This work is supported by

THE UNIVERSITY *of* EDINBURGH

School of informatics

EPSRC Centre for Doctoral Training in Pervasive Parallelism

EPSRC

Engineering and Physical Sciences Research Council

UNIVERSITY OF CAMBRIDGE

Continuation Passing Style for Effect Handlers

Formal Structures for Computation and Deduction

Daniel Hillerström [1] Sam Lindley [1]
Robert Atkey [2] KC Sivaramakrishnan [3]

[1]The University of Edinburgh, UK

[2]University of Strathclyde, UK

[3]University of Cambridge, UK

September 5, 2017

Algebraic effects by example: a drunk coin toss

Consider two effects: nondeterminism and exceptions

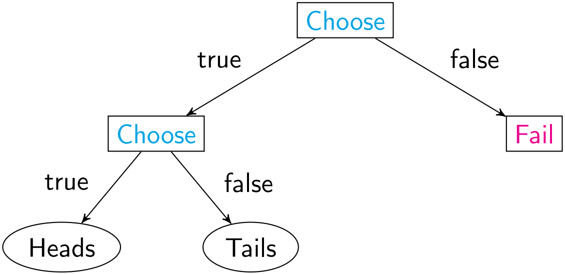
$$
\begin{aligned}
&\mathsf{drunkToss} : \mathsf{Toss} \mathrel{!} \{\mathsf{Choose} : \mathsf{Bool}; \mathsf{Fail} : \mathsf{Zero}\} \\
&\mathsf{drunkToss} = \textbf{if do } \mathsf{Choose} \textbf{ then} \\
&\qquad\qquad\qquad \textbf{if do } \mathsf{Choose} \textbf{ then } \mathsf{Heads} \textbf{ else } \mathsf{Tails} \\
&\qquad\qquad\quad \textbf{else absurd do } \mathsf{Fail}
\end{aligned}
$$

Algebraic effects by example: a drunk coin toss

Consider two effects: nondeterminism and exceptions

drunkToss : Toss ! {Choose : Bool; Fail : Zero}
drunkToss = **if do** Choose **then**
    if do Choose **then** Heads **else** Tails
  else absurd do Fail

Drunk toss computation tree



Effect handlers by example: modular interpretations

Choose handler

$$\mathsf{allChoices} : \alpha \; ! \; \{\mathsf{Choose} : \mathsf{Bool}; \rho\} \Rightarrow \mathsf{List}\ \alpha \; ! \; \{\rho\}$$
$$\begin{aligned}\mathsf{allChoices} = \;& \mathbf{return}\ x \mapsto [x] \\ & \mathsf{Choose}\ r \mapsto r\ \mathsf{true} \mathbin{+\!\!+} r\ \mathsf{false}\end{aligned}$$

Fail handler

$$\mathsf{fail} : \alpha \; ! \; \{\mathsf{Fail} : \mathsf{Zero}; \rho\} \Rightarrow \mathsf{List}\ \alpha \; ! \; \{\rho\}$$
$$\begin{aligned}\mathsf{fail} = \;& \mathbf{return}\ x \mapsto [x] \\ & \mathsf{Fail}\ r \mapsto []\end{aligned}$$

Effect handlers by example: modular interpretations

Choose handler

$$\text{allChoices} : \alpha \ ! \ \{\text{Choose} : \text{Bool}; \rho\} \Rightarrow \text{List } \alpha \ ! \ \{\rho\}$$
$$\begin{aligned}\text{allChoices} = \ & \textbf{return } x \mapsto [x] \\ & \text{Choose } r \mapsto r \text{ true} \mathbin{+\!\!+} r \text{ false}\end{aligned}$$

Fail handler

$$\text{fail} : \alpha \ ! \ \{\text{Fail} : \text{Zero}; \rho\} \Rightarrow \text{List } \alpha \ ! \ \{\rho\}$$
$$\begin{aligned}\text{fail} = \ & \textbf{return } x \mapsto [x] \\ & \text{Fail } r \mapsto []\end{aligned}$$

Two possible interpretations:

$$\begin{aligned}& \textbf{handle } (\textbf{handle } \text{drunkToss } \textbf{with } \text{fail}) \textbf{ with } \text{allChoices} \\ & \Rightarrow [[\text{Heads}], [\text{Tails}], []]\end{aligned}$$

$$\begin{aligned}& \textbf{handle } (\textbf{handle } \text{drunkToss } \textbf{with } \text{allChoices}) \textbf{ with } \text{fail} \\ & \Rightarrow []\end{aligned}$$

Effect handlers by example: modular interpretations

Choose handler

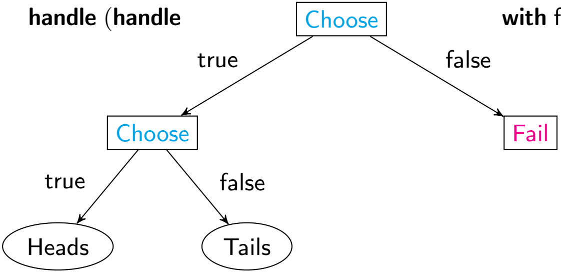
$\text{allChoices} : \alpha \; ! \; \{\text{Choose} : \text{Bool}; \rho\} \Rightarrow \text{List } \alpha \; ! \; \{\rho\}$
$\text{allChoices} = \textbf{return } x \mapsto [x]$
$\qquad\qquad\quad \text{Choose } r \mapsto r \text{ true} +\!\!+ \; r \text{ false}$

Fail handler

$\text{fail} : \alpha \; ! \; \{\text{Fail} : \text{Zero}; \rho\} \Rightarrow \text{List } \alpha \; ! \; \{\rho\}$
$\text{fail} = \textbf{return } x \mapsto [x]$
$\qquad\quad \text{Fail } r \mapsto []$

Two possible interpretations:

handle (**handle**



with fail) **with** allChoices

Effect handlers by example: modular interpretations

Choose handler

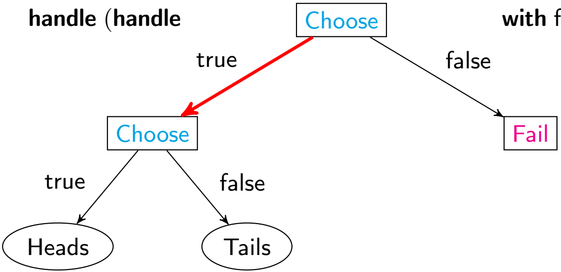
$\mathsf{allChoices} : \alpha \; ! \; \{\mathsf{Choose} : \mathsf{Bool}; \rho\} \Rightarrow \mathsf{List}\; \alpha \; ! \; \{\rho\}$
$\mathsf{allChoices} = \mathbf{return}\; x \mapsto [x]$
$\qquad\qquad\quad \mathsf{Choose}\; r \mapsto r\; \mathsf{true} +\!\!+\; r\; \mathsf{false}$

Fail handler

$\mathsf{fail} : \alpha \; ! \; \{\mathsf{Fail} : \mathsf{Zero}; \rho\} \Rightarrow \mathsf{List}\; \alpha \; ! \; \{\rho\}$
$\mathsf{fail} = \mathbf{return}\; x \mapsto [x]$
$\qquad\quad \mathsf{Fail}\; r \mapsto []$

Two possible interpretations:

handle (**handle**



with fail) **with** allChoices

Effect handlers by example: modular interpretations

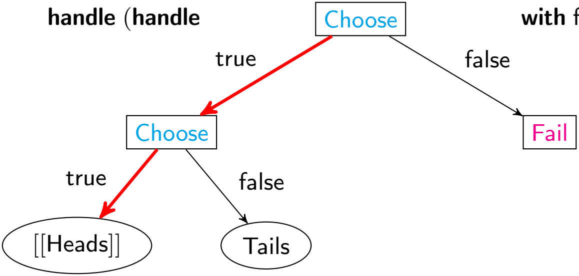
Choose handler

$$\mathsf{allChoices} : \alpha \; ! \; \{\mathsf{Choose} : \mathsf{Bool}; \rho\} \Rightarrow \mathsf{List}\; \alpha \; ! \; \{\rho\}$$
$$\begin{aligned}\mathsf{allChoices} = \; & \mathbf{return}\; x \mapsto [x] \\ & \mathsf{Choose}\; r \mapsto r\; \mathsf{true} +\!\!+\; r\; \mathsf{false}\end{aligned}$$

Fail handler

$$\mathsf{fail} : \alpha \; ! \; \{\mathsf{Fail} : \mathsf{Zero}; \rho\} \Rightarrow \mathsf{List}\; \alpha \; ! \; \{\rho\}$$
$$\begin{aligned}\mathsf{fail} = \; & \mathbf{return}\; x \mapsto [x] \\ & \mathsf{Fail}\; r \mapsto []\end{aligned}$$

