Longrightarrow 42 \\
\quad \langle \mathrm{choose}\; () \to r \rangle \mapsto r\; \mathrm{True} & \mathbf{handle}\; \mathrm{toss}\; ()\; \mathbf{with}\; \mathrm{trueChoice} \Longrightarrow \mathrm{Heads}
\end{array}
$$

$$
\begin{array}{ll}
\mathrm{allChoices} = & \text{— non-linear handler} \\
\quad \mathbf{return}\; x \mapsto [x] & \mathbf{handle}\; 42\; \mathbf{with}\; \mathrm{allChoices} \Longrightarrow [42] \\
\quad \langle \mathrm{choose}\; () \to r \rangle \mapsto r\; \mathrm{True} +\!\!+\; r\; \mathrm{False} & \mathbf{handle}\; \mathrm{toss}\; ()\; \mathbf{with}\; \mathrm{allChoices} \Longrightarrow [\mathrm{Heads}, \mathrm{Tails}]
\end{array}
$$

$$
\begin{array}{l}
\mathbf{handle}\; (\mathbf{handle}\; \mathrm{drunkTosses}\; 2\; \mathbf{with}\; \mathrm{maybeFail})\; \mathbf{with}\; \mathrm{allChoices} \Longrightarrow \\
\qquad [\mathrm{Just}\; [\mathrm{Heads}, \mathrm{Heads}],\; \mathrm{Just}\; [\mathrm{Heads}, \mathrm{Tails}],\; \mathrm{Nothing}, \\
\qquad\; \mathrm{Just}\; [\mathrm{Tails}, \mathrm{Heads}],\; \mathrm{Just}\; [\mathrm{Tails}, \mathrm{Tails}],\; \mathrm{Nothing}, \\
\qquad\; \mathrm{Nothing}] \\
\mathbf{handle}\; (\mathbf{handle}\; \mathrm{drunkTosses}\; 2\; \mathbf{with}\; \mathrm{allChoices})\; \mathbf{with}\; \mathrm{maybeFail} \Longrightarrow
\end{array}
$$

Example 1: choice and failure

Handlers

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

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

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

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

$$\begin{aligned} &\textbf{return}\ x \mapsto x \\ &\langle \mathsf{choose}\ () \to r\rangle \mapsto r\ \mathsf{True} \end{aligned}$$

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

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

$$\begin{aligned} &\textbf{return}\ x \mapsto [x] \\ &\langle \mathsf{choose}\ () \to r\rangle \mapsto r\ \mathsf{True} +\!\!+\ r\ \mathsf{False} \end{aligned}$$

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

$$\begin{aligned} &\textbf{handle}\ (\textbf{handle}\ \mathsf{drunkTosses}\ 2\ \textbf{with}\ \mathsf{maybeFail})\ \textbf{with}\ \mathsf{allChoices} \Longrightarrow \\ &\qquad [\mathsf{Just}\ [\mathsf{Heads}, \mathsf{Heads}],\ \mathsf{Just}\ [\mathsf{Heads}, \mathsf{Tails}],\ \mathsf{Nothing}, \\ &\qquad\ \mathsf{Just}\ [\mathsf{Tails}, \mathsf{Heads}],\ \mathsf{Just}\ [\mathsf{Tails}, \mathsf{Tails}],\ \mathsf{Nothing}, \\ &\qquad\ \mathsf{Nothing}] \\ &\textbf{handle}\ (\textbf{handle}\ \mathsf{drunkTosses}\ 2\ \textbf{with}\ \mathsf{allChoices})\ \textbf{with}\ \mathsf{maybeFail} \Longrightarrow \mathsf{Nothing} \end{aligned}$$

Small-step operational semantics for (deep) effect handlers

Reduction rules

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

$$
\begin{aligned}
\text{where } H = \textbf{return}\, x \quad &\mapsto N_{\mathrm{ret}} \\
\langle \mathsf{op}_1\ p \rightarrow r \rangle &\mapsto N_{\mathsf{op}_1} \\
&\cdots \\
\langle \mathsf{op}_{\mathsf{k}}\ p \rightarrow r \rangle &\mapsto N_{\mathsf{op}_{\mathsf{k}}}
\end{aligned}
$$

Evaluation contexts

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

Example 2: generators

Effect signature

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

Example 2: generators

Effect signature

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

A simple generator

$$\mathsf{nats}\, n = \mathsf{send}\, n; \mathsf{nats}\,(n+1)$$

Example 2: generators

Effect signature

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

A simple generator

$$\mathsf{nats}\, n = \mathsf{send}\, n; \mathsf{nats}\,(n+1)$$

Handler — parameterised handler

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

Example 2: generators

Effect signature

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

A simple generator

$$\mathsf{nats}\, n = \mathsf{send}\; n; \mathsf{nats}\,(n+1)$$

Handler — parameterised handler

$$\begin{array}{l} \mathsf{until}\, stop = \\ \quad \begin{array}{lcl} \mathbf{return}\,() & \mapsto & [] \\ \langle \mathsf{send}\; n \to r \rangle & \mapsto & \mathbf{if}\;\; n < stop\; \mathbf{then}\;\; n :: r\; stop\,() \\ & & \mathbf{else}\;\; [] \end{array} \end{array}$$

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

Example 3: cooperative concurrency (static)

Effect signature

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

Example 3: cooperative concurrency (static)

Effect signature

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

Two cooperative lightweight threads

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

Example 3: cooperative concurrency (static)

Effect signature

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

Two cooperative lightweight threads

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

Handler — parameterised handler

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

Example 3: cooperative concurrency (static)

Effect signature

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

Two cooperative lightweight threads

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

Handler — parameterised handler

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

Helpers

$$\begin{aligned}
\mathsf{coopWith}\ t &= \lambda rs.\lambda().\mathbf{handle}\ t\,()\ \mathbf{with}\ \mathsf{coop}\ rs \\
\mathsf{cooperate}\ ts &= \mathsf{coopWith}\,\mathsf{id}\,(\mathsf{map}\ \mathsf{coopWith}\ ts)\,()
\end{aligned}$$

