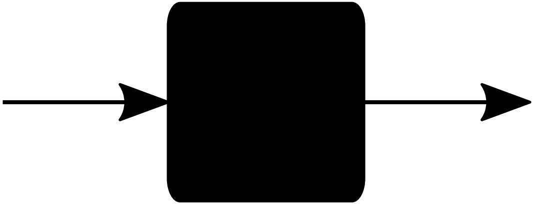
Algebraic effects and effect handlers

Sam Lindley

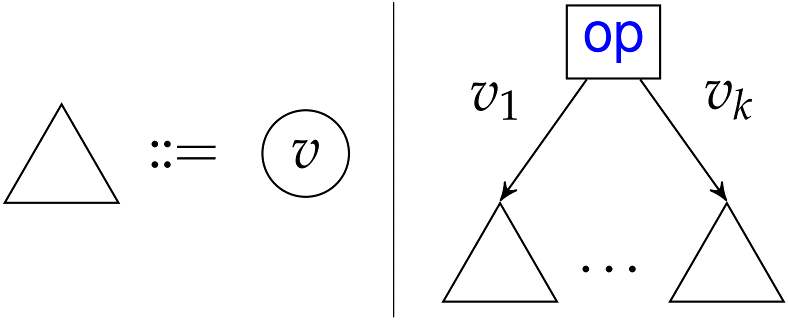
Heriot-Watt University / The University of Edinburgh / Effect Handlers Ltd.

17th December 2020

What is a pure computation?

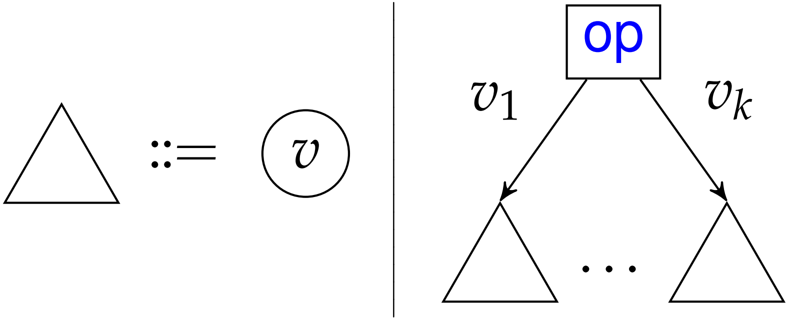


What is an effectful computation?



A **command-response** tree (aka **interaction tree**)

What is an effectful computation?

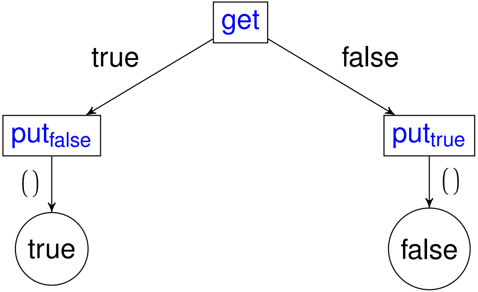


A **command-response** tree (aka **interaction tree**)

Effectful computation is all about **interaction** with some **context**

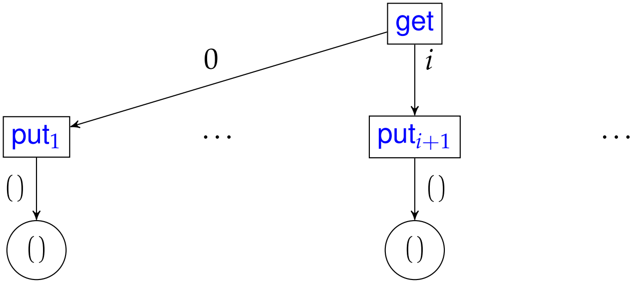
Example: boolean state (bit toggling)

$$\begin{aligned} \mathsf{get} &: \mathsf{bool} \\ \mathsf{put}_{\mathsf{true}} &: 1 \\ \mathsf{put}_{\mathsf{false}} &: 1 \end{aligned}$$



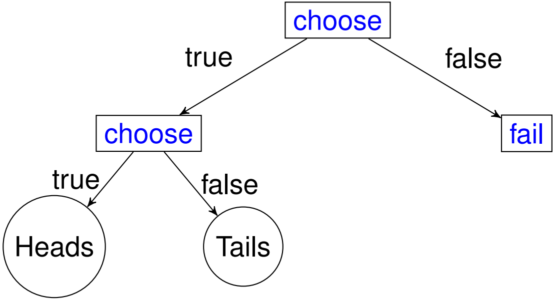
Example: natural number state (increment)

$$\begin{aligned} \mathsf{get} &: \mathbb{N} \\ \mathsf{put}_i &: 1, \quad i \in \mathbb{N} \end{aligned}$$