Two possible interpretations:

handle (**handle**  **with** fail) **with** allChoices

Effect handlers by example: modular interpretations

Choose handler

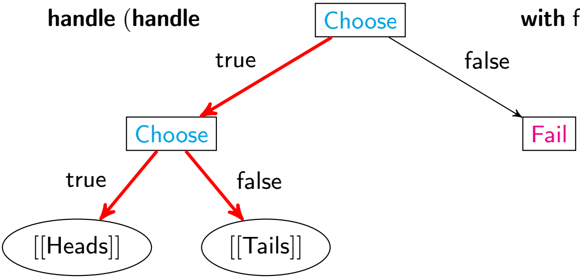
allChoices : α ! {Choose : Bool; ρ} $\Rightarrow$ List α ! {ρ}
allChoices = **return** $x \mapsto [x]$
Choose $r \mapsto r$ true ++ r false

Fail handler

fail : α ! {Fail : Zero; ρ} $\Rightarrow$ List α ! {ρ}
fail = **return** $x \mapsto [x]$
Fail $r \mapsto []$

Two possible interpretations:

handle (**handle** $\boxed{\text{Choose}}$ **with** fail) **with** allChoices



Effect handlers by example: modular interpretations

Choose handler

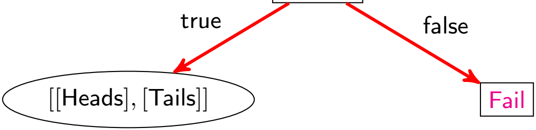
$\text{allChoices} : \alpha \ ! \ \{\text{Choose} : \text{Bool}; \rho\} \Rightarrow \text{List } \alpha \ ! \ \{\rho\}$
$\text{allChoices} = \textbf{return } x \mapsto [x]$
$\qquad\qquad\qquad \text{Choose } r \mapsto r \text{ true} +\!\!+ \ r \text{ false}$

Fail handler

$\text{fail} : \alpha \ ! \ \{\text{Fail} : \text{Zero}; \rho\} \Rightarrow \text{List } \alpha \ ! \ \{\rho\}$
$\text{fail} = \textbf{return } x \mapsto [x]$
$\qquad\quad \text{Fail } r \mapsto []$

Two possible interpretations:

handle (**handle** $\boxed{\text{Choose}}$ **with** fail) **with** allChoices



Effect handlers by example: modular interpretations

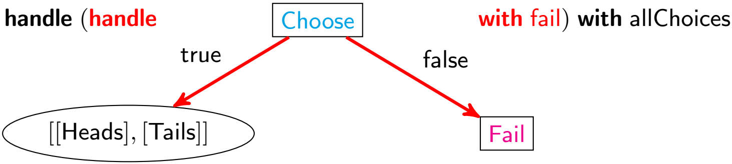
Choose handler

allChoices : α ! {Choose : Bool; ρ} $\Rightarrow$ List α ! {ρ}
allChoices = **return** $x \mapsto [x]$
Choose $r \mapsto r$ true ++ r false

Fail handler

fail : α ! {Fail : Zero; ρ} $\Rightarrow$ List α ! {ρ}
fail = **return** $x \mapsto [x]$
Fail $r \mapsto$ []

Two possible interpretations:



Effect handlers by example: modular interpretations

Choose handler

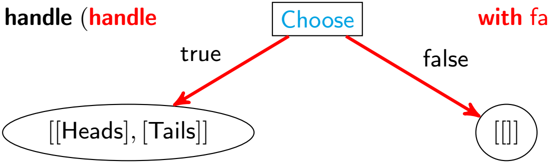
$$\begin{array}{l}\mathsf{allChoices} : \alpha \mathbin{!} \{\mathsf{Choose} : \mathsf{Bool}; \rho\} \Rightarrow \mathsf{List}\ \alpha \mathbin{!} \{\rho\} \\ \mathsf{allChoices} = \mathbf{return}\ x \mapsto [x] \\ \qquad\qquad\quad \mathsf{Choose}\ r \mapsto r\ \mathsf{true} \mathbin{+\!\!+} r\ \mathsf{false}\end{array}$$

Fail handler

$$\begin{array}{l}\mathsf{fail} : \alpha \mathbin{!} \{\mathsf{Fail} : \mathsf{Zero}; \rho\} \Rightarrow \mathsf{List}\ \alpha \mathbin{!} \{\rho\} \\ \mathsf{fail} = \mathbf{return}\ x \mapsto [x] \\ \qquad\quad \mathsf{Fail}\ r \mapsto []\end{array}$$

Two possible interpretations:

handle (**handle** ... **with** fail) **with** allChoices



Effect handlers by example: modular interpretations

Choose handler

$$\text{allChoices} : \alpha \ ! \ \{\text{Choose} : \text{Bool}; \rho\} \Rightarrow \text{List } \alpha \ ! \ \{\rho\}$$
$$\text{allChoices} = \textbf{return } x \mapsto [x]$$
$$\qquad\qquad \text{Choose } r \mapsto r \text{ true} +\!\!+ \ r \text{ false}$$

Fail handler

$$\text{fail} : \alpha \ ! \ \{\text{Fail} : \text{Zero}; \rho\} \Rightarrow \text{List } \alpha \ ! \ \{\rho\}$$
$$\text{fail} = \textbf{return } x \mapsto [x]$$
$$\qquad\qquad \text{Fail } r \mapsto []$$

Two possible interpretations:

[[Heads], [Tails], []]

Effect handlers by example: modular interpretations

Choose handler

$$
\begin{aligned}
&\mathsf{allChoices} : \alpha \,!\, \{\mathsf{Choose} : \mathsf{Bool}; \rho\} \Rightarrow \mathsf{List}\ \alpha \,!\, \{\rho\} \\
&\mathsf{allChoices} = \mathbf{return}\ x \mapsto [x] \\
&\qquad\qquad\quad \mathsf{Choose}\ r \mapsto r\ \mathsf{true} \mathbin{+\!\!+} r\ \mathsf{false}
\end{aligned}
$$

Fail handler

$$
\begin{aligned}
&\mathsf{fail} : \alpha \,!\, \{\mathsf{Fail} : \mathsf{Zero}; \rho\} \Rightarrow \mathsf{List}\ \alpha \,!\, \{\rho\} \\
&\mathsf{fail} = \mathbf{return}\ x \mapsto [x] \\
&\qquad\quad \mathsf{Fail}\ r \mapsto []
\end{aligned}
$$

Two possible interpretations:

$$
\begin{aligned}
&\mathbf{handle}\ (\mathbf{handle}\ \mathsf{drunkToss}\ \mathbf{with}\ \mathsf{allChoices})\ \mathbf{with}\ \mathsf{fail} \\
&\Rightarrow []
\end{aligned}
$$

Effect handlers by example: modular interpretations

Choose handler

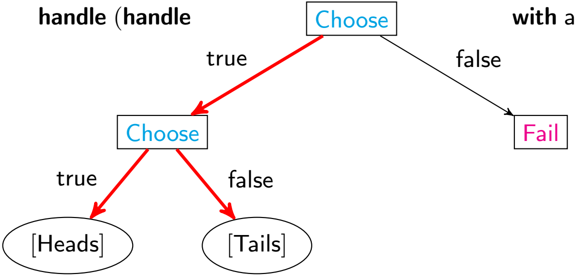
$\text{allChoices} : \alpha \; ! \; \{\text{Choose} : \text{Bool}; \rho\} \Rightarrow \text{List } \alpha \; ! \; \{\rho\}$
$\text{allChoices} = \textbf{return } x \mapsto [x]$
$\qquad\qquad\quad \text{Choose } r \mapsto r \text{ true} +\!\!+ \; r \text{ false}$

Fail handler

$\text{fail} : \alpha \; ! \; \{\text{Fail} : \text{Zero}; \rho\} \Rightarrow \text{List } \alpha \; ! \; \{\rho\}$
$\text{fail} = \textbf{return } x \mapsto [x]$
$\qquad\quad \text{Fail } r \mapsto []$

Two possible interpretations:

handle (**handle**   **with** allChoices) **with** fail



Effect handlers by example: modular interpretations

Choose handler

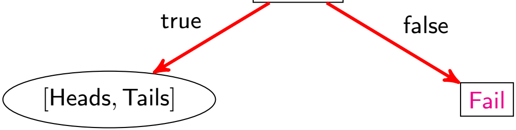
$\mathsf{allChoices} : \alpha \; ! \; \{\mathsf{Choose} : \mathsf{Bool}; \rho\} \Rightarrow \mathsf{List} \; \alpha \; ! \; \{\rho\}$
$\mathsf{allChoices} = \mathbf{return} \; x \mapsto [x]$
$\qquad\qquad\quad \mathsf{Choose} \; r \mapsto r \; \mathsf{true} \mathbin{+\!\!+} r \; \mathsf{false}$