Example 3: cooperative concurrency (static)

Effect signature

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

Two cooperative lightweight threads

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

Handler — parameterised handler

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

Helpers

$$\begin{aligned}
\mathrm{coopWith}\ t &= \lambda rs.\lambda().\mathbf{handle}\ t\,()\ \mathbf{with}\ \mathrm{coop}\ rs \\
\mathrm{cooperate}\ ts &= \mathrm{coopWith}\ \mathrm{id}\,(\mathrm{map}\ \mathrm{coopWith}\ \ ts)\,()
\end{aligned}$$

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

A1 B1 A2 B2

Small-step operational semantics for parameterised effect handlers

Reduction rules

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

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

Evaluation contexts

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

Small-step operational semantics for parameterised effect handlers

Reduction rules

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

$$
\begin{array}{rrcl}
\text{where } H\ h = & \textbf{return}\, x & \mapsto & N_{\mathrm{ret}} \\
& \langle \mathsf{op_1}\ p \to r \rangle & \mapsto & N_{\mathsf{op_1}} \\
& & \cdots & \\
& \langle \mathsf{op_k}\ p \to r \rangle & \mapsto & N_{\mathsf{op_k}}
\end{array}
$$

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 non-parameterised handlers

Example 4: cooperative concurrency (dynamic)

Effect signature — recursive effect signature

$$\mathsf{Co} = \{\mathsf{yield} : 1 \Rightarrow 1, \quad \mathsf{fork} : (1 \rightarrow [\mathsf{Co}]1) \Rightarrow 1\}$$

Example 4: cooperative concurrency (dynamic)

Effect signature — recursive effect signature

$$\mathsf{Co} = \{\mathsf{yield} : 1 \Rightarrow 1, \quad \mathsf{fork} : (1 \to [\mathsf{Co}]1) \Rightarrow 1\}$$

A single cooperative program

$$\begin{aligned}\mathsf{main}\,() = {} & \mathsf{print}\ \text{"M1 "}; \mathsf{fork}\,(\lambda().\mathsf{print}\ \text{"A1 "}; \mathsf{yield}\,(); \mathsf{print}\ \text{"A2 "}); \\ & \mathsf{print}\ \text{"M2 "}; \mathsf{fork}\,(\lambda().\mathsf{print}\ \text{"B1 "}; \mathsf{yield}\,(); \mathsf{print}\ \text{"B2 "}); \mathsf{print}\ \text{"M3 "}\end{aligned}$$

Example 4: cooperative concurrency (dynamic)

Effect signature — recursive effect signature

$$\mathsf{Co} = \{\mathsf{yield} : 1 \Rightarrow 1, \quad \mathsf{fork} : (1 \to [\mathsf{Co}]1) \Rightarrow 1\}$$

A single cooperative program

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

Parameterised handler and helpers

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

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

$$\begin{aligned}\mathsf{coopWith}\; t &= \lambda rs.\lambda().\textbf{handle}\; t\,()\;\textbf{with}\;\mathsf{coop}\; rs \\ \mathsf{cooperate}\; ts &= \mathsf{coopWith}\,\mathsf{id}\,(\mathsf{map}\;\mathsf{coopWith}\; ts)\,()\end{aligned}$$

Example 4: cooperative concurrency (dynamic)

Effect signature — recursive effect signature

$$\mathsf{Co} = \{\mathsf{yield} : 1 \Rightarrow 1, \quad \mathsf{fork} : (1 \to [\mathsf{Co}]1) \Rightarrow 1\}$$

A single cooperative program

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

Parameterised handler and helpers

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

$$\begin{aligned}\mathsf{coopWith}\ t &= \lambda rs.\lambda().\mathbf{handle}\ t\,()\ \mathbf{with}\ \mathsf{coop}\ rs \\ \mathsf{cooperate}\ ts &= \mathsf{coopWith}\,\mathsf{id}\,(\mathsf{map}\ \mathsf{coopWith}\ ts)\,()\end{aligned}$$

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

M1 A1 M2 B1 A2 M3 B2

Example 4: cooperative concurrency (dynamic)

Effect signature — recursive effect signature

$$\mathsf{Co} = \{\mathsf{yield} : 1 \Rightarrow 1, \quad \mathsf{fork} : (1 \to [\mathsf{Co}]1) \Rightarrow 1\}$$

A single cooperative program

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

Parameterised handler and helpers

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

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

$$\begin{aligned}\mathsf{coopWith}\ t &= \lambda rs.\lambda().\mathbf{handle}\ t\,()\ \mathbf{with}\ \mathsf{coop}\ rs \\ \mathsf{cooperate}\ ts &= \mathsf{coopWith}\ \mathsf{id}\,(\mathsf{map}\ \mathsf{coopWith}\ ts)\,()\end{aligned}$$

Example 4: cooperative concurrency (dynamic)

Effect signature — recursive effect signature

$$\mathsf{Co} = \{\mathsf{yield} : 1 \Rightarrow 1, \quad \mathsf{fork} : (1 \to [\mathsf{Co}]1) \Rightarrow 1\}$$

A single cooperative program

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

Parameterised handler and helpers

$$\begin{array}{lll} \mathsf{coop}\,([]) = & & \\ \quad \mathbf{return}\,() & \mapsto & () \\ \quad \langle \mathsf{yield}\,() \to r' \rangle & \mapsto & r'\,[]\,() \\ \quad \langle \mathsf{fork}\ t \to r' \rangle & \mapsto & r'\,[\mathsf{coopWith}\ t]\,() \end{array}$$

