Algebraic effects and effect handlers

Sam Lindley

The University of Edinburgh

SPLV 2024

Quiz

Quiz

What is an effect?

Quiz

What is an effect?

What is a pure computation?

Quiz

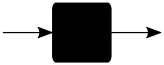
What is an effect?

What is a pure computation?

What is an effectful computation?

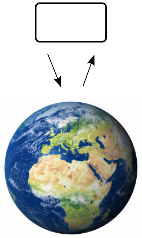
Effects

Programs as black boxes (Church-Turing model)?



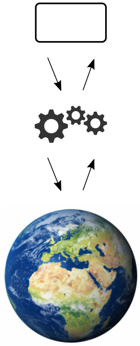
Effects

Programs must interact with their environment



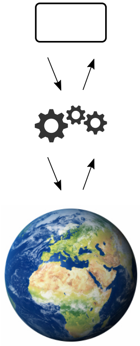
Effects

Programs must interact with their environment



Effects

Programs must interact with their environment

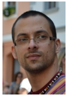


Effects are pervasive

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

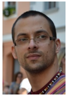
Typically ad hoc and hard-wired

Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009 (and ETAPS 2022 test of time award)

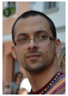
Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009 (and ETAPS 2022 test of time award)

Monads $\longrightarrow$ Algebraic Effects $\longrightarrow$ Effect Handlers

Effect handlers

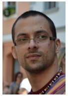
 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009 (and ETAPS 2022 test of time award)

Monads $\longrightarrow$ Algebraic Effects $\longrightarrow$ Effect Handlers

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

Effect handlers

 Gordon Plotkin  Matija Pretnar

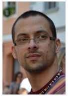
Handlers of algebraic effects, ESOP 2009 (and ETAPS 2022 test of time award)

Monads $\longrightarrow$ Algebraic Effects $\longrightarrow$ Effect Handlers

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

Give programmer direct access to **context**
(c.f. resumable exceptions, monads, delimited control)

Effect handlers

 Gordon Plotkin  Matija Pretnar

Handlers of algebraic effects, ESOP 2009 (and ETAPS 2022 test of time award)

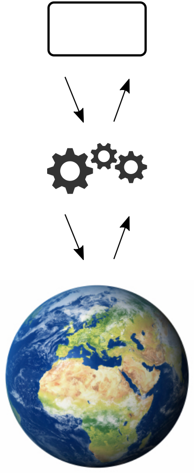
Monads $\longrightarrow$ Algebraic Effects $\longrightarrow$ Effect Handlers

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

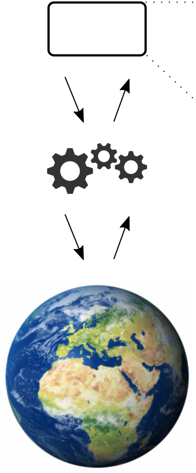
Give programmer direct access to **context**
(c.f. resumable exceptions, monads, delimited control)

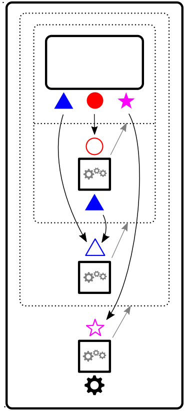
Effect handlers in practice:
OCaml 5, GitHub (Semantic), Meta (React), Uber (Pyro), WasmFX, ...