Fail handler

$\mathsf{fail} : \alpha \; ! \; \{\mathsf{Fail} : \mathsf{Zero}; \rho\} \Rightarrow \mathsf{List} \; \alpha \; ! \; \{\rho\}$
$\mathsf{fail} = \mathbf{return} \; x \mapsto [x]$
$\qquad\quad \mathsf{Fail} \; r \mapsto []$

Two possible interpretations:

handle (**handle** $\boxed{\mathsf{Choose}}$ **with** allChoices) **with** fail



Effect handlers by example: modular interpretations

Choose handler

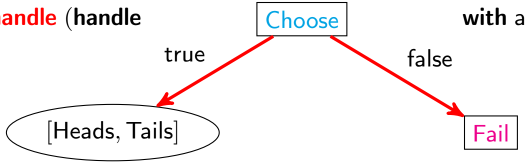
$$\mathsf{allChoices} : \alpha \; ! \; \{\mathsf{Choose} : \mathsf{Bool}; \rho\} \Rightarrow \mathsf{List} \; \alpha \; ! \; \{\rho\}$$
$$\mathsf{allChoices} = \textbf{return} \; x \mapsto [x]$$
$$\qquad\qquad \mathsf{Choose} \; r \mapsto r \; \mathsf{true} +\!\!+ \; r \; \mathsf{false}$$

Fail handler

$$\mathsf{fail} : \alpha \; ! \; \{\mathsf{Fail} : \mathsf{Zero}; \rho\} \Rightarrow \mathsf{List} \; \alpha \; ! \; \{\rho\}$$
$$\mathsf{fail} = \textbf{return} \; x \mapsto [x]$$
$$\qquad\qquad \mathsf{Fail} \; r \mapsto []$$

Two possible interpretations:

handle (**handle** … **with** allChoices) **with** fail



Effect handlers by example: modular interpretations

Choose handler

$$\mathsf{allChoices} : \alpha \; ! \; \{\mathsf{Choose} : \mathsf{Bool}; \rho\} \Rightarrow \mathsf{List}\; \alpha \; ! \; \{\rho\}$$
$$\begin{aligned}\mathsf{allChoices} = \; & \mathbf{return}\; x \mapsto [x] \\ & \mathsf{Choose}\; r \mapsto r\; \mathsf{true} +\!\!+\; r\; \mathsf{false}\end{aligned}$$

Fail handler

$$\mathsf{fail} : \alpha \; ! \; \{\mathsf{Fail} : \mathsf{Zero}; \rho\} \Rightarrow \mathsf{List}\; \alpha \; ! \; \{\rho\}$$
$$\begin{aligned}\mathsf{fail} = \; & \mathbf{return}\; x \mapsto [x] \\ & \mathsf{Fail}\; r \mapsto []\end{aligned}$$

Two possible interpretations:

$[]$

Operational semantics for effect handlers

The operational semantics for handlers is fairly simple

$$\begin{array}{l}\textbf{handle}\ (\textbf{return}\ V)\ \textbf{with}\ H \rightsquigarrow N[V/x],\\ \qquad \text{where } H^{\mathrm{ret}} = \{\textbf{return}\ x \mapsto N\}\end{array}$$

$$\begin{array}{l}\textbf{handle}\ \mathcal{E}[\textbf{do}\ \ell\ V]\ \textbf{with}\ H \rightsquigarrow N[V/p, \lambda y.\,\textbf{handle}\ \mathcal{E}[\textbf{return}\ y]\ \textbf{with}\ H/r],\\ \qquad \text{where } \ell \notin BL(\mathcal{E}) \text{ and } H^{\ell} = \{\ell\ p\ r \mapsto N\}\end{array}$$

The set of bound labels, BL, is defined inductively

$$BL([\ ]) = \emptyset \quad BL(\textbf{let}\ x \leftarrow \mathcal{E}\ \textbf{in}\ N) = BL(\mathcal{E}) \quad BL(\textbf{handle}\ \mathcal{E}\ \textbf{with}\ H) = BL(\mathcal{E}) \cup \mathrm{dom}(H)$$

Evaluation contexts, $\mathcal{E}$, are standard

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

Why continuation passing style?

So, effect handlers are great! Now, how do we compile them?

Why continuation passing style (CPS)?

- CPS is good for implementing control flow
- CPS is formulated on a restricted subset of λ-calculus
- The Links compiler (Cooper, Lindley, Wadler, and Yallop 2006)
  - Server-side uses a CEK machine
  - Client-side uses CPS

Source and target calculi

Our source calculus is $\lambda^{\rho}_{\text{eff}}$ (Hillerström and Lindley 2016).

$$
\begin{array}{lrl}
\text{Values} & V, W & ::= x \mid \lambda x.\, M \mid \langle\rangle \mid \langle \ell = V; W \rangle \mid (\ell\, V) \\
\text{Computations} & M, N & ::= V\, W \\
& & \mid\ \ \textbf{let}\ \langle \ell = x; y \rangle = V\ \textbf{in}\ N \mid \textbf{case}\ V \{\ell\, x \mapsto M; y \mapsto N\} \mid \textbf{absurd}\, V \\
& & \mid\ \ \textbf{return}\ V \mid \textbf{let}\ x \leftarrow M\ \textbf{in}\ N \\
& & \mid\ \ \textbf{do}\ \ell\ V \mid \textbf{handle}\ M\ \textbf{with}\ H \\
\text{Handlers} & H & ::= \{\textbf{return}\ x \mapsto M\} \mid \{\ell\ p\ r \mapsto M\} \uplus H\}
\end{array}
$$

Our target is an untyped λ-calculus

$$
\begin{array}{lrl}
\text{Values} & V, W & ::= x \mid \lambda x.M \mid \langle\rangle \mid \langle \ell = V; W \rangle \mid \ell\ V \\
\text{Computations} & M, N & ::= V \mid M\, W \mid \textbf{let}\ \langle \ell = x; y \rangle = V\ \textbf{in}\ N \\
& & \mid\ \ \textbf{case}\ V\ \{\ell\ x \mapsto M; y \mapsto N\} \mid \textbf{absurd}\ V
\end{array}
$$

Source and target calculi

Our source calculus is $\lambda^{\rho}_{\text{eff}}$ (Hillerström and Lindley 2016).

$$
\begin{array}{lrl}
\text{Values} & V, W ::= & x \mid \lambda x.\, M \mid \langle\rangle \mid \langle \ell = V; W \rangle \mid (\ell\, V) \\
\text{Computations} & M, N ::= & V\, W \\
 & \mid & \mathbf{let}\ \langle \ell = x; y \rangle = V\ \mathbf{in}\ N \mid \mathbf{case}\ V \{\ell\ x \mapsto M; y \mapsto N\} \mid \mathbf{absurd}\, V \\
 & \mid & \boxed{\mathbf{return}\ V \mid \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N} \\
 & \mid & \mathbf{do}\ \ell\ V \mid \mathbf{handle}\ M\ \mathbf{with}\ H \\
\text{Handlers} & H ::= & \{\mathbf{return}\ x \mapsto M\} \mid \{\ell\ p\ r \mapsto M\} \uplus H\}
\end{array}
$$

Our target is an untyped λ-calculus

$$
\begin{array}{lrl}
\text{Values} & V, W ::= & x \mid \lambda x.M \mid \langle\rangle \mid \langle \ell = V; W \rangle \mid \ell\ V \\
\text{Computations} & M, N ::= & V \mid M\ W \mid \mathbf{let}\ \langle \ell = x; y \rangle = V\ \mathbf{in}\ N \\
 & \mid & \mathbf{case}\ V\ \{\ell\ x \mapsto M; y \mapsto N\} \mid \mathbf{absurd}\ V
\end{array}
$$

Source and target calculi

Our source calculus is $\lambda^{\rho}_{\text{eff}}$ (Hillerström and Lindley 2016).

$$
\begin{array}{lll}
\text{Values} & V, W & ::= x \mid \lambda x.\, M \mid \langle\rangle \mid \langle \ell = V; W\rangle \mid (\ell\, V) \\
\text{Computations} & M, N & ::= V\, W \\
& & \quad \mid \ \textbf{let}\ \langle \ell = x; y\rangle = V\ \textbf{in}\ N \mid \textbf{case}\ V\{\ell\ x \mapsto M; y \mapsto N\} \mid \textbf{absurd}\, V \\
& & \quad \mid \ \textbf{return}\ V \mid \textbf{let}\ x \leftarrow M\ \textbf{in}\ N \\
& & \quad \mid \ \boxed{\textbf{do}\ \ell\ V \mid \textbf{handle}\ M\ \textbf{with}\ H} \\
\text{Handlers} & H & ::= \{\textbf{return}\ x \mapsto M\} \mid \{\ell\ p\ r \mapsto M\} \uplus H\}
\end{array}
$$