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

$$\begin{aligned}\mathsf{coopWith}\ t &= \lambda rs.\lambda().\mathbf{handle}\ t\,()\ \mathbf{with}\ \mathsf{coop}\ rs \\ \mathsf{cooperate}\ ts &= \mathsf{coopWith}\ \mathsf{id}\,(\mathsf{map}\ \mathsf{coopWith}\ ts)\,()\end{aligned}$$

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

M1 M2 M3 A1 B1 A2 B2

Example 5: cooperative concurrency (with UNIX-style fork)

Effect signature

$$\mathsf{Co} = \{\mathsf{yield} : 1 \Rightarrow 1, \quad \mathsf{ufork} : 1 \Rightarrow \mathsf{Bool}\}$$

Example 5: cooperative concurrency (with UNIX-style fork)

Effect signature

$$\mathsf{Co} = \{\mathsf{yield} : 1 \Rightarrow 1, \quad \mathsf{ufork} : 1 \Rightarrow \mathsf{Bool}\}$$

A single cooperative program

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

Example 5: cooperative concurrency (with UNIX-style fork)

Effect signature

$$\mathrm{Co} = \{\mathsf{yield} : 1 \Rightarrow 1, \quad \mathsf{ufork} : 1 \Rightarrow \mathrm{Bool}\}$$

A single cooperative program

$$\begin{aligned}
\mathrm{main}\,() = {} & \mathrm{print}\text{ “M1 ”}; \mathbf{if}\ \mathsf{ufork}\,()\ \mathbf{then}\ \mathrm{print}\text{ “A1 ”}; \mathsf{yield}\,(); \mathrm{print}\text{ “A2 ”} \\
& \mathbf{else}\ \mathrm{print}\text{ “M2 ”}; \mathbf{if}\ \mathsf{ufork}\,()\ \mathbf{then}\ \mathrm{print}\text{ “B1 ”}; \mathsf{yield}\,(); \mathrm{print}\text{ “B2 ”}\ \mathbf{else}\ \mathrm{print}\text{ “M3 ”}
\end{aligned}$$

Parameterised handler

$$\begin{aligned}
& \mathrm{coop}\,([]) = \\
& \quad \mathbf{return}\,() \mapsto () \\
& \quad \langle \mathsf{yield}\,() \to r' \rangle \mapsto r'\,[]\,() \\
& \quad \langle \mathsf{ufork}\,() \to r' \rangle \mapsto r'\,[\lambda rs\,().r'\ rs\ \mathrm{False}] \\
& \qquad\qquad\qquad \mathrm{True}
\end{aligned}$$

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

Example 5: cooperative concurrency (with UNIX-style fork)

Effect signature

$$\mathsf{Co} = \{\mathsf{yield} : 1 \Rightarrow 1, \quad \mathsf{ufork} : 1 \Rightarrow \mathsf{Bool}\}$$

A single cooperative program

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

Parameterised handler

$$\begin{array}{lll} \multicolumn{3}{l}{\mathsf{coop}\,([]) =} \\ \quad \mathbf{return}\,() & \mapsto & () \\ \quad \langle \mathsf{yield}\,() \to r' \rangle & \mapsto & r'\,[]\,() \\ \quad \langle \mathsf{ufork}\,() \to r' \rangle & \mapsto & r'\,[\lambda rs\,().r'\,rs\,\mathsf{False}] \\ & & \quad \mathsf{True} \end{array}$$

$$\begin{array}{lll} \multicolumn{3}{l}{\mathsf{coop}\,(r :: rs) =} \\ \quad \mathbf{return}\,() & \mapsto & r\,rs\,() \\ \quad \langle \mathsf{yield}\,() \to r' \rangle & \mapsto & r\,(rs +\!\!+ [r'])\,() \\ \quad \langle \mathsf{ufork}\,() \to r' \rangle & \mapsto & r'\,(r :: rs +\!\!+ [\lambda rs\,().r'\,rs\,\mathsf{False}]) \\ & & \quad \mathsf{True} \end{array}$$

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

M1 A1 M2 B1 A2 M3 B2

Example 5: cooperative concurrency (with UNIX-style fork)

Effect signature

$$\mathsf{Co} = \{\mathsf{yield} : 1 \Rightarrow 1, \quad \mathsf{ufork} : 1 \Rightarrow \mathsf{Bool}\}$$

A single cooperative program

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

Parameterised handler

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

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

Example 5: cooperative concurrency (with UNIX-style fork)

Effect signature

$$\mathsf{Co} = \{\mathsf{yield} : 1 \Rightarrow 1, \quad \mathsf{ufork} : 1 \Rightarrow \mathsf{Bool}\}$$

A single cooperative program

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

Parameterised handler

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

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

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

M1 M2 M3 A1 B1 A2 B2

Example 6: pipes

Effect signatures

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

Example 6: pipes

Effect signatures

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

A producer and a consumer

$$\mathsf{nats}\, n = \mathsf{send}\, n; \mathsf{nats}\,(n+1) \qquad \mathsf{grabANat}\,() = \mathsf{receive}\,()$$

Example 6: pipes

Effect signatures

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

A producer and a consumer

$$\mathsf{nats}\, n = \mathsf{send}\, n; \mathsf{nats}\, (n+1) \qquad \mathsf{grabANat}\, () = \mathsf{receive}\, ()$$

Pipes and copipes as shallow handlers

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

Example 6: pipes

Effect signatures

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

A producer and a consumer

$$\mathsf{nats}\, n = \mathsf{send}\, n; \mathsf{nats}\,(n+1) \qquad \mathsf{grabANat}\,() = \mathsf{receive}\,()$$

Pipes and copipes as shallow handlers

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

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

Example 6: pipes

Effect signatures

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

A producer and a consumer

$$\text{nats}\, n = \textsf{send}\, n; \text{nats}\,(n+1) \qquad \text{grabANat}\,() = \textsf{receive}\,()$$