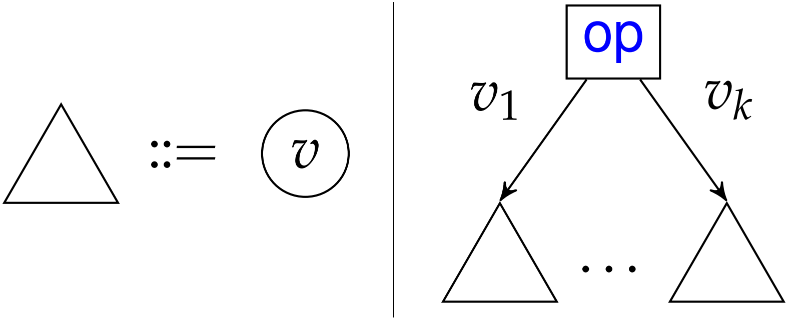


Example: nondeterminism (drunk coin toss)

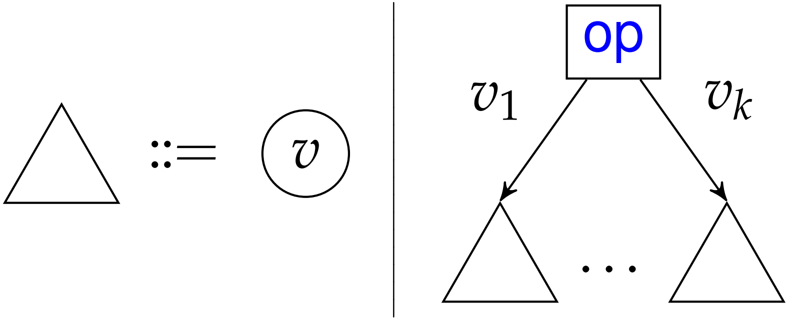
choose : bool
fail : 0



What is an effectful computation?



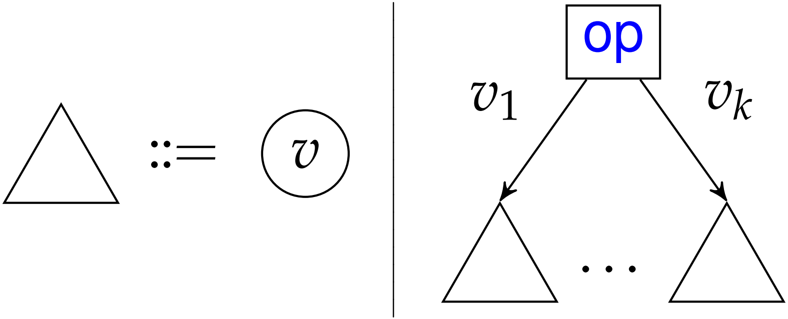
What is an effectful computation?



Equivalently (ignoring edge labels)

$$m ::= \textbf{return}\, v \mid \mathsf{op}\ \langle m_1,\ \ldots,\ m_k \rangle$$

What is an effectful computation?



Equivalently (ignoring edge labels)

$$m ::= \textbf{return}\, v \mid \textsf{op}\, \langle m_1,\ \ldots,\ m_k \rangle$$

Equivalently (accounting for edge labels)

$$m ::= \textbf{return}\, v \mid \textsf{op}\, (\lambda x.\textbf{case}\, x \{v_1 \mapsto m_1;\ \ldots;\ v_k \mapsto m_k\})$$

Examples

Boolean state

$$\mathsf{toggle} = \mathsf{get}\,\langle \mathsf{put}_{\mathsf{false}}\,\langle \mathbf{return}\,\mathsf{true}\rangle,\ \mathsf{put}_{\mathsf{true}}\,\langle \mathbf{return}\,\mathsf{false}\rangle\rangle$$

$$\mathbf{let}\ s = \mathsf{get}\,()\ \mathbf{in}\ \mathsf{put}\,(\mathsf{not}\ s);\ s$$

Examples

Boolean state

$$\mathsf{toggle} = \mathsf{get}\,\langle \mathsf{put}_{\mathsf{false}}\,\langle \mathbf{return}\,\mathsf{true}\rangle,\ \mathsf{put}_{\mathsf{true}}\,\langle \mathbf{return}\,\mathsf{false}\rangle\rangle$$

$$\mathbf{let}\ s = \mathsf{get}\,()\ \mathbf{in}\ \mathsf{put}\,(\mathsf{not}\ s);\ s$$