Effect handlers as composable user-defined operating systems

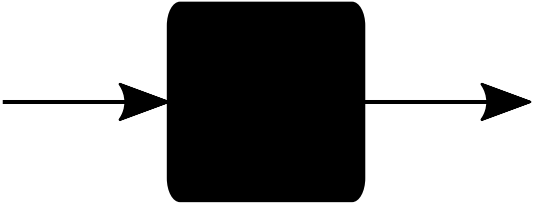


Effect handlers as composable user-defined operating systems

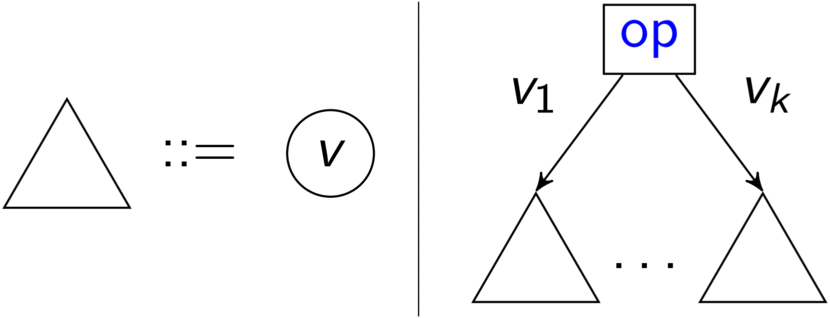




Pure computations

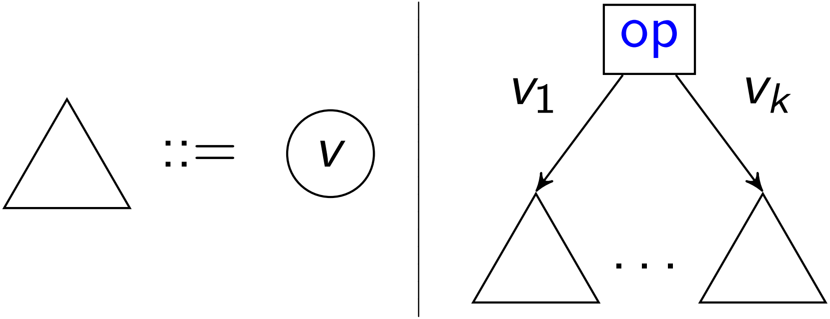


Effectful computations



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

Effectful computations

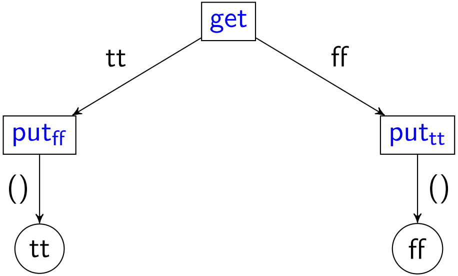


A **command-response** tree (aka **interaction tree**)

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

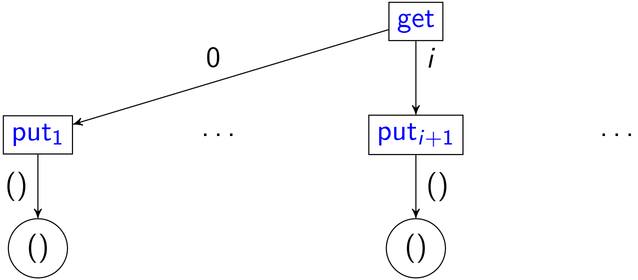
Example: boolean state (bit toggling)

$$\begin{aligned} \mathsf{get} &: \mathsf{Bool} \\ \mathsf{put}_{\mathsf{tt}} &: 1 \\ \mathsf{put}_{\mathsf{ff}} &: 1 \end{aligned}$$



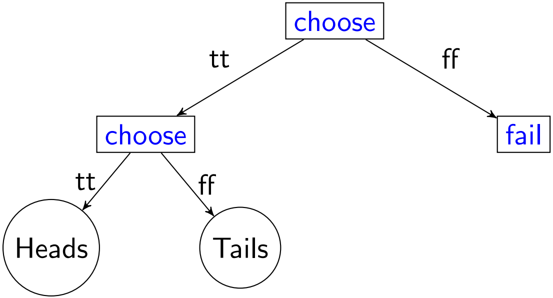
Example: natural number state (increment)

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

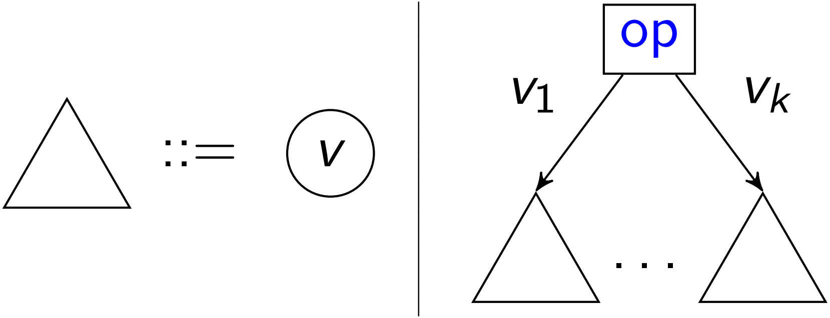


Example: nondeterminism (drunk coin toss)

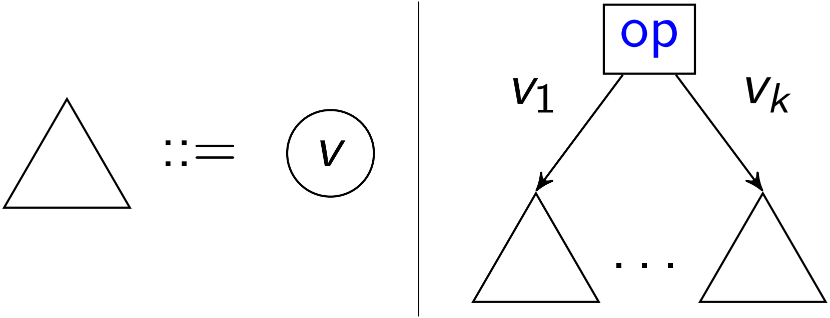
$$\begin{array}{lcl} \text{choose} & : & \text{Bool} \\ \text{fail} & : & 0 \end{array}$$



What is an effectful computation?