Pipes and copipes as shallow handlers

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

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

Exercise: implement pipes using deep handlers

Small-step operational semantics for shallow effect handlers

Reduction rules

$$\begin{array}{rcl}
\mathbf{let}\ x = V\ \mathbf{in}\ N & \leadsto & N[V/x] \\
\mathbf{handle}^{\dagger}\ V\ \mathbf{with}\ H & \leadsto & N_{\mathrm{ret}}[V/x] \\
\mathbf{handle}^{\dagger}\ \mathcal{E}[\mathsf{op}\ V]\ \mathbf{with}\ H & \leadsto & N_{\mathsf{op}}[V/p, (\lambda x.\mathcal{E}[x])/r], \quad \mathsf{op} \# \mathcal{E}
\end{array}$$

$$\begin{array}{rrcl}
\text{where } H = & \mathbf{return}\, x & \mapsto & N_{\mathrm{ret}} \\
& \langle \mathsf{op}_1\ p \to r \rangle & \mapsto & N_{\mathsf{op}_1} \\
& & \dots & \\
& \langle \mathsf{op}_k\ p \to r \rangle & \mapsto & N_{\mathsf{op}_k}
\end{array}$$

Evaluation contexts

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

Small-step operational semantics for shallow effect handlers

Reduction rules

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

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

Evaluation contexts

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

Exercise: express shallow handlers as deep handlers

Built-in effects

Console I/O

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

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

Generative state

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

Example 7: actors

Process ids

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

Effect signature

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

Example 7: actors

Process ids

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

Effect signature

$$\begin{array}{rl}\mathsf{Actor}\ a = \{\mathsf{self} & : \qquad\qquad\qquad\qquad 1 \Rightarrow \mathsf{Pid}\ a, \\ \mathsf{spawn} & : b.\ (1 \to [\mathsf{Actor}\ b]1) \Rightarrow \mathsf{Pid}\ b, \\ \mathsf{send} & : b. \qquad\quad (b \times \mathsf{Pid}\ b) \Rightarrow 1, \\ \mathsf{recv} & : \qquad\qquad\qquad\qquad 1 \Rightarrow a\}\end{array}$$

An actor chain

$$\begin{array}{l}\mathsf{spawnMany}\ p\ 0 = \mathsf{send}\ (\text{“ping!”}, p) \\ \mathsf{spawnMany}\ p\ n = \mathsf{spawnMany}\ (\mathsf{spawn}\ (\lambda().\mathbf{let}\ s = \mathsf{recv}\ ()\ \mathbf{in}\ \mathsf{print}\ \text{“.”}; \mathsf{send}\ (s, p)))\ (n-1)\end{array}$$

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

Example 7: actors

Actors via cooperative concurrency

$$
\begin{array}{l}
\mathsf{act}\ \mathit{mine} = \\
\quad \begin{array}{lcl}
\mathbf{return}\,() & \mapsto & () \\
\langle \mathsf{self}\,() \to r \rangle & \mapsto & r\ \mathit{mine}\ \mathit{mine} \\
\langle \mathsf{spawn}\ \mathit{you} \to r \rangle & \mapsto & \mathbf{let}\ \mathit{yours} = \mathsf{new}\,[]\ \mathbf{in} \\
 & & \mathsf{fork}\,(\lambda().\mathsf{act}\ \mathit{yours}\,(\mathit{you}\,()));\ r\ \mathit{mine}\ \mathit{yours} \\
\langle \mathsf{send}\,(m, \mathit{yours}) \to r \rangle & \mapsto & \mathbf{let}\ \mathit{ms} = \mathsf{read}\ \mathit{yours}\ \mathbf{in} \\
 & & \mathsf{write}\,(\mathit{yours}, \mathit{ms} \mathbin{+\!\!+} [m]);\ r\ \mathit{mine}\,() \\
\langle \mathsf{recv}\,() \to r \rangle & \mapsto & \mathbf{case}\ \mathsf{read}\ \mathit{mine}\ \mathbf{of} \\
 & & \quad \begin{array}{lcl}
 [] & \mapsto & \mathsf{yield}\,();\ r\ \mathit{mine}\,(\mathsf{recv}\,()) \\
 (m :: \mathit{ms}) & \mapsto & \mathsf{write}\,(\mathit{mine}, \mathit{ms});\ r\ \mathit{mine}\ m
 \end{array}
\end{array}
\end{array}
$$

Example 7: actors

Actors via cooperative concurrency

$$
\begin{array}{l}
\mathrm{act}\ \mathit{mine} = \\
\quad \begin{array}{lll}
\mathbf{return}\,() & \mapsto & () \\
\langle \mathsf{self}\,() \to r \rangle & \mapsto & r\ \mathit{mine}\ \mathit{mine} \\
\langle \mathsf{spawn}\ \mathit{you} \to r \rangle & \mapsto & \mathbf{let}\ \mathit{yours} = \mathsf{new}\,[]\ \mathbf{in} \\
 & & \mathsf{fork}\,(\lambda().\mathrm{act}\ \mathit{yours}\,(\mathit{you}\,()));\ r\ \mathit{mine}\ \mathit{yours} \\
\langle \mathsf{send}\,(m, \mathit{yours}) \to r \rangle & \mapsto & \mathbf{let}\ \mathit{ms} = \mathsf{read}\ \mathit{yours}\ \mathbf{in} \\
 & & \mathsf{write}\,(\mathit{yours}, \mathit{ms} +\!\!+ [m]);\ r\ \mathit{mine}\,() \\
\langle \mathsf{recv}\,() \to r \rangle & \mapsto & \mathbf{case}\ \mathsf{read}\ \mathit{mine}\ \mathbf{of} \\
 & & \quad \begin{array}{lll}
 [] & \mapsto & \mathsf{yield}\,();\ r\ \mathit{mine}\,(\mathsf{recv}\,()) \\
 (m :: \mathit{ms}) & \mapsto & \mathsf{write}\,(\mathit{mine}, \mathit{ms});\ r\ \mathit{mine}\ m
 \end{array}
\end{array}
\end{array}
$$

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

....ping!