Natural number state

$$\mathsf{increment} = \mathsf{get}\,\langle \mathsf{put}_1\,\langle \mathbf{return}\,()\rangle,\ \ldots,\ \mathsf{put}_{i+1}\,\langle \mathbf{return}\,()\rangle,\ \ldots\rangle$$

$$\mathsf{put}\,(1 + \mathsf{get}\,())$$

Examples

Boolean state

$$\mathsf{toggle} = \mathsf{get}\,\langle \mathsf{put}_{\mathsf{false}}\,\langle \mathbf{return}\,\mathsf{true}\rangle,\ \mathsf{put}_{\mathsf{true}}\,\langle \mathbf{return}\,\mathsf{false}\rangle\rangle$$

$$\mathbf{let}\ s = \mathsf{get}\,()\ \mathbf{in}\ \mathsf{put}\,(\mathsf{not}\ s);\ s$$

Natural number state

$$\mathsf{increment} = \mathsf{get}\,\langle \mathsf{put}_1\,\langle \mathbf{return}\,()\rangle,\ \ldots,\ \mathsf{put}_{i+1}\,\langle \mathbf{return}\,()\rangle,\ \ldots\rangle$$

$$\mathsf{put}\,(1 + \mathsf{get}\,())$$

Nondeterminism

$$\mathsf{drunkToss} = \mathsf{choose}\,\langle \mathsf{choose}\,\langle \mathbf{return}\,\mathsf{Heads},\ \mathbf{return}\,\mathsf{Tails}\rangle,\ \mathsf{fail}\langle\rangle\rangle$$

$$\mathbf{if}\ \mathsf{choose}\,()\ \mathbf{then}\ (\mathbf{if}\ \mathsf{choose}\,()\ \mathbf{then}\ \mathsf{Heads}\ \mathbf{else}\ \mathsf{Tails})\ \mathbf{else}\ \mathsf{fail}\,()$$

Command-response trees as free monads

- A computation of type $\mathsf{comp}\ A$ is a tree whose leaves have type A
- Return is **return**
- Bind perfoms substitution at the leaves

$$\begin{aligned} \mathbf{return}\ v \gg= r &= r\ v \\ \mathsf{op}\ \langle m_1, \ldots, m_n \rangle \gg= r &= \mathsf{op}\ \langle m_1 \gg= r, \ldots, m_n \gg= r \rangle \end{aligned}$$

Algebraic effects

An algebraic effect is given by

1. a **signature** of operations
2. a collection of **equations**

Algebraic effects

An algebraic effect is given by

1. a **signature** of operations
2. a collection of **equations**

Example: boolean state

Signature

$$\begin{aligned} \mathsf{get} &: \mathsf{bool} \\ \mathsf{put}_{\mathsf{true}} &: 1 \\ \mathsf{put}_{\mathsf{false}} &: 1 \end{aligned}$$

Algebraic effects

An algebraic effect is given by

1. a **signature** of operations
2. a collection of **equations**

Example: boolean state

Signature

$$\begin{aligned} \mathsf{get} &: \mathsf{bool} \\ \mathsf{put}_{\mathsf{true}} &: 1 \\ \mathsf{put}_{\mathsf{false}} &: 1 \end{aligned}$$

Equations

$$\begin{aligned}
\mathsf{put}_s \langle \mathsf{put}_{s'} \langle m \rangle \rangle &\simeq \mathsf{put}_{s'} \langle m \rangle && \text{(put-put)} \\
\mathsf{put}_s \langle \mathsf{get} \langle m_{\mathsf{true}}, m_{\mathsf{false}} \rangle \rangle &\simeq \mathsf{put}_s \langle m_s \rangle && \text{(put-get)} \\
\mathsf{get} \langle \mathsf{put}_{\mathsf{true}} \langle m \rangle, \mathsf{put}_{\mathsf{false}} \langle n \rangle \rangle &\simeq \mathsf{get} \langle m, n \rangle && \text{(get-put)} \\
\mathsf{get} \langle \mathsf{get} \langle m, m' \rangle, \mathsf{get} \langle n', n \rangle \rangle &\simeq \mathsf{get} \langle m, n \rangle && \text{(get-get)}
\end{aligned}$$