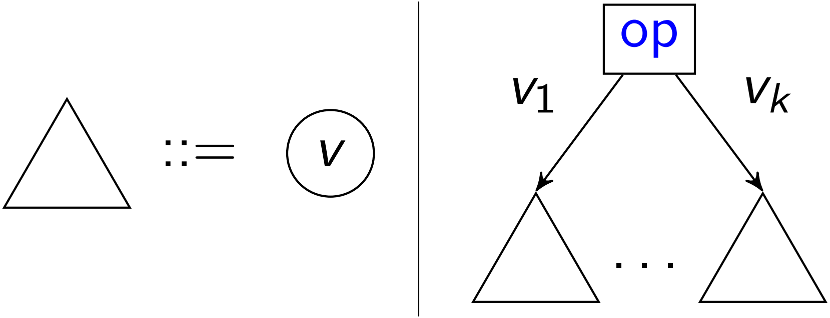
What is an effectful computation?



Equivalently (ignoring edge labels)

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

What is an effectful computation?



Equivalently (ignoring edge labels)

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

Equivalently (accounting for edge labels)

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

Examples

Boolean state

$$\mathrm{toggle} = \mathsf{get}\,\langle \mathsf{put}_{\mathrm{ff}}\,\langle \mathbf{return}\,\mathrm{tt}\rangle,\ \mathsf{put}_{\mathrm{tt}}\,\langle \mathbf{return}\,\mathrm{ff}\rangle\rangle$$

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

Examples

Boolean state

$$\text{toggle} = \text{get}\,\langle \text{put}_{\text{ff}}\,\langle \textbf{return}\,\text{tt}\rangle,\ \text{put}_{\text{tt}}\,\langle \textbf{return}\,\text{ff}\rangle\rangle$$

$$\textbf{let}\ s = \text{get}\,()\ \textbf{in}\ \text{put}\,(\text{not}\ s);\ s$$

Natural number state

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

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

Examples

Boolean state

$$\text{toggle} = \text{get}\,\langle \text{put}_{\text{ff}}\,\langle \textbf{return}\,\text{tt}\rangle,\ \text{put}_{\text{tt}}\,\langle \textbf{return}\,\text{ff}\rangle\rangle$$

$$\textbf{let}\ s = \text{get}\,()\ \textbf{in}\ \text{put}\,(\text{not}\ s);\ s$$

Natural number state

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

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

Nondeterminism

$$\text{drunkToss} = \text{choose}\,\langle \text{choose}\,\langle \textbf{return}\,\text{Heads},\ \textbf{return}\,\text{Tails}\rangle,\ \text{fail}\langle\rangle\rangle$$

$$\textbf{if}\ \text{choose}\,()\ \textbf{then}\ (\textbf{if}\ \text{choose}\,()\ \textbf{then}\ \text{Heads}\ \textbf{else}\ \text{Tails})\ \textbf{else}\ \textbf{absurd}\ (\text{fail}\,())$$

Command-response trees as free monads

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

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

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

$$\begin{aligned} \mathsf{get} &: \mathsf{Bool} \\ \mathsf{put}_{\mathsf{tt}} &: 1 \\ \mathsf{put}_{\mathsf{ff}} &: 1 \end{aligned}$$

Algebraic effects

An algebraic effect is given by

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

Example: boolean state

Signature

$$
\begin{aligned}
\mathsf{get} &: \mathsf{Bool} \\
\mathsf{put}_{\mathsf{tt}} &: 1 \\
\mathsf{put}_{\mathsf{ff}} &: 1
\end{aligned}
$$

Equations

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

Aside: the (get-get) equation is redundant

$$\mathrm{get}\,\langle \mathrm{get}\,\langle m, m' \rangle, \mathrm{get}\,\langle n', n \rangle \rangle$$