Effect handler oriented programming languages

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

Effect handlers — some of my contributions

Handlers in action (ICFP 2013)
with Kammar and Oury

Effect handlers in Links (TyDe 2016 / JFP 2020)
with Hillerström

Frank programming language (POPL 2017 / JFP 2020)
with Convent, McBride, and McLaughlin

Effect handlers — some of my contributions

Handlers in action (ICFP 2013)
with Kammar and Oury

Effect handlers in Links (TyDe 2016 / JFP 2020)
with Hillerström

Frank programming language (POPL 2017 / JFP 2020)
with Convent, McBride, and McLaughlin

Expressive power of effect handlers (ICFP 2017 / JFP 2019)
with Forster, Kammar, and Pretnar

Effect handlers — some of my contributions

Handlers in action (ICFP 2013)
with Kammar and Oury

Effect handlers in Links (TyDe 2016 / JFP 2020)
with Hillerström

Frank programming language (POPL 2017 / JFP 2020)
with Convent, McBride, and McLaughlin

Expressive power of effect handlers (ICFP 2017 / JFP 2019)
with Forster, Kammar, and Pretnar

Continuation-passing style for effect handlers (FSCD 2017 / JFP 2020)
with Atkey, Hillerström, and Sivaramakrishnan

Shallow effect handlers (APLAS 2018 / JFP 2020)
with Hillerström

Linear effect handlers for session exceptions (POPL 2019)
with Decova, Fowler, and Morris

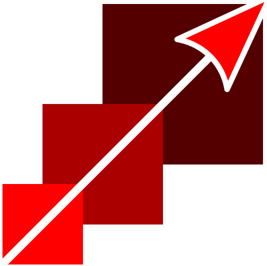
Scalability challenges

Modularity — effect typing

- Effect encapsulation
- Linearity
- Generativity
- Indexed effects
- Equations

Efficiency — compilation techniques

- Segmented stacks
  (Multicore OCaml / C library)
- Continuation Passing Style
  (JavaScript backends)
- Fusion (Haskell libraries / Eff)
- Staging (Scala Effekt library)



New directions

Effect handlers for Wasm
add effect handlers once and for all — avoid pitfalls of JavaScript

Asynchronous effects
pre-emptive concurrency; reactive programming

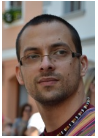
Gradually typed effect handlers
transition mainstream languages towards effect typing

Hardware capabilities as dynamic effects
safe effect handlers in C? efficient implementation?

Lexically scoped effect handlers
improved hygiene? improved performance? improved reasoning?

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?",
Dagstuhl and OPLSS 2018

Bonus slides

Example 8: effect pollution

Effect signatures

$$\text{Receiver} = \{\text{receive} : 1 \Rightarrow \text{Nat}\} \qquad \text{Failure} = \{\text{fail} : a.1 \Rightarrow a\}$$

Example 8: effect pollution

Effect signatures

$$\mathsf{Receiver} = \{\mathsf{receive} : 1 \Rightarrow \mathsf{Nat}\} \qquad \mathsf{Failure} = \{\mathsf{fail} : a.1 \Rightarrow a\}$$

Handlers

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

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

Example 8: effect pollution

Effect signatures

$$\text{Receiver} = \{\text{receive} : 1 \Rightarrow \text{Nat}\} \qquad \text{Failure} = \{\text{fail} : a.1 \Rightarrow a\}$$

Handlers

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

$$\begin{aligned}
&\text{maybeFail} = \\
&\quad \textbf{return}\, x \quad\quad\; \mapsto \text{Just}\, x \\
&\quad \langle \text{fail}\,() \to r\rangle \mapsto \text{Nothing}
\end{aligned}$$

$$\text{bad}\; ns\; t = \textbf{handle}\; (\textbf{handle}\; t\,()\; \textbf{with}\; \text{receives}\; ns)\; \textbf{with}\; \text{maybeFail}$$

Example 8: effect pollution

Effect signatures

$$\text{Receiver} = \{\mathsf{receive} : 1 \Rightarrow \mathsf{Nat}\} \qquad \text{Failure} = \{\mathsf{fail} : a.1 \Rightarrow a\}$$

Handlers

$$\begin{array}{ll} \multicolumn{2}{l}{\mathsf{receives}\,([]) =} \\ \quad \mathbf{return}\,x & \mapsto x \\ \quad \langle \mathsf{receive}\,() \to r \rangle & \mapsto \mathsf{fail}\,() \end{array} \qquad \begin{array}{ll} \multicolumn{2}{l}{\mathsf{receives}\,(n :: ns) =} \\ \quad \mathbf{return}\,x & \mapsto x \\ \quad \langle \mathsf{receive}\,() \to r \rangle & \mapsto r\;ns\;n \end{array}$$

$$\begin{array}{ll} \multicolumn{2}{l}{\mathsf{maybeFail} =} \\ \quad \mathbf{return}\,x & \mapsto \mathsf{Just}\,x \\ \quad \langle \mathsf{fail}\,() \to r \rangle & \mapsto \mathsf{Nothing} \end{array}$$

$$\mathsf{bad}\;ns\;t = \mathbf{handle}\;(\mathbf{handle}\;t\;()\;\mathbf{with}\;\mathsf{receives}\;ns)\;\mathbf{with}\;\mathsf{maybeFail}$$

$$\mathsf{bad}\,[1,2]\,(\lambda().\mathsf{receive}\,() + \mathsf{fail}\,()) \Longrightarrow \mathsf{Nothing}$$

Example 9: counting

Predicates as higher order functions

$$\mathsf{Pred} = (\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}$$

Signature of a counting function

$$\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat}$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1) = 2$$