Aside: the (get-get) equation is redundant

$$
\begin{array}{ll}
 & \mathsf{get}\,\langle \mathsf{get}\,\langle m, m' \rangle, \mathsf{get}\,\langle n', n \rangle \rangle \\
\simeq & \text{(get-put)} \\
 & \mathsf{get}\,\langle \mathsf{put}_{\mathsf{true}}\,\langle \mathsf{get}\,\langle m, m' \rangle \rangle, \mathsf{put}_{\mathsf{false}}\,\langle \mathsf{get}\,\langle n', n \rangle \rangle \rangle \\
\simeq & \text{(put-get)} \times 2 \\
 & \mathsf{get}\,\langle \mathsf{put}_{\mathsf{true}}\,\langle m \rangle, \mathsf{put}_{\mathsf{false}}\,\langle n \rangle \rangle \\
\simeq & \text{(get-put)} \\
 & \mathsf{get}\,\langle m, n \rangle
\end{array}
$$

Interpreting algebraic effects

Example: boolean state

Standard interpretation ($[\![\mathsf{comp}\,A]\!] = \mathsf{bool} \to [\![A]\!] \times \mathsf{bool}$)

$$
\begin{aligned}
[\![\mathbf{return}\; v]\!] &= \lambda s.([\![v]\!], s) \\
[\![\mathsf{get}\;\langle m, n\rangle]\!] &= \lambda s.\mathbf{if}\; s\; \mathbf{then} [\![m]\!] s\; \mathbf{else}\; [\![n]\!] s \\
[\![\mathsf{put}_{s'}\;\langle m\rangle]\!] &= \lambda s.[\![m]\!] s'
\end{aligned}
$$

Discard interpretation ($[\![\mathsf{comp}\,A]\!] = \mathsf{bool} \to [\![A]\!]$)

$$
\begin{aligned}
[\![\mathbf{return}\; v]\!] &= \lambda s.[\![v]\!] \\
[\![\mathsf{get}\;\langle m, n\rangle]\!] &= \lambda s.\mathbf{if}\; s\; \mathbf{then} [\![m]\!] s\; \mathbf{else}\; [\![n]\!] s \\
[\![\mathsf{put}_{s'}\;\langle m\rangle]\!] &= \lambda s.[\![m]\!] s'
\end{aligned}
$$

Logging interpretation ($[\![\mathsf{comp}\,A]\!] = \mathsf{bool} \to [\![A]\!] \times \mathsf{list}\,\mathsf{bool}$)

$$
\begin{aligned}
[\![\mathbf{return}\; v]\!] &= \lambda s.([\![v]\!], [s]) \\
[\![\mathsf{get}\;\langle m, n\rangle]\!] &= \lambda s.\mathbf{if}\; s\; \mathbf{then} [\![m]\!] s\; \mathbf{else}\; [\![n]\!] s \\
[\![\mathsf{put}_{s'}\;\langle m\rangle]\!] &= \lambda s.\mathbf{let}\; (x, ss) \leftarrow [\![m]\!] s'\; \mathbf{in}\; (x, s :: ss)
\end{aligned}
$$

Example: boolean state, standard interpretation

$$[\![\mathsf{comp}\,A]\!] = \mathsf{bool} \to [\![A]\!] \times \mathsf{bool}$$

$$\begin{aligned}
[\![\mathbf{return}\,v]\!] &= \lambda s.([\![v]\!], s)\\
[\![\mathsf{get}\,\langle m, n\rangle]\!] &= \lambda s.\mathbf{if}\,s\,\mathbf{then}\,[\![m]\!]s\,\mathbf{else}\,[\![n]\!]s\\
[\![\mathsf{put}_{s'}\,\langle m\rangle]\!] &= \lambda s.[\![m]\!]s'
\end{aligned}$$

Sound and complete with respect to the equations

$$m \simeq n \iff [\![m]\!] = [\![n]\!]$$

Bit toggling

$$[\![\mathsf{toggle}]\!] = \lambda s.\mathbf{if}\,s\,\mathbf{then}\,(\mathsf{true}, \mathsf{false})\,\mathbf{else}\,(\mathsf{false}, \mathsf{true})$$

Example: boolean state, discard interpretation

$$[\![\mathsf{comp}\,A]\!] = \mathsf{bool} \to [\![A]\!]$$

$$\begin{aligned}
[\![\mathbf{return}\,v]\!] &= \lambda s.[\![v]\!] \\
[\![\mathsf{get}\,\langle m, n\rangle]\!] &= \lambda s.\mathbf{if}\,s\,\mathbf{then}[\![m]\!]s\,\mathbf{else}\,[\![n]\!]s \\
[\![\mathsf{put}_{s'}\,\langle m\rangle]\!] &= \lambda s.[\![m]\!]s'
\end{aligned}$$