Aside: the (get-get) equation is redundant

$$\begin{aligned} & \mathsf{get}\,\langle \mathsf{get}\,\langle m, m' \rangle, \mathsf{get}\,\langle n', n \rangle \rangle \\ \simeq \quad & \text{(get-put)} \\ & \mathsf{get}\,\langle \mathsf{put}_{\mathsf{tt}}\,\langle \mathsf{get}\,\langle \mathsf{get}\,\langle m, m' \rangle, \mathsf{get}\,\langle n', n \rangle \rangle \rangle, \mathsf{put}_{\mathsf{ff}}\,\langle \mathsf{get}\,\langle \mathsf{get}\,\langle m, m' \rangle, \mathsf{get}\,\langle n', n \rangle \rangle \rangle \rangle \end{aligned}$$

Aside: the (get-get) equation is redundant

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

Aside: the (get-get) equation is redundant

$$
\begin{aligned}
& \mathsf{get}\,\langle \mathsf{get}\,\langle m, m' \rangle, \mathsf{get}\,\langle n', n \rangle \rangle \\
\simeq \quad & \text{(get-put)} \\
& \mathsf{get}\,\langle \mathsf{put}_{\mathsf{tt}}\,\langle \mathsf{get}\,\langle \mathsf{get}\,\langle m, m' \rangle, \mathsf{get}\,\langle n', n \rangle \rangle \rangle, \mathsf{put}_{\mathsf{ff}}\,\langle \mathsf{get}\,\langle \mathsf{get}\,\langle m, m' \rangle, \mathsf{get}\,\langle n', n \rangle \rangle \rangle \rangle \\
\simeq \quad & \text{(put-get)} \times 2 \\
& \mathsf{get}\,\langle \mathsf{put}_{\mathsf{tt}}\,\langle \mathsf{get}\,\langle m, m' \rangle \rangle, \mathsf{put}_{\mathsf{ff}}\,\langle \mathsf{get}\,\langle n', n \rangle \rangle \rangle \\
\simeq \quad & \text{(put-get)} \times 2 \\
& \mathsf{get}\,\langle \mathsf{put}_{\mathsf{tt}}\,\langle m \rangle, \mathsf{put}_{\mathsf{ff}}\,\langle n \rangle \rangle
\end{aligned}
$$

Aside: the (get-get) equation is redundant

$$
\begin{aligned}
& \mathsf{get}\,\langle \mathsf{get}\,\langle m, m' \rangle, \mathsf{get}\,\langle n', n \rangle \rangle \\
\simeq \quad & \text{(get-put)} \\
& \mathsf{get}\,\langle \mathsf{put}_{\mathsf{tt}}\,\langle \mathsf{get}\,\langle \mathsf{get}\,\langle m, m' \rangle, \mathsf{get}\,\langle n', n \rangle \rangle \rangle, \mathsf{put}_{\mathsf{ff}}\,\langle \mathsf{get}\,\langle \mathsf{get}\,\langle m, m' \rangle, \mathsf{get}\,\langle n', n \rangle \rangle \rangle \rangle \\
\simeq \quad & \text{(put-get)} \times 2 \\
& \mathsf{get}\,\langle \mathsf{put}_{\mathsf{tt}}\,\langle \mathsf{get}\,\langle m, m' \rangle \rangle, \mathsf{put}_{\mathsf{ff}}\,\langle \mathsf{get}\,\langle n', n \rangle \rangle \rangle \\
\simeq \quad & \text{(put-get)} \times 2 \\
& \mathsf{get}\,\langle \mathsf{put}_{\mathsf{tt}}\,\langle m \rangle, \mathsf{put}_{\mathsf{ff}}\,\langle n \rangle \rangle \\
\simeq \quad & \text{(put-get)} \times 2 \\
& \mathsf{get}\,\langle \mathsf{put}_{\mathsf{tt}}\,\langle \mathsf{get}\,\langle m, n \rangle \rangle, \mathsf{put}_{\mathsf{ff}}\,\langle \mathsf{get}\,\langle m, n \rangle \rangle \rangle
\end{aligned}
$$