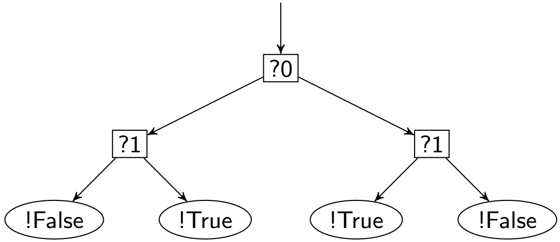
Example 9: counting

Counting with a choice handler

$$
\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{count} = \lambda p.\mathbf{handle}\ p\,(\lambda\_.\mathsf{choose}\,())\ \mathbf{with} \\
&\qquad\qquad\qquad \mathbf{return}\ x \qquad\qquad \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\
&\qquad\qquad\qquad \langle \mathsf{choose}\ () \to r \rangle \ \mapsto r\,\mathsf{True} + r\,\mathsf{False}
\end{aligned}
$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1)$$



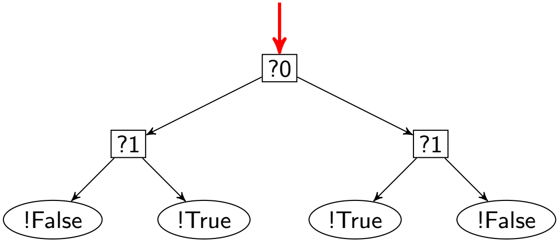
Example 9: counting

Counting with a choice handler

$$
\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat}\\
&\mathsf{count} = \lambda p.\mathbf{handle}\ p\,(\lambda\_.\mathsf{choose}\,())\ \mathbf{with}\\
&\qquad\qquad\qquad \mathbf{return}\ x \qquad\qquad\ \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0\\
&\qquad\qquad\qquad \langle \mathsf{choose}\ () \to r\rangle \ \mapsto r\,\mathsf{True} + r\,\mathsf{False}
\end{aligned}
$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1)$$



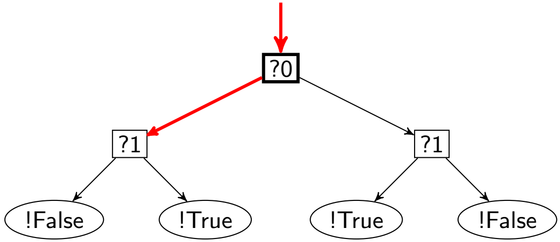
Example 9: counting

Counting with a choice handler

$$\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat}$$
$$\mathsf{count} = \lambda p.\mathbf{handle}\ p\,(\lambda\_.\mathsf{choose}\,())\ \mathbf{with}$$
$$\begin{array}{ll} \mathbf{return}\ x & \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\ \langle \mathsf{choose}\ () \to r\rangle & \mapsto r\,\mathsf{True} + r\,\mathsf{False} \end{array}$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1)$$



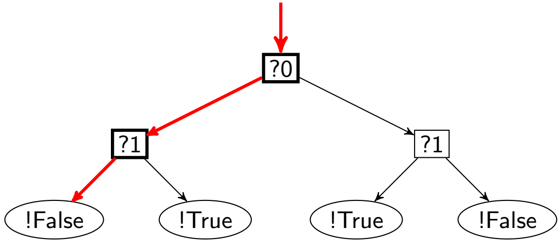
Example 9: counting

Counting with a choice handler

$$\begin{aligned}
&\text{count} : ((\text{Nat} \to \text{Bool}) \to \text{Bool}) \to \text{Nat} \\
&\text{count} = \lambda p.\textbf{handle}\ p\,(\lambda_.\text{choose}\,())\ \textbf{with} \\
&\qquad\qquad\qquad \textbf{return}\ x \qquad\quad\ \ \mapsto \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ 0 \\
&\qquad\qquad\qquad \langle \text{choose}\ () \to r\rangle \mapsto r\,\text{True} + r\,\text{False}
\end{aligned}$$

Exclusive or

$$\text{count}\,(\lambda v.\textbf{if}\ v\ 0\ \textbf{then}\ \text{not}\,(v\ 1)\ \textbf{else}\ v\ 1)$$



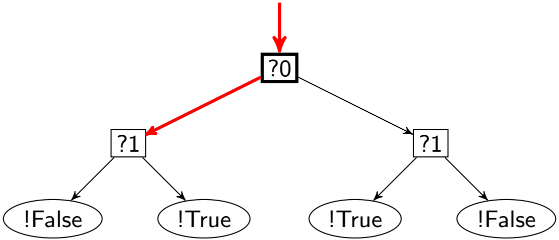
Example 9: counting

Counting with a choice handler

$$\begin{array}{l} \mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\ \mathsf{count} = \lambda p.\mathbf{handle}\ p\,(\lambda\_.\mathsf{choose}\,())\ \mathbf{with} \\ \qquad\qquad\qquad \begin{array}{ll} \mathbf{return}\ x & \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\ \langle \mathsf{choose}\ () \to r \rangle & \mapsto r\,\mathsf{True} + r\,\mathsf{False} \end{array} \end{array}$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1)$$



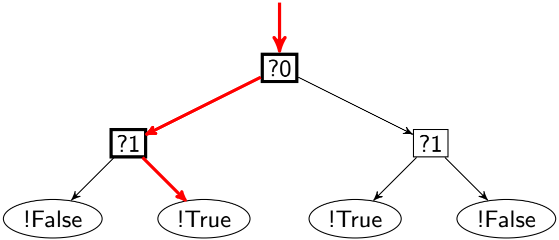
Example 9: counting

Counting with a choice handler

$$
\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat}\\
&\mathsf{count} = \lambda p.\mathbf{handle}\ p\,(\lambda\_.\mathsf{choose}\,())\ \mathbf{with}\\
&\qquad\qquad\quad \mathbf{return}\ x \qquad\qquad \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0\\
&\qquad\qquad\quad \langle \mathsf{choose}\ () \to r\rangle\ \ \mapsto r\,\mathsf{True} + r\,\mathsf{False}
\end{aligned}
$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1)$$