Sound with respect to the equations

$$m \simeq n \implies [\![m]\!] = [\![n]\!]$$

Not complete because:

$$[\![\mathsf{put}_s\,\langle \mathbf{return}\,v\rangle]\!] = [\![\mathbf{return}\,v]\!]$$

Bit toggling

$$[\![\mathsf{toggle}]\!] = \lambda s.\mathbf{if}\,s\,\mathbf{then}\,\mathsf{true}\,\mathbf{else}\,\mathsf{false} = \lambda s.s$$

Example: boolean state, logging interpretation

$$[\![\mathsf{comp}\,A]\!] = \mathsf{bool} \to [\![A]\!] \times \mathsf{list\,bool}$$

$$\begin{aligned}
[\![\mathbf{return}\,v]\!] &= \lambda s.([\![v]\!],[s]) \\
[\![\mathsf{get}\,\langle m,n\rangle]\!] &= \lambda s.\mathbf{if}\,s\,\mathbf{then}\,[\![m]\!]s\,\mathbf{else}\,[\![n]\!]s \\
[\![\mathsf{put}_{s'}\,\langle m\rangle]\!] &= \lambda s.\mathbf{let}\,(x,ss) \leftarrow [\![m]\!]s'\,\mathbf{in}\,(x,s :: ss)
\end{aligned}$$

Complete with respect to the equations

$$m \simeq n \quad \Longleftarrow \quad [\![m]\!] = [\![n]\!]$$

Not sound because:

$$[\![\mathsf{put}_s\,\langle \mathsf{put}_{s'}\,\langle m\rangle\rangle]\!] \neq [\![\mathsf{put}_{s'}\,\langle m\rangle]\!]$$
$$[\![\mathsf{get}\,\langle \mathsf{put}_{\mathsf{true}}\,\langle m\rangle, \mathsf{put}_{\mathsf{false}}\,\langle n\rangle\rangle]\!] \neq [\![\mathsf{get}\,\langle m,n\rangle]\!]$$

Bit toggling

$$[\![\mathsf{toggle}]\!] = \lambda s.\mathbf{if}\,s\,\mathbf{then}\,(\mathsf{true},[\mathsf{true},\mathsf{false}])\,\mathbf{else}\,(\mathsf{false},[\mathsf{false},\mathsf{true}])$$

Algebraic effects without equations

Different interpretations are useful in practice

So we will adopt **free** algebraic effects — no equations

Algebraic computations are command-response trees modulo equations

Abstract computations are plain command-response trees

Different interpretations give different meanings to the same abstract computation

Interpretations as effect handlers

Example: boolean state

Meta level interpretation (enumerated continuations)

$$
\begin{aligned}
[\![\mathbf{return}\, v]\!] &= \lambda s.([\![v]\!], s) \\
[\![\mathsf{get}\,\langle m, n\rangle]\!] &= \lambda s.\mathbf{if}\ s\ \mathbf{then} [\![m]\!] s\ \mathbf{else}\ [\![n]\!] s \\
[\![\mathsf{put}_{s'}\,\langle m\rangle]\!] &= \lambda s.[\![m]\!] s'
\end{aligned}
$$

Meta level interpretation (continuations as functions)

$$
\begin{aligned}
[\![\mathbf{return}\, v]\!] &= \lambda s.([\![v]\!], s) \\
[\![\mathsf{get}\, k]\!] &= \lambda s.[\![k\, s]\!]\, s \\
[\![\mathsf{put}_{s'}\, k]\!] &= \lambda s.[\![k\, ()]\!]\, s'
\end{aligned}
$$

Object level effect handler

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

Interpretations as effect handlers

Example: nondeterminism

Meta level interpretation (enumerated continuations)

$$
\begin{aligned}
[\![\mathbf{return}\, v]\!] &= [[\![v]\!]] \\
[\![\textsf{choose}\ \langle m, n\rangle]\!] &= [\![m]\!] +\!\!+ [\![n]\!] \\
[\![\textsf{fail}\ \langle\rangle]\!] &= []
\end{aligned}
$$