Aside: the (get-get) equation is redundant

$$
\begin{aligned}
& \mathsf{get}\,\langle \mathsf{get}\,\langle m, m' \rangle, \mathsf{get}\,\langle n', n \rangle \rangle \\
\simeq \quad & \text{(get-put)} \\
& \mathsf{get}\,\langle \mathsf{put}_{\mathsf{tt}}\,\langle \mathsf{get}\,\langle \mathsf{get}\,\langle m, m' \rangle, \mathsf{get}\,\langle n', n \rangle \rangle \rangle, \mathsf{put}_{\mathsf{ff}}\,\langle \mathsf{get}\,\langle \mathsf{get}\,\langle m, m' \rangle, \mathsf{get}\,\langle n', n \rangle \rangle \rangle \rangle \\
\simeq \quad & \text{(put-get)} \times 2 \\
& \mathsf{get}\,\langle \mathsf{put}_{\mathsf{tt}}\,\langle \mathsf{get}\,\langle m, m' \rangle \rangle, \mathsf{put}_{\mathsf{ff}}\,\langle \mathsf{get}\,\langle n', n \rangle \rangle \rangle \\
\simeq \quad & \text{(put-get)} \times 2 \\
& \mathsf{get}\,\langle \mathsf{put}_{\mathsf{tt}}\,\langle m \rangle, \mathsf{put}_{\mathsf{ff}}\,\langle n \rangle \rangle \\
\simeq \quad & \text{(put-get)} \times 2 \\
& \mathsf{get}\,\langle \mathsf{put}_{\mathsf{tt}}\,\langle \mathsf{get}\,\langle m, n \rangle \rangle, \mathsf{put}_{\mathsf{ff}}\,\langle \mathsf{get}\,\langle m, n \rangle \rangle \rangle \\
\simeq \quad & \text{(get-put)} \\
& \mathsf{get}\,\langle m, n \rangle
\end{aligned}
$$

Interpreting algebraic effects

Example: boolean state

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

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

Interpreting algebraic effects

Example: boolean state

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

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

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

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

Interpreting algebraic effects

Example: boolean state

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

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

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

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

Logging interpretation ($[\![\text{comp}\, A]\!] = \text{Bool} \to [\![A]\!] \times \text{List Bool}$)

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

Example: boolean state, standard interpretation

$$[\![\text{comp}\ A]\!] = \text{Bool} \to [\![A]\!] \times \text{Bool}$$

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

Sound and complete with respect to the equations

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

Bit toggling

$$[\![\text{toggle}]\!] = \lambda s.\mathbf{if}\ s\ \mathbf{then}\ (\text{tt}, \text{ff})\ \mathbf{else}\ (\text{ff}, \text{tt})$$

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

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

Not complete because:

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

Bit toggling

$$[\![\mathsf{toggle}]\!] = \lambda s.\mathbf{if}\; s\; \mathbf{then}\; \mathsf{tt}\; \mathbf{else}\; \mathsf{ff} = \lambda s.s$$

Example: boolean state, logging interpretation

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

Complete with respect to the equations

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

Not sound because:

$$
\begin{aligned}
[\![\mathsf{put}_s\,\langle \mathsf{put}_{s'}\,\langle m\rangle\rangle]\!] &\neq [\![\mathsf{put}_{s'}\,\langle m\rangle]\!] \\
[\![\mathsf{get}\,\langle \mathsf{put}_{\mathsf{tt}}\,\langle m\rangle, \mathsf{put}_{\mathsf{ff}}\,\langle n\rangle\rangle]\!] &\neq [\![\mathsf{get}\,\langle m, n\rangle]\!]
\end{aligned}
$$

Bit toggling

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

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 — standard interpretation

Meta level interpretation (enumerated continuations)

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

Interpretations as effect handlers

Example: boolean state — standard interpretation

Meta level interpretation (enumerated continuations)

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

Meta level interpretation (continuations as functions)

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

Interpretations as effect handlers

Example: boolean state — standard interpretation

Meta level interpretation (enumerated continuations)

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

Meta level interpretation (continuations as functions)

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

Object level effect handler

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

Interpretations as effect handlers

Example: nondeterminism

Meta level interpretation (enumerated continuations)

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

Interpretations as effect handlers

Example: nondeterminism

Meta level interpretation (enumerated continuations)

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

Meta level interpretation (continuations as functions)

$$
\begin{aligned}
[\![\textbf{return}\ v]\!] &= [[\![v]\!]] \\
[\![\mathsf{choose}\ k]\!] &= [\![k\ \mathsf{tt}]\!] +\!\!+ [\![k\ \mathsf{ff}]\!] \\
[\![\mathsf{fail}\ k]\!] &= []
\end{aligned}
$$

Interpretations as effect handlers

Example: nondeterminism

Meta level interpretation (enumerated continuations)

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

Meta level interpretation (continuations as functions)

$$\begin{aligned} [\![\mathbf{return}\; v]\!] &= [[\![v]\!]] \\ [\![\mathsf{choose}\; k]\!] &= [\![k\;\mathsf{tt}]\!] +\!\!+ [\![k\;\mathsf{ff}]\!] \\ [\![\mathsf{fail}\; k]\!] &= [] \end{aligned}$$