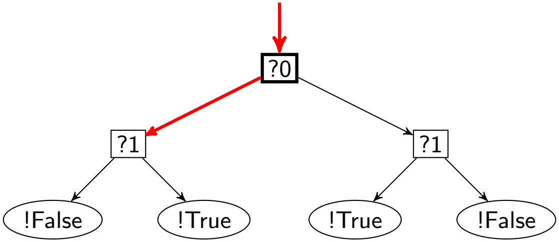
Example 9: counting

Counting with a choice handler

$$\begin{array}{l} \text{count} : ((\text{Nat} \to \text{Bool}) \to \text{Bool}) \to \text{Nat} \\ \text{count} = \lambda p.\textbf{handle}\ p\,(\lambda\_.\text{choose}\,())\ \textbf{with} \\ \qquad\qquad\qquad \begin{array}{ll} \textbf{return}\ x & \mapsto \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ 0 \\ \langle \text{choose}\ () \to r \rangle & \mapsto r\,\text{True} + r\,\text{False} \end{array} \end{array}$$

Exclusive or

$$\text{count}\,(\lambda v.\textbf{if}\ v\ 0\ \textbf{then}\ \text{not}\,(v\ 1)\ \textbf{else}\ v\ 1)$$



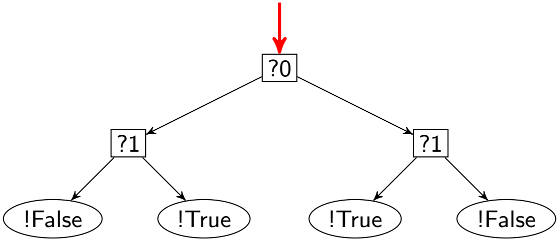
Example 9: counting

Counting with a choice handler

$$\begin{array}{l}
\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
\mathsf{count} = \lambda p.\mathbf{handle}\ p\,(\lambda_.\mathsf{choose}\,())\ \mathbf{with} \\
\qquad\qquad \begin{array}{ll}
\mathbf{return}\ x & \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\
\langle \mathsf{choose}\ () \to r \rangle & \mapsto r\,\mathsf{True} + r\,\mathsf{False}
\end{array}
\end{array}$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1)$$



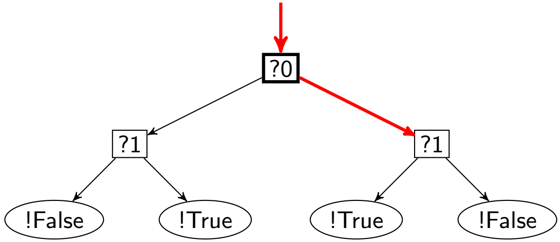
Example 9: counting

Counting with a choice handler

$$
\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{count} = \lambda p.\mathbf{handle}\ p\,(\lambda\_.\mathsf{choose}\,())\ \mathbf{with} \\
&\qquad\qquad \mathbf{return}\ x \qquad\qquad \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\
&\qquad\qquad \langle \mathsf{choose}\ () \to r \rangle \ \mapsto r\,\mathsf{True} + r\,\mathsf{False}
\end{aligned}
$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1)$$



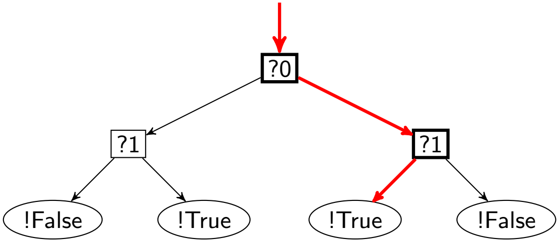
Example 9: counting

Counting with a choice handler

$$
\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{count} = \lambda p.\mathbf{handle}\ p\,(\lambda\_.\mathsf{choose}\,())\ \mathbf{with} \\
&\qquad\qquad\qquad \begin{array}{ll} \mathbf{return}\ x & \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\ \langle \mathsf{choose}\ () \to r \rangle & \mapsto r\,\mathsf{True} + r\,\mathsf{False} \end{array}
\end{aligned}
$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1)$$



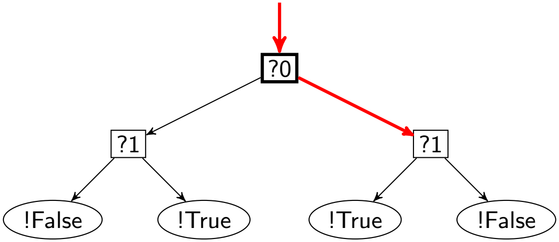
Example 9: counting

Counting with a choice handler

$$
\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{count} = \lambda p.\mathbf{handle}\ p\,(\lambda\_.\mathsf{choose}\,())\ \mathbf{with} \\
&\qquad\qquad\qquad \mathbf{return}\ x \qquad\qquad \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\
&\qquad\qquad\qquad \langle \mathsf{choose}\ () \to r \rangle \ \mapsto r\,\mathsf{True} + r\,\mathsf{False}
\end{aligned}
$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1)$$



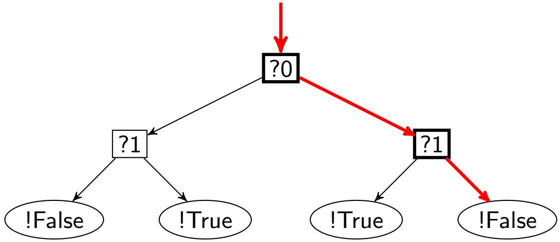
Example 9: counting

Counting with a choice handler

$$
\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{count} = \lambda p.\mathbf{handle}\ p\,(\lambda\_.\mathsf{choose}\,())\ \mathbf{with} \\
&\qquad\qquad \mathbf{return}\ x \qquad\qquad \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\
&\qquad\qquad \langle \mathsf{choose}\ () \to r \rangle \ \mapsto r\,\mathsf{True} + r\,\mathsf{False}
\end{aligned}
$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1)$$