Our target is an untyped λ-calculus

$$
\begin{array}{lll}
\text{Values} & V, W & ::= x \mid \lambda x.M \mid \langle\rangle \mid \langle \ell = V; W\rangle \mid \ell\ V \\
\text{Computations} & M, N & ::= V \mid M\ W \mid \textbf{let}\ \langle \ell = x; y\rangle = V\ \textbf{in}\ N \\
& & \quad \mid \ \textbf{case}\ V\ \{\ell\ x \mapsto M; y \mapsto N\} \mid \textbf{absurd}\ V
\end{array}
$$

Source and target calculi

Our source calculus is $\lambda_{\text{eff}}^{\rho}$ (Hillerström and Lindley 2016).

$$
\begin{array}{lrcl}
\text{Values} & V, W & ::= & x \mid \lambda x.\, M \mid \langle\rangle \mid \langle \ell = V; W\rangle \mid (\ell\, V) \\
\text{Computations} & M, N & ::= & V\, W \\
 & & \mid & \mathbf{let}\ \langle \ell = x; y\rangle = V\ \mathbf{in}\ N \mid \mathbf{case}\ V\{\ell\, x \mapsto M; y \mapsto N\} \mid \mathbf{absurd}\, V \\
 & & \mid & \mathbf{return}\ V \mid \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N \\
 & & \mid & \mathbf{do}\ \ell\ V \mid \mathbf{handle}\ M\ \mathbf{with}\ H \\
\text{Handlers} & H & ::= & \boxed{\{\mathbf{return}\ x \mapsto M\} \mid \{\ell\ p\ r \mapsto M\} \uplus H\}}
\end{array}
$$

Our target is an untyped λ-calculus

$$
\begin{array}{lrcl}
\text{Values} & V, W & ::= & x \mid \lambda x.M \mid \langle\rangle \mid \langle \ell = V; W\rangle \mid \ell\ V \\
\text{Computations} & M, N & ::= & V \mid M\ W \mid \mathbf{let}\ \langle \ell = x; y\rangle = V\ \mathbf{in}\ N \\
 & & \mid & \mathbf{case}\ V\ \{\ell\ x \mapsto M; y \mapsto N\} \mid \mathbf{absurd}\ V
\end{array}
$$

Curried CPS translation for fine-grain call-by-value

The continuation represents the *execution context*.

Computations

$$
\begin{aligned}
[\![V\ W]\!] &= [\![V]\!]\ [\![W]\!] \\
[\![\mathbf{let}\ \langle \ell = x; y\rangle = V\ \mathbf{in}\ N]\!] &= \mathbf{let}\ \langle \ell = x; y\rangle = [\![V]\!]\ \mathbf{in}\ [\![N]\!] \\
[\![\mathbf{case}\ V\ \{\ell\ x \mapsto M; y \mapsto N\}]\!] &= \mathbf{case}\ [\![V]\!]\ \{\ell\ x \mapsto [\![M]\!]; y \mapsto [\![N]\!]\} \\
[\![\mathbf{absurd}\ V]\!] &= \mathbf{absurd}\ [\![V]\!] \\
[\![\mathbf{return}\ V]\!] &= \lambda k.k\ [\![V]\!] \\
[\![\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N]\!] &= \lambda k.[\![M]\!](\lambda x.[\![N]\!]k)
\end{aligned}
$$

Values

$$
\begin{aligned}
[\![x]\!] &= x \\
[\![\lambda x.M]\!] &= \lambda x.[\![M]\!] \\
[\![\langle\rangle]\!] &= \langle\rangle \\
[\![\langle \ell = V; W\rangle]\!] &= \langle \ell = [\![V]\!]; [\![W]\!]\rangle \\
[\![\ell\ V]\!] &= \ell\ [\![V]\!]
\end{aligned}
$$

Curried CPS for handlers

With handlers there are two contexts: a *pure* and an *effect* context.
Idea: implicit stack of alternating pure and effect continuations.

$$
\begin{aligned}
[\![\mathbf{do}\ \ell\ V]\!] &= \lambda k.\lambda h.h\ (\ell\ \langle [\![V]\!], \lambda x.k\ x\ h\rangle) \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= [\![M]\!]\ [\![H^{\mathrm{ret}}]\!]\ [\![H^{\mathrm{ops}}]\!] \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x.\lambda h.[\![N]\!] \\
[\![\{\ell\ p\ r \mapsto N_\ell\}_{\ell \in \mathcal{L}}]\!] &= \lambda z.\mathbf{case}\ z\ \{(\ell\ \langle p, r\rangle \mapsto [\![N_\ell]\!])_{\ell \in \mathcal{L}}; y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \lambda k'.\lambda h'.\mathbf{vmap}\,(\lambda \langle p, r\rangle\ k.k\ \langle p, \lambda x.r\,x\,k'\ h'\rangle)\ y\ h' \\
\top[\![M]\!] &= [\![M]\!]\ (\lambda x.\lambda h.x)\ (\lambda z.\mathbf{absurd}\ z)
\end{aligned}
$$

Curried CPS for handlers

With handlers there are two contexts: a *pure* and an *effect* context.
Idea: implicit stack of alternating pure and effect continuations.

$$
\begin{aligned}
\boxed{[\![\mathbf{do}\ \ell\ V]\!] = \lambda k.\lambda h.h\ (\ell\ \langle [\![V]\!], \lambda x.k\ x\ h\rangle)} \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= [\![M]\!]\ [\![H^{\mathrm{ret}}]\!]\ [\![H^{\mathrm{ops}}]\!] \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x.\lambda h.[\![N]\!] \\
[\![\{\ell\ p\ r \mapsto N_\ell\}_{\ell\in\mathcal{L}}]\!] &= \lambda z.\mathbf{case}\ z\ \{(\ell\ \langle p, r\rangle \mapsto [\![N_\ell]\!])_{\ell\in\mathcal{L}}; y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \lambda k'.\lambda h'.\mathbf{vmap}\,(\lambda\langle p, r\rangle\ k.k\ \langle p, \lambda x.r\, x\, k'\ h'\rangle)\ y\ h' \\
\top[\![M]\!] &= [\![M]\!]\ (\lambda x.\lambda h.x)\ (\lambda z.\mathbf{absurd}\ z)
\end{aligned}
$$

Curried CPS for handlers

With handlers there are two contexts: a *pure* and an *effect* context.
Idea: implicit stack of alternating pure and effect continuations.

$$
\begin{aligned}
[\![\mathbf{do}\ \ell\ V]\!] &= \lambda k.\lambda h.h\ (\ell\ \langle [\![V]\!], \lambda x.k\ x\ h\rangle) \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= [\![M]\!]\ [\![H^{\mathrm{ret}}]\!]\ [\![H^{\mathrm{ops}}]\!] \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x.\lambda h.[\![N]\!] \\
[\![\{\ell\ p\ r \mapsto N_\ell\}_{\ell\in\mathcal{L}}]\!] &= \lambda z.\mathbf{case}\ z\ \{(\ell\ \langle p, r\rangle \mapsto [\![N_\ell]\!])_{\ell\in\mathcal{L}}; y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \lambda k'.\lambda h'.\mathbf{vmap}\,(\lambda\langle p, r\rangle\ k.k\ \langle p, \lambda x.r\,x\,k'\ h'\rangle)\ y\ h' \\
\top[\![M]\!] &= [\![M]\!]\ (\lambda x.\lambda h.x)\ (\lambda z.\mathbf{absurd}\ z)
\end{aligned}
$$

Curried CPS for handlers

With handlers there are two contexts: a *pure* and an *effect* context.
Idea: implicit stack of alternating pure and effect continuations.

$$
\begin{aligned}
[\![\mathbf{do}\ \ell\ V]\!] &= \lambda k.\lambda h.h\ (\ell\ \langle [\![V]\!], \lambda x.k\ x\ h\rangle) \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= [\![M]\!]\ [\![H^{\mathrm{ret}}]\!]\ [\![H^{\mathrm{ops}}]\!] \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x.\lambda h.[\![N]\!] \\
[\![\{\ell\ p\ r \mapsto N_\ell\}_{\ell \in \mathcal{L}}]\!] &= \lambda z.\mathbf{case}\ z\ \{(\ell\ \langle p, r\rangle \mapsto [\![N_\ell]\!])_{\ell \in \mathcal{L}}; y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \lambda k'.\lambda h'.\mathbf{vmap}\,(\lambda\langle p, r\rangle\ k.k\ \langle p, \lambda x.r\,x\,k'\ h'\rangle)\ y\ h' \\
\boxed{\top[\![M]\!]} &\boxed{= [\![M]\!]\ (\lambda x.\lambda h.x)\ (\lambda z.\mathbf{absurd}\ z)}
\end{aligned}
$$