Object level effect handler

$$\begin{aligned} &\mathbf{return}\; v && \mapsto [v] \\ &\langle\mathsf{choose}\,() \to r\rangle && \mapsto r\;\mathsf{tt} +\!\!+ r\;\mathsf{ff} \\ &\langle\mathsf{fail}\,() \to r\rangle && \mapsto [] \end{aligned}$$

Parameterised operations (term parameters)

For convenience we write

$$\mathsf{op} : A \twoheadrightarrow B \qquad \text{instead of} \qquad \mathsf{op}_i : B, \quad i \in A$$

and to perform an operation:

$$\mathsf{op}\ i \qquad \text{instead of} \qquad \mathsf{op}_i$$

Parameterised operations (term parameters)

For convenience we write

$$\mathsf{op} : A \twoheadrightarrow B \qquad \text{instead of} \qquad \mathsf{op}_i : B, \quad i \in A$$

and to perform an operation:

$$\mathsf{op}\ i \qquad \text{instead of} \qquad \mathsf{op}_i$$

For uniformity we parameterise all operations in effect signatures.

Parameterised operations (term parameters)

For convenience we write

$$\mathsf{op} : A \twoheadrightarrow B \qquad \text{instead of} \qquad \mathsf{op}_i : B, \quad i \in A$$

and to perform an operation:

$$\mathsf{op}\ i \qquad \text{instead of} \qquad \mathsf{op}_i$$

For uniformity we parameterise all operations in effect signatures.

Examples

natural number state	$\{\mathsf{put} : \mathsf{Nat} \twoheadrightarrow 1, \quad \mathsf{get} : 1 \twoheadrightarrow \mathsf{Nat}\}$
boolean state	$\{\mathsf{put} : \mathsf{Bool} \twoheadrightarrow 1, \quad \mathsf{get} : 1 \twoheadrightarrow \mathsf{Bool}\}$
nondeterminism	$\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}, \quad \mathsf{fail} : 1 \twoheadrightarrow 0\}$

Parametric operations (type parameters)

It can be useful to parameterise an operation by one or more types.

Example: nondeterminism

$$\{\mathsf{choose} : 1 \twoheadrightarrow \mathsf{Bool}, \quad \mathsf{fail} : 1 \twoheadrightarrow 0\}$$

becomes

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

and

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

becomes

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

Example: choice and failure

Effect signature

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

Example: choice and failure

Effect signature

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

Drunk coin tossing

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

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

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

Example: choice and failure

Handlers

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

Example: choice and failure

Handlers

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

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

Example: choice and failure

Handlers

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

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

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

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

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

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

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

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

Example: choice and failure

Handlers

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

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

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

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

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

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

Example: choice and failure

Handlers

$\text{maybeFail} =$ — exception handler

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

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

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

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

$\text{trueChoice} =$ — linear handler

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

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

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

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

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

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

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

Example: choice and failure

Handlers

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Example: choice and failure

Handlers

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

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

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

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

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

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

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

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

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

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

Example: choice and failure

Handlers

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

$\quad \textbf{return}\, x \mapsto \text{Just}\, x \qquad \textbf{handle}\ 42\ \textbf{with}\ \text{maybeFail} \Longrightarrow \text{Just}\, 42$

$\quad \langle \text{fail}\, () \rangle \mapsto \text{Nothing} \qquad \textbf{handle}\ \text{fail}\, ()\ \textbf{with}\ \text{maybeFail} \Longrightarrow \text{Nothing}$

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

$\quad \textbf{return}\, x \mapsto x \qquad \textbf{handle}\ 42\ \textbf{with}\ \text{trueChoice} \Longrightarrow 42$

$\quad \langle \text{choose}\, () \to r \rangle \mapsto r\, \text{tt} \qquad \textbf{handle}\ \text{toss}\, ()\ \textbf{with}\ \text{trueChoice} \Longrightarrow \text{Heads}$

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

$\quad \textbf{return}\, x \mapsto [x] \qquad \textbf{handle}\ 42\ \textbf{with}\ \text{allChoices} \Longrightarrow [42]$