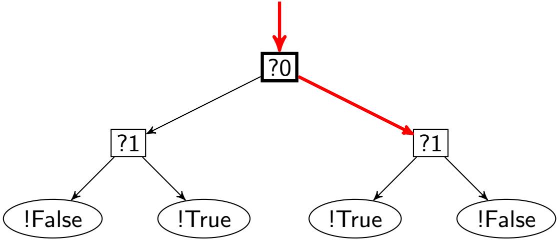
Example 9: counting

Counting with a choice handler

$$
\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{count} = \lambda p.\mathbf{handle}\ p\,(\lambda\_.\mathsf{choose}\,())\ \mathbf{with} \\
&\qquad\qquad\quad \mathbf{return}\ x \qquad\qquad \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\
&\qquad\qquad\quad \langle \mathsf{choose}\ () \to r\rangle \ \ \mapsto r\,\mathsf{True} + r\,\mathsf{False}
\end{aligned}
$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1)$$



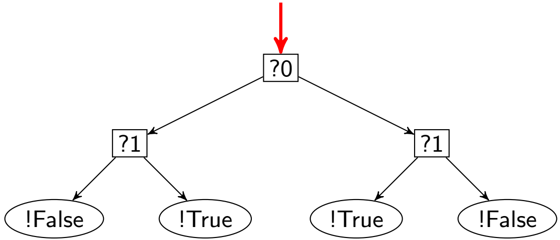
Example 9: counting

Counting with a choice handler

$$\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{count} = \lambda p.\mathbf{handle}\ p\,(\lambda_.\mathsf{choose}\,())\ \mathbf{with} \\
&\qquad\qquad \begin{array}{ll}
\mathbf{return}\ x & \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\
\langle \mathsf{choose}\ () \to r \rangle & \mapsto r\,\mathsf{True} + r\,\mathsf{False}
\end{array}
\end{aligned}$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1)$$



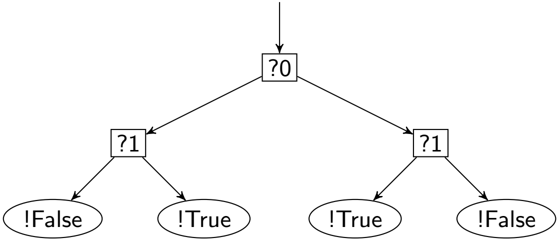
Example 9: counting

Counting with a choice handler

$$
\begin{aligned}
&\mathsf{count} : ((\mathsf{Nat} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Nat} \\
&\mathsf{count} = \lambda p.\mathbf{handle}\ p\,(\lambda\_.\mathsf{choose}\,())\ \mathbf{with} \\
&\qquad\qquad\qquad \mathbf{return}\ x \qquad\qquad \mapsto \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\
&\qquad\qquad\qquad \langle \mathsf{choose}\ () \to r \rangle \ \mapsto r\,\mathsf{True} + r\,\mathsf{False}
\end{aligned}
$$

Exclusive or

$$\mathsf{count}\,(\lambda v.\mathbf{if}\ v\ 0\ \mathbf{then}\ \mathsf{not}\,(v\ 1)\ \mathbf{else}\ v\ 1)$$



Example 10: pipes using multihandlers

Effect signatures

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

$$\text{Fail} = \{\text{fail} : a.1 \Rightarrow a\}$$

Example 10: pipes using multihandlers

Effect signatures

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

$$\mathsf{Fail} = \{\mathsf{fail} : a.1 \Rightarrow a\}$$

A producer and a consumer

$$\mathsf{nats}\, n = \mathsf{send}\, n; \mathsf{nats}\,(n+1) \qquad \mathsf{grabANat}\,() = \mathsf{receive}\,()$$

Example 10: pipes using multihandlers

Effect signatures

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

$$\mathsf{Fail} = \{\mathsf{fail} : a.1 \Rightarrow a\}$$

A producer and a consumer

$$\mathsf{nats}\; n = \mathsf{send}\; n; \mathsf{nats}\,(n+1) \qquad \mathsf{grabANat}\,() = \mathsf{receive}\,()$$

A pipe multihandler

$$\begin{array}{lllll}
\mathsf{pipe} = & \multicolumn{4}{l}{\text{— multihandler}} \\
 & \langle \mathsf{send}\; n & \mid \mathsf{receive}\,() & \to r\rangle & \mapsto r\,()\; n \\
 & \langle _ & \mid \mathbf{return}\; x\rangle & & \mapsto x \\
 & \langle \mathbf{return}\,() & \mid \mathsf{receive}\,()\rangle & & \mapsto \mathsf{fail}\,()
\end{array}$$

Example 10: pipes using multihandlers

Effect signatures

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

$$\mathsf{Fail} = \{\mathsf{fail} : a.1 \Rightarrow a\}$$

A producer and a consumer

$$\mathsf{nats}\, n = \mathsf{send}\, n; \mathsf{nats}\,(n+1) \qquad \mathsf{grabANat}\,() = \mathsf{receive}\,()$$

A pipe multihandler

$$\begin{array}{lllll}
\mathsf{pipe} = & \multicolumn{4}{l}{\text{— multihandler}} \\
& \langle \mathsf{send}\, n & \mid \mathsf{receive}\,() & \to r\rangle & \mapsto r\,()\,n \\
& \langle _ & \mid \mathbf{return}\, x\rangle & & \mapsto x \\
& \langle \mathbf{return}\,() & \mid \mathsf{receive}\,()\rangle & & \mapsto \mathsf{fail}\,()
\end{array}$$

$$\mathbf{handle}\ \mathsf{nats}\, 0 \mid \mathsf{grabANat}\,()\ \mathbf{with}\ \mathsf{pipe} \implies 0$$