Curried CPS for handlers

With handlers there are two contexts: a *pure* and an *effect* context.
Idea: implicit stack of alternating pure and effect continuations.

$$
\begin{aligned}
[\![\mathbf{do}\ \ell\ V]\!] &= \lambda k.\lambda h.h\ (\ell\ \langle [\![V]\!], \lambda x.k\ x\ h\rangle) \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= [\![M]\!]\ [\![H^{\mathrm{ret}}]\!]\ [\![H^{\mathrm{ops}}]\!] \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x.\lambda h.[\![N]\!] \\
[\![\{\ell\ p\ r \mapsto N_\ell\}_{\ell\in\mathcal{L}}]\!] &= \lambda z.\mathbf{case}\ z\ \{(\ell\ \langle p, r\rangle \mapsto [\![N_\ell]\!])_{\ell\in\mathcal{L}}; y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \lambda k'.\lambda h'.\mathbf{vmap}\,(\lambda\langle p, r\rangle\ k.k\ \langle p, \lambda x.r\, x\, k'\ h'\rangle)\ y\ h' \\
\top[\![M]\!] &= [\![M]\!]\ (\lambda x.\lambda h.x)\ (\lambda z.\mathbf{absurd}\ z)
\end{aligned}
$$

Problem: $[\![-]\!]$ yields administrative redexes and is not properly tail-recursive!

$$
\top[\![\mathbf{return}\ \langle\rangle]\!] \ = \ (\lambda k.k\ \langle\rangle)\,(\lambda x.\lambda h.x)\,(\lambda z.\mathbf{absurd}\,z)
$$

Curried CPS for handlers

With handlers there are two contexts: a *pure* and an *effect* context.
Idea: implicit stack of alternating pure and effect continuations.

$$
\begin{aligned}
[\![\mathbf{do}\ \ell\ V]\!] &= \lambda k.\lambda h.h\ (\ell\ \langle [\![V]\!], \lambda x.k\ x\ h\rangle) \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= [\![M]\!]\ [\![H^{\mathrm{ret}}]\!]\ [\![H^{\mathrm{ops}}]\!] \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x.\lambda h.[\![N]\!] \\
[\![\{\ell\ p\ r \mapsto N_\ell\}_{\ell\in\mathcal{L}}]\!] &= \lambda z.\mathbf{case}\ z\ \{(\ell\ \langle p, r\rangle \mapsto [\![N_\ell]\!])_{\ell\in\mathcal{L}}; y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \lambda k'.\lambda h'.\mathbf{vmap}\,(\lambda\langle p, r\rangle\ k.k\ \langle p, \lambda x.r\,x\,k'\ h'\rangle)\ y\ h' \\
\top[\![M]\!] &= [\![M]\!]\ (\lambda x.\lambda h.x)\ (\lambda z.\mathbf{absurd}\ z)
\end{aligned}
$$

Problem: $[\![-]\!]$ yields administrative redexes and is not properly tail-recursive!

$$
\begin{aligned}
\top[\![\mathbf{return}\ \langle\rangle]\!] &= (\lambda k.k\ \langle\rangle)\,(\lambda x.\lambda h.x)\,(\lambda z.\mathbf{absurd}\,z) \\
&\rightsquigarrow ((\lambda x.\lambda h.x)\ \langle\rangle)\,(\lambda z.\mathbf{absurd}\,z)
\end{aligned}
$$

Curried CPS for handlers

With handlers there are two contexts: a *pure* and an *effect* context.
Idea: implicit stack of alternating pure and effect continuations.

$$
\begin{aligned}
[\![\mathbf{do}\ \ell\ V]\!] &= \lambda k.\lambda h.h\ (\ell\ \langle [\![V]\!], \lambda x.k\ x\ h\rangle) \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= [\![M]\!]\ [\![H^{\mathrm{ret}}]\!]\ [\![H^{\mathrm{ops}}]\!] \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x.\lambda h.[\![N]\!] \\
[\![\{\ell\ p\ r \mapsto N_\ell\}_{\ell\in\mathcal{L}}]\!] &= \lambda z.\mathbf{case}\ z\ \{(\ell\ \langle p, r\rangle \mapsto [\![N_\ell]\!])_{\ell\in\mathcal{L}}; y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \lambda k'.\lambda h'.\mathbf{vmap}\,(\lambda\langle p, r\rangle\ k.k\ \langle p, \lambda x.r\,x\,k'\ h'\rangle)\ y\ h' \\
\top[\![M]\!] &= [\![M]\!]\ (\lambda x.\lambda h.x)\ (\lambda z.\mathbf{absurd}\ z)
\end{aligned}
$$

Problem: $[\![-]\!]$ yields administrative redexes and is not properly tail-recursive!

$$
\begin{aligned}
\top[\![\mathbf{return}\ \langle\rangle]\!] \ &= \ (\lambda k.k\ \langle\rangle)\,(\lambda x.\lambda h.x)\,(\lambda z.\mathbf{absurd}\,z) \\
&\leadsto \ ((\lambda x.\lambda h.x)\ \langle\rangle)\ (\lambda z.\mathbf{absurd}\,z) \leadsto (\lambda h.\langle\rangle)\ (\lambda z.\mathbf{absurd}\,z)
\end{aligned}
$$

Curried CPS for handlers

With handlers there are two contexts: a *pure* and an *effect* context.
Idea: implicit stack of alternating pure and effect continuations.

$$\begin{aligned}
[\![\textbf{do}\ \ell\ V]\!] &= \lambda k.\lambda h.h\ (\ell\ \langle [\![V]\!], \lambda x.k\ x\ h\rangle) \\
[\![\textbf{handle}\ M\ \textbf{with}\ H]\!] &= [\![M]\!]\ [\![H^{\mathrm{ret}}]\!]\ [\![H^{\mathrm{ops}}]\!] \\
[\![\{\textbf{return}\ x \mapsto N\}]\!] &= \lambda x.\lambda h.[\![N]\!] \\
[\![\{\ell\ p\ r \mapsto N_\ell\}_{\ell\in\mathcal{L}}]\!] &= \lambda z.\textbf{case}\ z\ \{(\ell\ \langle p, r\rangle \mapsto [\![N_\ell]\!])_{\ell\in\mathcal{L}}; y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \lambda k'.\lambda h'.\textbf{vmap}\,(\lambda\langle p, r\rangle\ k.k\ \langle p, \lambda x.r\, x\, k'\ h'\rangle)\ y\ h' \\
\top[\![M]\!] &= [\![M]\!]\ (\lambda x.\lambda h.x)\ (\lambda z.\textbf{absurd}\ z)
\end{aligned}$$

Problem: $[\![-]\!]$ yields administrative redexes and is not properly tail-recursive!

$$\begin{aligned}
\top[\![\textbf{return}\ \langle\rangle]\!] &= (\lambda k.k\ \langle\rangle)\,(\lambda x.\lambda h.x)\,(\lambda z.\textbf{absurd}\, z) \\
&\rightsquigarrow ((\lambda x.\lambda h.x)\ \langle\rangle)\ (\lambda z.\textbf{absurd}\, z) \rightsquigarrow (\lambda h.\langle\rangle)\ (\lambda z.\textbf{absurd}\, z) \rightsquigarrow \langle\rangle
\end{aligned}$$

Uncurried CPS for handlers

Idea: make the continuation stack explicit (Materzok and Biernacki 2012).

$$
\begin{aligned}
[\![\mathbf{return}\ V]\!] &= \lambda(k :: ks).k\ [\![V]\!]\ ks \\
[\![\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N]\!] &= \lambda(k :: ks).[\![M]\!]((\lambda x\, ks.[\![N]\!](k :: ks)) :: ks) \\
[\![\mathbf{do}\ \ell\ V]\!] &= \lambda(k :: h :: ks).h\,(\ell\ \langle [\![V]\!], h :: k :: []\rangle)\ ks \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= \lambda ks.[\![M]\!]([\![H^{\mathrm{ret}}]\!] :: [\![H^{\mathrm{ops}}]\!] :: ks) \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x\, ks.\mathbf{let}\ (h :: ks') = ks\ \mathbf{in}\ [\![N]\!]\ ks \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell\in\mathcal{L}}\}]\!] &= \lambda z\, ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \mathbf{let}\ r = \mathbf{fun}\, s\ \mathbf{in}\ [\![N_\ell]\!]\ ks)_{\ell\in\mathcal{L}}; \\
&\qquad\qquad\qquad\qquad\quad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \mathbf{let}\ (k' :: h' :: ks') = ks\ \mathbf{in} \\
&\quad\ \mathbf{vmap}\,(\lambda\langle p, r\rangle\,(k :: ks).k\,\langle p, h' :: k' :: r\rangle\ ks)\ y\ ks' \\
\top[\![M]\!] &= [\![M]\!]\ ((\lambda x\, ks.x) :: (\lambda z\, ks.\mathbf{absurd}\ z) :: [])
\end{aligned}
$$