Meta level interpretation (continuations as functions)

$$
\begin{aligned}
[\![\mathbf{return}\, v]\!] &= [[\![v]\!]] \\
[\![\textsf{choose}\, k]\!] &= [\![k\, \textsf{true}]\!] +\!\!+ [\![k\, \textsf{false}]\!] \\
[\![\textsf{fail}\, k]\!] &= []
\end{aligned}
$$

Object level effect handler

$$
\begin{array}{ll}
\mathbf{return}\, v & \mapsto [v] \\
\langle \textsf{choose}\ () \to r\rangle & \mapsto r\, \textsf{true} +\!\!+ r\, \textsf{false} \\
\langle \textsf{fail}\ () \to r\rangle & \mapsto []
\end{array}
$$

Example: choice and failure

Effect signature

$$\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{bool},\ \ \mathsf{fail} : a.1 \twoheadrightarrow a\}$$

Example: choice and failure

Effect signature

$$\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{bool},\ \ \mathsf{fail} : a.1 \twoheadrightarrow a\}$$

Drunk coin tossing

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

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

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

Example: choice and failure

Handlers

$\mathsf{maybeFail} =$ — exception handler

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

Example: choice and failure

Handlers

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

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

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

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

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

Example: choice and failure

Handlers

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

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

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

Example: choice and failure

Handlers

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

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

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

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

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

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

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

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

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

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

Example: 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

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

$\quad \langle \mathsf{choose}\, () \to r \rangle \mapsto r\, \mathsf{true} \mathbin{+\!\!+} r\, \mathsf{false}$

Example: 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

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

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

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

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

Example: choice and failure

Handlers

maybeFail = — exception handler

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

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

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

trueChoice = — linear handler

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

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

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

allChoices = — non-linear handler

$$\begin{array}{lll} \textbf{return}\, x & \mapsto & [x] \\ \langle \mathsf{choose}\,() \to r\rangle & \mapsto & r\, \mathsf{true} \mathbin{+\!\!+} r\, \mathsf{false} \end{array}$$

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

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

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

Example: 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

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

$\quad \langle \mathsf{choose}\,() \to r\rangle \mapsto r\, \mathsf{true} \mathbin{+\!\!+} r\, \mathsf{false}$

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

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

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

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

Operational semantics

Reduction rules

$$\textbf{handle } V \textbf{ with } H \rightsquigarrow N[V/x]$$

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

where

$$\begin{aligned} H = \textbf{return } x &\mapsto N \\ \mathsf{op}_1\ p\ r &\mapsto N_{\mathsf{op}_1} \\ &\dots \\ \mathsf{op}_k\ p\ r &\mapsto N_{\mathsf{op}_k} \end{aligned}$$

Evaluation contexts

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

Typing rules

Effects

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

Computations

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

Operations

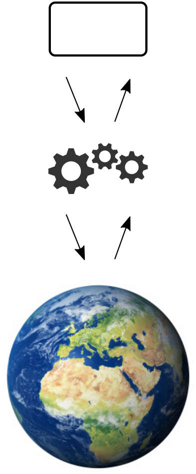
$$\frac{\Gamma \vdash V : A}{\Gamma \vdash \mathsf{op}\ V : B!(E \uplus \{\mathsf{op} : A \twoheadrightarrow B\})}$$

Handlers

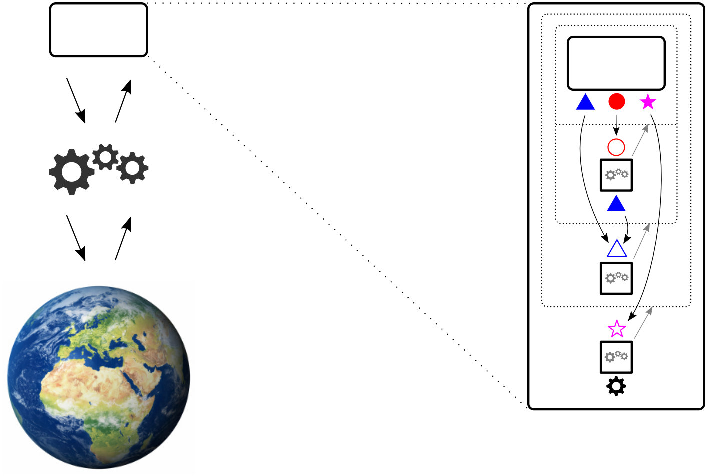
$$\frac{\Gamma \vdash M : C \qquad \Gamma \vdash H : C \Rightarrow D}{\Gamma \vdash \textbf{handle}\ M\ \textbf{with}\ H : D}$$