$\quad \langle \text{choose}\, () \to r \rangle \mapsto r\, \text{tt} +\!\!+\ r\, \text{ff} \qquad \textbf{handle}\ \text{toss}\, ()\ \textbf{with}\ \text{allChoices} \Longrightarrow [\text{Heads}, \text{Tails}]$

$$\textbf{handle}\ (\textbf{handle}\ \text{drunkTosses}\, 2\ \textbf{with}\ \text{maybeFail})\ \textbf{with}\ \text{allChoices} \Longrightarrow$$
$$[\text{Just}\,[\text{Heads}, \text{Heads}],\ \text{Just}\,[\text{Heads}, \text{Tails}],\ \text{Nothing},$$
$$\text{Just}\,[\text{Tails}, \text{Heads}],\ \text{Just}\,[\text{Tails}, \text{Tails}],\ \text{Nothing},$$
$$\text{Nothing}]$$

Operational semantics

Reduction rules

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

where

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

Evaluation contexts

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

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

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

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

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

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

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

Exercise: Adapt the typing rules to accommodate parametric operations

What is an effect handler?

What is an effect handler?

- A **modular** interpreter for effectful computations

What is an effect handler?

- A **modular** interpreter for effectful computations
- A generalisation of an exception handler
  - based on exceptional syntax [Benton and Kennedy, 2001]

$\textbf{let}\ x = (\textbf{try}\ M\ \textbf{with}\ H)\ \textbf{in}\ N$	conventional exception handlers
$\Downarrow$	
$\textbf{try}\ M\ \textbf{as}\ x\ \textbf{in}\ N\ \textbf{otherwise}\ H$	exceptional syntax
$\Downarrow$	
$\textbf{handle}\ M\ \textbf{with}\ \{\textbf{return}\ x \mapsto N; H\}$	effect handlers

    success continuations aid composition, optimisation, and reasoning
  - resumable

What is an effect handler?

- A **modular** interpreter for effectful computations
- A generalisation of an exception handler
  - based on exceptional syntax [Benton and Kennedy, 2001]

$\textbf{let}\ x = (\textbf{try}\ M\ \textbf{with}\ H)\ \textbf{in}\ N$	conventional exception handlers
$\Downarrow$	
$\textbf{try}\ M\ \textbf{as}\ x\ \textbf{in}\ N\ \textbf{otherwise}\ H$	exceptional syntax
$\Downarrow$	
$\textbf{handle}\ M\ \textbf{with}\ \{\textbf{return}\ x \mapsto N; H\}$	effect handlers

    success continuations aid composition, optimisation, and reasoning
  - resumable
- A morphism between (free) algebras

What is an effect handler?

- A **modular** interpreter for effectful computations
- A generalisation of an exception handler
  - based on exceptional syntax [Benton and Kennedy, 2001]

$\textbf{let}\ x = (\textbf{try}\ M\ \textbf{with}\ H)\ \textbf{in}\ N$	conventional exception handlers
$\Downarrow$	
$\textbf{try}\ M\ \textbf{as}\ x\ \textbf{in}\ N\ \textbf{otherwise}\ H$	exceptional syntax
$\Downarrow$	
$\textbf{handle}\ M\ \textbf{with}\ \{\textbf{return}\ x \mapsto N; H\}$	effect handlers

    success continuations aid composition, optimisation, and reasoning
  - resumable
- A morphism between (free) algebras
- A fold (catamorphism) over a command-response tree

What is an effect handler?

- A **modular** interpreter for effectful computations
- A generalisation of an exception handler
  - based on exceptional syntax [Benton and Kennedy, 2001]

$\mathbf{let}\ x = (\mathbf{try}\ M\ \mathbf{with}\ H)\ \mathbf{in}\ N$	conventional exception handlers
$\Downarrow$	
$\mathbf{try}\ M\ \mathbf{as}\ x\ \mathbf{in}\ N\ \mathbf{otherwise}\ H$	exceptional syntax
$\Downarrow$	
$\mathbf{handle}\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N; H\}$	effect handlers

    success continuations aid composition, optimisation, and reasoning
  - resumable
- A morphism between (free) algebras
- A fold (catamorphism) over a command-response tree
- A structured delimited control operator

What is an effect handler?