Uncurried CPS for handlers

Idea: make the continuation stack explicit (Materzok and Biernacki 2012).

$$\boxed{\begin{aligned}
[\![\mathbf{return}\ V]\!] &= \lambda(k :: ks).k\ [\![V]\!]\ ks \\
[\![\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N]\!] &= \lambda(k :: ks).[\![M]\!]((\lambda x\, ks.[\![N]\!](k :: ks)) :: ks)
\end{aligned}}$$

$$\begin{aligned}
[\![\mathbf{do}\ \ell\ V]\!] &= \lambda(k :: h :: ks).h\,(\ell\ \langle [\![V]\!], h :: k :: []\rangle)\ ks \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= \lambda ks.[\![M]\!]([\![H^{\mathrm{ret}}]\!] :: [\![H^{\mathrm{ops}}]\!] :: ks) \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x\, ks.\mathbf{let}\ (h :: ks') = ks\ \mathbf{in}\ [\![N]\!]\ ks \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell \in \mathcal{L}}\}]\!] &= \lambda z\, ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \mathbf{let}\ r = \mathbf{fun}\, s\ \mathbf{in}\ [\![N_\ell]\!]\ ks)_{\ell \in \mathcal{L}}; \\
&\qquad\qquad\qquad\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \mathbf{let}\ (k' :: h' :: ks') = ks\ \mathbf{in} \\
&\quad\ \mathbf{vmap}\,(\lambda\langle p, r\rangle\ (k :: ks).k\ \langle p, h' :: k' :: r\rangle\ ks)\ y\ ks' \\
\top[\![M]\!] &= [\![M]\!]\ ((\lambda x\, ks.x) :: (\lambda z\, ks.\mathbf{absurd}\ z) :: [])
\end{aligned}$$

Uncurried CPS for handlers

Idea: make the continuation stack explicit (Materzok and Biernacki 2012).

$$
\begin{aligned}
[\![\textbf{return}\ V]\!] &= \lambda(k :: ks).k\ [\![V]\!]\ ks \\
[\![\textbf{let}\ x \leftarrow M\ \textbf{in}\ N]\!] &= \lambda(k :: ks).[\![M]\!]((\lambda x\, ks.[\![N]\!](k :: ks)) :: ks) \\
\boxed{[\![\textbf{do}\ \ell\ V]\!] = \lambda(k :: h :: ks).h\,(\ell\ \langle [\![V]\!], h :: k :: []\rangle)\ ks} \\
[\![\textbf{handle}\ M\ \textbf{with}\ H]\!] &= \lambda ks.[\![M]\!]([\![H^{\mathrm{ret}}]\!] :: [\![H^{\mathrm{ops}}]\!] :: ks) \\
[\![\{\textbf{return}\ x \mapsto N\}]\!] &= \lambda x\, ks.\textbf{let}\ (h :: ks') = ks\ \textbf{in}\ [\![N]\!]\ ks \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell \in \mathcal{L}}\}]\!] &= \lambda z\, ks.\textbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \textbf{let}\ r = \textbf{fun}\ s\ \textbf{in}\ [\![N_\ell]\!]\ ks)_{\ell \in \mathcal{L}}; \\
&\qquad\qquad\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \textbf{let}\ (k' :: h' :: ks') = ks\ \textbf{in} \\
&\quad\ \textbf{vmap}\,(\lambda\langle p, r\rangle\ (k :: ks).k\ \langle p, h' :: k' :: r\rangle\ ks)\ y\ ks' \\
\top[\![M]\!] &= [\![M]\!]\ ((\lambda x\, ks.x) :: (\lambda z\, ks.\textbf{absurd}\ z) :: [])
\end{aligned}
$$

Uncurried CPS for handlers

Idea: make the continuation stack explicit (Materzok and Biernacki 2012).

$$
\begin{aligned}
[\![\textbf{return}\ V]\!] &= \lambda(k :: ks).k\ [\![V]\!]\ ks \\
[\![\textbf{let}\ x \leftarrow M\ \textbf{in}\ N]\!] &= \lambda(k :: ks).[\![M]\!]((\lambda x\, ks.[\![N]\!](k :: ks)) :: ks) \\
[\![\textbf{do}\ \ell\ V]\!] &= \lambda(k :: h :: ks).h\,(\ell\ \langle [\![V]\!], h :: k :: []\rangle)\ ks \\
[\![\textbf{handle}\ M\ \textbf{with}\ H]\!] &= \lambda ks.[\![M]\!]([\![H^{\mathrm{ret}}]\!] :: [\![H^{\mathrm{ops}}]\!] :: ks) \\
[\![\{\textbf{return}\ x \mapsto N\}]\!] &= \lambda x\, ks.\textbf{let}\ (h :: ks') = ks\ \textbf{in}\ [\![N]\!]\ ks \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell\in\mathcal{L}}\}]\!] &= \lambda z\, ks.\textbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \textbf{let}\ r = \textbf{fun}\, s\ \textbf{in}\ [\![N_\ell]\!]\ ks)_{\ell\in\mathcal{L}}; \\
&\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \textbf{let}\ (k' :: h' :: ks') = ks\ \textbf{in} \\
&\quad\ \textbf{vmap}\,(\lambda\langle p, r\rangle\ (k :: ks).k\ \langle p, h' :: k' :: r\rangle\ ks)\ y\ ks' \\
\top[\![M]\!] &= [\![M]\!]\ ((\lambda x\, ks.x) :: (\lambda z\, ks.\textbf{absurd}\ z) :: [])
\end{aligned}
$$

Uncurried CPS for handlers

Idea: make the continuation stack explicit (Materzok and Biernacki 2012).

$$
\begin{aligned}
[\![\mathbf{return}\ V]\!] &= \lambda(k :: ks).k\ [\![V]\!]\ ks \\
[\![\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N]\!] &= \lambda(k :: ks).[\![M]\!]((\lambda x\, ks.[\![N]\!](k :: ks)) :: ks) \\
[\![\mathbf{do}\ \ell\ V]\!] &= \lambda(k :: h :: ks).h\,(\ell\ \langle [\![V]\!], h :: k :: [] \rangle)\ ks \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= \lambda ks.[\![M]\!]([\![H^{\mathrm{ret}}]\!] :: [\![H^{\mathrm{ops}}]\!] :: ks) \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x\, ks.\mathbf{let}\ (h :: ks') = ks\ \mathbf{in}\ [\![N]\!]\ ks \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell\in\mathcal{L}}\}]\!] &= \lambda z\, ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \mathbf{let}\ r = \mathbf{fun}\, s\ \mathbf{in}\ [\![N_\ell]\!]\ ks)_{\ell\in\mathcal{L}}; \\
&\qquad\qquad\qquad\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \mathbf{let}\ (k' :: h' :: ks') = ks\ \mathbf{in} \\
&\quad\ \mathbf{vmap}\,(\lambda\langle p, r\rangle\ (k :: ks).k\ \langle p, h' :: k' :: r\rangle\ ks)\ y\ ks' \\
\top[\![M]\!] &= [\![M]\!]\ ((\lambda x\, ks.x) :: (\lambda z\, ks.\mathbf{absurd}\ z) :: [])
\end{aligned}
$$

But $[\![-]\!]$ still yields the administrative redexes

$$
\top[\![\mathbf{return}\ \langle\rangle]\!] \;=\; (\lambda(k :: ks).k\ \langle\rangle\ ks)\,((\lambda x\ \ ks.x) :: (\lambda z.\mathbf{absurd}\, z) :: [])
$$

Uncurried CPS for handlers

Idea: make the continuation stack explicit (Materzok and Biernacki 2012).