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

Effect handlers as composable user-defined operating systems



Effect handlers as composable user-defined operating systems



Example: cooperative concurrency

Effect signature

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

Example: cooperative concurrency

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: cooperative concurrency

Effect signature

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

Two cooperative lightweight threads

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

Handler — parameterised handler

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

Example: cooperative concurrency

Effect signature

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

Two cooperative lightweight threads

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

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: cooperative concurrency

Effect signature

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

Two cooperative lightweight threads

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

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

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

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

A1 B1 A2 B2

Operational semantics (parameterised handlers)

Reduction rules

$$\textbf{handle}\ V\ \textbf{with}\ H\ W \leadsto N[V/x, W/h]$$

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

where

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

Evaluation contexts

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

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

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

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

Example: cooperative concurrency with UNIX-style fork

Effect signature

$$\{\mathsf{yield} : 1 \twoheadrightarrow 1, \;\; \mathsf{ufork} : 1 \twoheadrightarrow \mathsf{bool}\}$$

Example: cooperative concurrency with UNIX-style fork

Effect signature

$$\{\textsf{yield} : 1 \twoheadrightarrow 1,\ \ \textsf{ufork} : 1 \twoheadrightarrow \textsf{bool}\}$$

A single cooperative program

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

Example: cooperative concurrency with UNIX-style fork

Effect signature

$$\{\mathsf{yield} : 1 \twoheadrightarrow 1,\ \ \mathsf{ufork} : 1 \twoheadrightarrow \mathsf{bool}\}$$

A single cooperative program

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

Handler

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

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

Example: cooperative concurrency with UNIX-style fork

Effect signature

$$\{\mathsf{yield} : 1 \twoheadrightarrow 1, \ \ \mathsf{ufork} : 1 \twoheadrightarrow \mathsf{bool}\}$$

A single cooperative program

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

Handler

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

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

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

M1 A1 M2 B1 A2 M3 B2

Example: cooperative concurrency with UNIX-style fork

Effect signature

$$\{\mathsf{yield} : 1 \twoheadrightarrow 1,\ \ \mathsf{ufork} : 1 \twoheadrightarrow \mathsf{bool}\}$$

A single cooperative program

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

Handler

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

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

Example: cooperative concurrency with UNIX-style fork

Effect signature

$$\{\mathsf{yield} : 1 \twoheadrightarrow 1, \quad \mathsf{ufork} : 1 \twoheadrightarrow \mathsf{bool}\}$$

A single cooperative program

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

Handler

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

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

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

M1 M2 M3 A1 B1 A2 B2

Effect handler oriented programming languages

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

Resources

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: generators

Effect signature

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

Example: generators

Effect signature

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

A simple generator

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

Example: generators

Effect signature

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

A simple generator

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

Handler

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

Example: generators

Effect signature

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

A simple generator

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

Handler

$$\begin{aligned}
&\mathsf{until}\, stop = \quad \text{— affine handler}\\
&\quad \mathbf{return}\,() \qquad \mapsto []\\
&\quad \langle \mathsf{send}\, n \to r\rangle \mapsto \mathbf{if}\ n < stop\ \mathbf{then}\ n :: r\, stop\,()\\
&\qquad\qquad\qquad\qquad\ \mathbf{else}\ []
\end{aligned}$$

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

Example: cooperative concurrency with higher-order fork

Effect signature — recursive effect signature

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

Example: cooperative concurrency with higher-order fork

Effect signature — recursive effect signature

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

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: cooperative concurrency with higher-order fork