- A **modular** interpreter for effectful computations
- A generalisation of an exception handler
  - based on exceptional syntax [Benton and Kennedy, 2001]

$\mathbf{let}\ x = (\mathbf{try}\ M\ \mathbf{with}\ H)\ \mathbf{in}\ N$	conventional exception handlers
$\Downarrow$	
$\mathbf{try}\ M\ \mathbf{as}\ x\ \mathbf{in}\ N\ \mathbf{otherwise}\ H$	exceptional syntax
$\Downarrow$	
$\mathbf{handle}\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N; H\}$	effect handlers

    success continuations aid composition, optimisation, and reasoning
  - resumable
- A morphism between (free) algebras
- A fold (catamorphism) over a command-response tree
- A structured delimited control operator
- A composable user-defined operating system

Example: cooperative concurrency (parameterised handler)

Effect signature

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

Example: cooperative concurrency (parameterised handler)

Effect signature

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

Two cooperative lightweight threads

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

Example: cooperative concurrency (parameterised handler)

Effect signature

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

Two cooperative lightweight threads

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

Handler — parameterised handler

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

Example: cooperative concurrency (parameterised handler)

Effect signature

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

Two cooperative lightweight threads

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

Handler — parameterised handler

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

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 (parameterised handler)

Effect signature

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

Two cooperative lightweight threads

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

Handler — parameterised handler

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

Helpers

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

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

A1 B1 A2 B2

Operational semantics (parameterised handlers)

Reduction rules

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

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

Evaluation contexts

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

Operational semantics (parameterised handlers)

Reduction rules

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

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

Evaluation contexts

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

Exercise: express parameterised handlers as deep handlers

Typing rules (parameterised handlers)

Effects

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

Computations

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

Operations

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

Handlers

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

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

Example: cooperative concurrency with UNIX-style fork

Effect signature

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

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

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

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

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

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

Example: cooperative concurrency with UNIX-style fork

Effect signature

$$\{\text{yield} : 1 \twoheadrightarrow 1, \quad \text{ufork} : 1 \twoheadrightarrow \text{Bool}\}$$

A single cooperative program

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

Handler

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

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

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

M1 A1 M2 B1 A2 M3 B2

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

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

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

Example: cooperative concurrency with UNIX-style fork

Effect signature

$$\{\text{yield} : 1 \twoheadrightarrow 1, \quad \text{ufork} : 1 \twoheadrightarrow \text{Bool}\}$$

A single cooperative program

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

Handler

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

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

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

M1 M2 M3 A1 B1 A2 B2

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

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

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

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

M1 M2 M3 A1 B1 A2 B2

Exercise: implement a handler for a fork operation that uses the resumption linearly

Example: cooperative concurrency (shallow handler)

Effect signature

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

Example: cooperative concurrency (shallow handler)

Effect signature

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

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: cooperative concurrency (shallow handler)

Effect signature

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

Two cooperative lightweight threads

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

Handler

$$\begin{aligned}
&\text{cooperate}\,[] = () \\
&\text{cooperate}\,(t :: ts) = \\
&\quad \textbf{handle}\; t()\; \textbf{with} \\
&\qquad \textbf{return}\,() \qquad\quad \mapsto \text{cooperate}\,(ts) \\
&\qquad \langle \text{yield}\,() \to t \rangle \mapsto \text{cooperate}\,(ts \mathbin{+\!\!+} [t])
\end{aligned}$$

Example: cooperative concurrency (shallow handler)

Effect signature

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

Two cooperative lightweight threads

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

Handler

$$\begin{array}{l}
\text{cooperate}\,[] = ()\\
\text{cooperate}\,(t :: ts) =\\
\quad \textbf{handle}\; t()\; \textbf{with}\\
\qquad \begin{array}{ll}
\textbf{return}\,() & \mapsto \text{cooperate}\,(ts)\\
\langle \text{yield}\,() \to t \rangle & \mapsto \text{cooperate}\,(ts \mathbin{+\!\!+} [t])
\end{array}
\end{array}$$

$$\text{cooperate}\,[tA, tB]$$

Example: cooperative concurrency (shallow handler)

Effect signature

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

Two cooperative lightweight threads

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

Handler

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

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

A1 B1 A2 B2

Operational semantics (shallow handlers)

Reduction rules

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

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

Evaluation contexts

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

Operational semantics (shallow handlers)

Reduction rules

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

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

Evaluation contexts

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

Exercise: express shallow handlers as deep handlers

Typing rules (shallow handlers)

Effects

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

Computations

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

Operations

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

Handlers

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

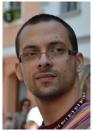
$$\frac{\Gamma, x : A \vdash N : D \qquad [\mathsf{op}_i : A_i \twoheadrightarrow B_i \in E]_i \qquad [\Gamma, p : A_i, r : B_i \to A!E \vdash N_i : D]_i}{\Gamma \vdash \begin{array}{l}\textbf{return}\ x \mapsto N \\ (\langle \mathsf{op}_i\ p \to r\rangle \mapsto N_i)_i\end{array} : A!E \Rightarrow D}$$

Effect handler oriented programming languages

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

Resources

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



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

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

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