$$
\begin{aligned}
[\![\mathbf{return}\ V]\!] &= \lambda(k :: ks).k\ [\![V]\!]\ ks \\
[\![\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N]\!] &= \lambda(k :: ks).[\![M]\!]((\lambda x\, ks.[\![N]\!](k :: ks)) :: ks) \\
[\![\mathbf{do}\ \ell\ V]\!] &= \lambda(k :: h :: ks).h\,(\ell\ \langle [\![V]\!], h :: k :: []\rangle)\ ks \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= \lambda ks.[\![M]\!]([\![H^{\mathrm{ret}}]\!] :: [\![H^{\mathrm{ops}}]\!] :: ks) \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x\, ks.\mathbf{let}\ (h :: ks') = ks\ \mathbf{in}\ [\![N]\!]\ ks \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell\in\mathcal{L}}\}]\!] &= \lambda z\, ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \mathbf{let}\ r = \mathbf{fun}\,s\ \mathbf{in}\ [\![N_\ell]\!]\ ks)_{\ell\in\mathcal{L}}; \\
&\qquad\qquad\qquad\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \mathbf{let}\ (k' :: h' :: ks') = ks\ \mathbf{in} \\
&\quad\ \mathbf{vmap}\,(\lambda\langle p, r\rangle\ (k :: ks).k\ \langle p, h' :: k' :: r\rangle\ ks)\ y\ ks' \\
\top[\![M]\!] &= [\![M]\!]\ ((\lambda x\, ks.x) :: (\lambda z\, ks.\mathbf{absurd}\ z) :: [])
\end{aligned}
$$

But $[\![-]\!]$ still yields the administrative redexes

$$
\begin{aligned}
\top[\![\mathbf{return}\ \langle\rangle]\!] &= (\lambda(k :: ks).k\ \langle\rangle\ ks)\ ((\lambda x\ ks.x) :: (\lambda z.\mathbf{absurd}\, z) :: []) \\
&\rightsquigarrow (\lambda x\ ks.x)\ \langle\rangle\ [(\lambda z.\mathbf{absurd}\, z)]
\end{aligned}
$$

Uncurried CPS for handlers

Idea: make the continuation stack explicit (Materzok and Biernacki 2012).

$$
\begin{aligned}
[\![\mathbf{return}\ V]\!] &= \lambda(k :: ks).k\ [\![V]\!]\ ks \\
[\![\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N]\!] &= \lambda(k :: ks).[\![M]\!]((\lambda x\, ks.[\![N]\!](k :: ks)) :: ks) \\
[\![\mathbf{do}\ \ell\ V]\!] &= \lambda(k :: h :: ks).h\,(\ell\ \langle [\![V]\!], h :: k :: [] \rangle)\ ks \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= \lambda ks.[\![M]\!]([\![H^{\mathrm{ret}}]\!] :: [\![H^{\mathrm{ops}}]\!] :: ks) \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x\, ks.\mathbf{let}\ (h :: ks') = ks\ \mathbf{in}\ [\![N]\!]\ ks \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell \in \mathcal{L}}\}]\!] &= \lambda z\, ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \mathbf{let}\ r = \mathbf{fun}\, s\ \mathbf{in}\ [\![N_\ell]\!]\ ks)_{\ell \in \mathcal{L}}; \\
&\qquad\qquad\qquad\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \mathbf{let}\ (k' :: h' :: ks') = ks\ \mathbf{in} \\
&\quad\ \mathbf{vmap}\,(\lambda\langle p, r\rangle\ (k :: ks).k\ \langle p, h' :: k' :: r\rangle\ ks)\ y\ ks' \\
\top[\![M]\!] &= [\![M]\!]\ ((\lambda x\, ks.x) :: (\lambda z\, ks.\mathbf{absurd}\ z) :: [])
\end{aligned}
$$

But $[\![-]\!]$ still yields the administrative redexes

$$
\begin{aligned}
\top[\![\mathbf{return}\ \langle\rangle]\!] &= (\lambda(k :: ks).k\ \langle\rangle\ ks)\,((\lambda x\ \ ks.x) :: (\lambda z.\mathbf{absurd}\, z) :: []) \\
&\rightsquigarrow (\lambda x\ \ ks.x)\ \langle\rangle\ [(\lambda z.\mathbf{absurd}\, z)] \rightsquigarrow^{+} \langle\rangle
\end{aligned}
$$

Higher-order CPS for handlers (I)

Idea: adopt a two level λ-calculus (Danvy and Filinski 1990; Danvy and Nielsen 2003) with two kinds of continuations

- Static continuations κ
- Dynamic continuations k

Correspondingly, two kinds of reductions

- $\overline{\textit{Static}}$ reductions at translation time
- $\underline{\textit{Dynamic}}$ reductions at run-time

Move between levels using reflection ($\uparrow$) and reification ($\downarrow$)

$$\begin{aligned} \downarrow\uparrow V &= V \\ \downarrow(V \mathbin{\overline{::}} VS) &= V \mathbin{\overline{::}} \downarrow VS \end{aligned}$$

Higher-order CPS for handlers (II)

Two kinds of reductions: $\overline{\textit{static}}$ and $\underline{\textit{dynamic}}$.

$$
\begin{aligned}
[\![\textbf{return } V]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\kappa \mathbin{\underline{@}} [\![V]\!] \mathbin{\underline{@}} {\downarrow}\kappa s \\
[\![\textbf{let } x \leftarrow M \textbf{ in } N]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.[\![M]\!] \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.[\![N]\!] \mathbin{\overline{@}} (\kappa \mathbin{\overline{::}} {\uparrow}ks)) \mathbin{\overline{::}} \kappa s) \\
[\![\textbf{do } \ell\ V]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \eta \mathbin{\overline{::}} \kappa s.\eta \mathbin{\underline{@}} (\ell\ \langle [\![V]\!], \eta \mathbin{\underline{::}} \kappa \mathbin{\underline{::}} [\,]\rangle) \mathbin{\underline{@}} {\downarrow}\kappa s \\
[\![\textbf{handle } M \textbf{ with } H]\!] &= \overline{\lambda}\kappa s.[\![M]\!] \mathbin{\overline{@}} ([\![H^{\mathrm{ret}}]\!] \mathbin{\overline{::}} [\![H^{\mathrm{ops}}]\!] \mathbin{\overline{::}} \kappa s) \\
[\![\{\textbf{return } x \mapsto N\}]\!] &= \underline{\lambda} x\, ks.\underline{\textbf{let}}\ (h \mathbin{\underline{::}} ks') = ks \mathbin{\underline{\textbf{in}}} [\![N]\!] \mathbin{\overline{@}} {\uparrow}ks' \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell \in \mathcal{L}}\}]\!] &= \underline{\lambda} z\, ks.\textbf{case } z\ \{(\ell\ \langle p, s\rangle \mapsto \underline{\textbf{let}}\ r = \textbf{fun}\, s \mathbin{\underline{\textbf{in}}} [\![N_\ell]\!] \mathbin{\overline{@}} {\uparrow}ks)_{\ell \in \mathcal{L}}; \\
&\qquad\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \underline{\textbf{let}}\ (k' \mathbin{\underline{::}} h' \mathbin{\underline{::}} ks') = ks \mathbin{\underline{\textbf{in}}} \\
&\ \textbf{vmap}\, (\underline{\lambda}\langle p, s\rangle\ (k \mathbin{\underline{::}} ks).k\ \langle p, h' \mathbin{\underline{::}} k' \mathbin{\underline{::}} s\rangle\ ks)\ y\ ks' \\
\top[\![M]\!] &= [\![M]\!] \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.x) \mathbin{\overline{::}} (\underline{\lambda} z\, ks.\textbf{absurd } z) \mathbin{\overline{::}} {\uparrow}[\,])
\end{aligned}
$$

Higher-order CPS for handlers (II)

Two kinds of reductions: $\overline{\textit{static}}$ and $\underline{\textit{dynamic}}$.

$$
\begin{aligned}
[\![\mathbf{return}\ V]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\kappa \mathbin{\underline{@}} [\![V]\!] \mathbin{\underline{@}} {\downarrow}\kappa s \\
[\![\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.[\![M]\!] \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.[\![N]\!] \mathbin{\overline{@}} (\kappa \mathbin{\overline{::}} {\uparrow} ks)) \mathbin{\overline{::}} \kappa s) \\
[\![\mathbf{do}\ \ell\ V]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \eta \mathbin{\overline{::}} \kappa s.\eta \mathbin{\underline{@}} (\ell\ \langle [\![V]\!], \eta \mathbin{\underline{::}} \kappa \mathbin{\underline{::}} []\rangle) \mathbin{\underline{@}} {\downarrow}\kappa s \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= \overline{\lambda}\kappa s.[\![M]\!] \mathbin{\overline{@}} ([\![H^{\mathrm{ret}}]\!] \mathbin{\overline{::}} [\![H^{\mathrm{ops}}]\!] \mathbin{\overline{::}} \kappa s) \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \underline{\lambda} x\, ks.\underline{\mathbf{let}}\ (h \mathbin{\underline{::}} ks') = ks\ \underline{\mathbf{in}}\ [\![N]\!] \mathbin{\overline{@}} {\uparrow} ks' \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell \in \mathcal{L}}\}]\!] &= \underline{\lambda} z\, ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \underline{\mathbf{let}}\ r = \mathbf{fun}\, s\ \underline{\mathbf{in}}\ [\![N_\ell]\!] \mathbin{\overline{@}} {\uparrow} ks)_{\ell \in \mathcal{L}}; \\
&\qquad\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \underline{\mathbf{let}}\ (k' \mathbin{\underline{::}} h' \mathbin{\underline{::}} ks') = ks\ \underline{\mathbf{in}} \\
&\quad\ \mathbf{vmap}\, (\underline{\lambda}\langle p, s\rangle\ (k \mathbin{\underline{::}} ks).k\ \langle p, h' \mathbin{\underline{::}} k' \mathbin{\underline{::}} s\rangle\ ks)\ y\ ks' \\
\top[\![M]\!] &= [\![M]\!] \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.x) \mathbin{\overline{::}} (\underline{\lambda} z\, ks.\mathbf{absurd}\ z) \mathbin{\overline{::}} {\uparrow}[])
\end{aligned}
$$