Effect signature — recursive effect signature

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

A single cooperative program

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

Handler — scoped handler

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

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

Example: cooperative concurrency with higher-order fork

Effect signature — recursive effect signature

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

A single cooperative program

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

Handler — scoped handler

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

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

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

M1 A1 M2 B1 A2 M3 B2

Example: cooperative concurrency with higher-order fork

Effect signature — recursive effect signature

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

A single cooperative program

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

Handler — scoped handler

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

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

Example: cooperative concurrency with higher-order fork

Effect signature — recursive effect signature

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

A single cooperative program

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

Handler — scoped handler

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

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

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

M1 M2 M3 A1 B1 A2 B2

Built-in effects

Console I/O

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

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

Generative state

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

Example: actors

Process ids

$$\mathsf{Pid}\, a = \mathsf{Ref}\, (\mathsf{list}\, a)$$

Effect signature

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

Example: actors

Process ids

$$\mathsf{Pid}\,a = \mathsf{Ref}\,(\mathsf{list}\,a)$$

Effect signature

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

An actor chain

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

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

Example: actors

Actors via cooperative concurrency

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathsf{act}\ \mathit{mine} =} \\
\quad \mathbf{return}\ () & \mapsto () \\
\quad \langle \mathsf{self}\ () \to r \rangle & \mapsto r\ \mathit{mine}\ \mathit{mine} \\
\quad \langle \mathsf{spawn}\ \mathit{you} \to r \rangle & \mapsto \mathbf{let}\ \mathit{yours} = \mathsf{new}\ []\ \mathbf{in} \\
 & \quad\ \ \mathsf{fork}\ (\lambda().\mathsf{act}\ \mathit{yours}\ (\mathit{you}\ ()));\ r\ \mathit{mine}\ \mathit{yours} \\
\quad \langle \mathsf{send}\ (m, \mathit{yours}) \to r \rangle & \mapsto \mathbf{let}\ \mathit{ms} = \mathsf{read}\ \mathit{yours}\ \mathbf{in} \\
 & \quad\ \ \mathsf{write}\ (\mathit{yours}, \mathit{ms} +\!\!+ [m]);\ r\ \mathit{mine}\ () \\
\quad \langle \mathsf{recv}\ () \to r \rangle & \mapsto \mathbf{case}\ \mathsf{read}\ \mathit{mine}\ \mathbf{of} \\
 & \qquad\qquad [] \qquad\quad\ \mapsto \mathsf{yield}\ ();\ r\ \mathit{mine}\ (\mathsf{recv}\ ()) \\
 & \qquad\qquad (m :: \mathit{ms}) \mapsto \mathsf{write}\ (\mathit{mine}, \mathit{ms});\ r\ \mathit{mine}\ m
\end{array}
$$

Example: actors

Actors via cooperative concurrency

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

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

....ping!

Example: pipes
Effect signatures

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

Example: pipes

Effect signatures

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

A producer and a consumer

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

Example: pipes

Effect signatures

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

A producer and a consumer

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

Pipes and copipes as shallow handlers

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

Example: pipes

Effect signatures

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

A producer and a consumer

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

Pipes and copipes as shallow handlers

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

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

Example: pipes

Effect signatures

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

A producer and a consumer

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

Pipes and copipes as shallow handlers

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

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

Exercise: implement pipes using deep handlers

Small-step operational semantics for shallow effect handlers

Reduction rules

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

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

Evaluation contexts

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

Small-step operational semantics for shallow effect handlers

Reduction rules

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

$$\begin{aligned}
\text{where } H = \textbf{return}\ x \quad &\mapsto N_{\mathsf{ret}} \\
\langle \mathsf{op_1}\ p \to r \rangle \quad &\mapsto N_{\mathsf{op_1}} \\
&\ldots \\
\langle \mathsf{op_k}\ p \to r \rangle \quad &\mapsto N_{\mathsf{op_k}}
\end{aligned}$$

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

Example: pipes using multihandlers

Effect signatures

$\mathsf{Sender} = \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\} \qquad \mathsf{Receiver} = \{\mathsf{receive} : 1 \twoheadrightarrow \mathsf{Nat}\} \qquad \mathsf{Fail} = \{\mathsf{fail} : a.1 \twoheadrightarrow a\}$

Example: pipes using multihandlers

Effect signatures

$\mathsf{Sender} = \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\}$ $\mathsf{Receiver} = \{\mathsf{receive} : 1 \twoheadrightarrow \mathsf{Nat}\}$ $\mathsf{Fail} = \{\mathsf{fail} : a.1 \twoheadrightarrow a\}$

A producer and a consumer

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

Example: pipes using multihandlers

Effect signatures

$$\mathsf{Sender} = \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\} \qquad \mathsf{Receiver} = \{\mathsf{receive} : 1 \twoheadrightarrow \mathsf{Nat}\} \qquad \mathsf{Fail} = \{\mathsf{fail} : a.1 \twoheadrightarrow a\}$$

A producer and a consumer

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

A pipe multihandler

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

Example: pipes using multihandlers

Effect signatures

$\mathsf{Sender} = \{\mathsf{send} : \mathsf{Nat} \twoheadrightarrow 1\} \qquad \mathsf{Receiver} = \{\mathsf{receive} : 1 \twoheadrightarrow \mathsf{Nat}\} \qquad \mathsf{Fail} = \{\mathsf{fail} : a.1 \twoheadrightarrow a\}$

A producer and a consumer

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

A pipe multihandler

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

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