Higher-order CPS for handlers (II)

Two kinds of reductions: $\overline{\textit{static}}$ and $\underline{\textit{dynamic}}$.

$$
\begin{aligned}
[\![\textbf{return}\ V]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\kappa \mathbin{\underline{@}} [\![V]\!] \mathbin{\underline{@}} \downarrow\kappa s \\
[\![\textbf{let}\ x \leftarrow M\ \textbf{in}\ N]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.[\![M]\!] \mathbin{\overline{@}} ((\underline{\lambda}x\ ks.[\![N]\!] \mathbin{\overline{@}} (\kappa \mathbin{\overline{::}} \uparrow ks)) \mathbin{\overline{::}} \kappa s) \\
\boxed{[\![\textbf{do}\ \ell\ V]\!] = \overline{\lambda}\kappa \mathbin{\overline{::}} \eta \mathbin{\overline{::}} \kappa s.\eta \mathbin{\underline{@}} (\ell\ \langle [\![V]\!], \eta \mathbin{\underline{::}} \kappa \mathbin{\underline{::}} [\,]\rangle) \mathbin{\underline{@}} \downarrow\kappa s} \\
[\![\textbf{handle}\ M\ \textbf{with}\ H]\!] &= \overline{\lambda}\kappa s.[\![M]\!] \mathbin{\overline{@}} ([\![H^{\mathrm{ret}}]\!] \mathbin{\overline{::}} [\![H^{\mathrm{ops}}]\!] \mathbin{\overline{::}} \kappa s) \\
[\![\{\textbf{return}\ x \mapsto N\}]\!] &= \underline{\lambda}x\ ks.\underline{\textbf{let}}\ (h \mathbin{\underline{::}} ks') = ks\ \underline{\textbf{in}}\ [\![N]\!] \mathbin{\overline{@}} \uparrow ks' \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell\in\mathcal{L}}\}]\!] &= \underline{\lambda}z\ ks.\textbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \underline{\textbf{let}}\ r = \textbf{fun}\ s\ \underline{\textbf{in}}\ [\![N_\ell]\!] \mathbin{\overline{@}} \uparrow ks)_{\ell\in\mathcal{L}}; \\
&\qquad\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \underline{\textbf{let}}\ (k' \mathbin{\underline{::}} h' \mathbin{\underline{::}} ks') = ks\ \underline{\textbf{in}} \\
&\quad\ \textbf{vmap}\ (\underline{\lambda}\langle p, s\rangle\ (k \mathbin{\underline{::}} ks).k\ \langle p, h' \mathbin{\underline{::}} k' \mathbin{\underline{::}} s\rangle\ ks)\ y\ ks' \\
\top[\![M]\!] &= [\![M]\!] \mathbin{\overline{@}} ((\underline{\lambda}x\ ks.x) \mathbin{\overline{::}} (\underline{\lambda}z\ ks.\textbf{absurd}\ z) \mathbin{\overline{::}} \uparrow[\,])
\end{aligned}
$$

Higher-order CPS for handlers (II)

Two kinds of reductions: $\overline{\textit{static}}$ and $\underline{\textit{dynamic}}$.

$$
\begin{aligned}
\llbracket \mathbf{return}\ V \rrbracket &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\kappa \mathbin{\underline{@}} \llbracket V \rrbracket \mathbin{\underline{@}} \downarrow\kappa s \\
\llbracket \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N \rrbracket &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\llbracket M \rrbracket \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.\llbracket N \rrbracket \mathbin{\overline{@}} (\kappa \mathbin{\overline{::}} \uparrow ks)) \mathbin{\overline{::}} \kappa s) \\
\llbracket \mathbf{do}\ \ell\ V \rrbracket &= \overline{\lambda}\kappa \mathbin{\overline{::}} \eta \mathbin{\overline{::}} \kappa s.\eta \mathbin{\underline{@}} (\ell\ \langle \llbracket V \rrbracket, \eta \mathbin{\underline{::}} \kappa \mathbin{\underline{::}} [] \rangle) \mathbin{\underline{@}} \downarrow\kappa s \\
\llbracket \mathbf{handle}\ M\ \mathbf{with}\ H \rrbracket &= \overline{\lambda}\kappa s.\llbracket M \rrbracket \mathbin{\overline{@}} (\llbracket H^{\mathrm{ret}} \rrbracket \mathbin{\overline{::}} \llbracket H^{\mathrm{ops}} \rrbracket \mathbin{\overline{::}} \kappa s) \\
\llbracket \{\mathbf{return}\ x \mapsto N\} \rrbracket &= \underline{\lambda} x\, ks.\underline{\mathbf{let}}\ (h \mathbin{\underline{::}} ks') = ks\ \underline{\mathbf{in}}\ \llbracket N \rrbracket \mathbin{\overline{@}} \uparrow ks' \\
\llbracket \{(\ell\ p\ r \mapsto N_\ell)_{\ell \in \mathcal{L}}\} \rrbracket &= \underline{\lambda} z\, ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s \rangle \mapsto \underline{\mathbf{let}}\ r = \mathbf{fun}\, s\ \underline{\mathbf{in}}\ \llbracket N_\ell \rrbracket \mathbin{\overline{@}} \uparrow ks)_{\ell \in \mathcal{L}}; \\
&\qquad\qquad\qquad\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \underline{\mathbf{let}}\ (k' \mathbin{\underline{::}} h' \mathbin{\underline{::}} ks') = ks\ \underline{\mathbf{in}} \\
&\ \mathbf{vmap}\, (\underline{\lambda} \langle p, s \rangle\, (k \mathbin{\underline{::}} ks).k\ \langle p, h' \mathbin{\underline{::}} k' \mathbin{\underline{::}} s \rangle\ ks)\ y\ ks' \\
\top \llbracket M \rrbracket &= \llbracket M \rrbracket \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.x) \mathbin{\overline{::}} (\underline{\lambda} z\, ks.\mathbf{absurd}\ z) \mathbin{\overline{::}} \uparrow [])
\end{aligned}
$$

Higher-order CPS for handlers (II)

Two kinds of reductions: $\overline{\textit{static}}$ and $\underline{\textit{dynamic}}$.

$$
\begin{aligned}
[\![\mathbf{return}\ V]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\kappa \mathbin{\underline{@}} [\![V]\!] \mathbin{\underline{@}} {\downarrow}\kappa s \\
[\![\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.[\![M]\!] \mathbin{\overline{@}} ((\underline{\lambda} x\ ks.[\![N]\!] \mathbin{\overline{@}} (\kappa \mathbin{\overline{::}} {\uparrow}ks)) \mathbin{\overline{::}} \kappa s) \\
[\![\mathbf{do}\ \ell\ V]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \eta \mathbin{\overline{::}} \kappa s.\eta \mathbin{\underline{@}} (\ell\ \langle [\![V]\!], \eta \mathbin{\underline{::}} \kappa \mathbin{\underline{::}} [\,]\rangle) \mathbin{\underline{@}} {\downarrow}\kappa s \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= \overline{\lambda}\kappa s.[\![M]\!] \mathbin{\overline{@}} ([\![H^{\mathrm{ret}}]\!] \mathbin{\overline{::}} [\![H^{\mathrm{ops}}]\!] \mathbin{\overline{::}} \kappa s) \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \underline{\lambda} x\ ks.\underline{\mathbf{let}}\ (h \mathbin{\underline{::}} ks') = ks\ \underline{\mathbf{in}}\ [\![N]\!] \mathbin{\overline{@}} {\uparrow}ks' \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell\in\mathcal{L}}\}]\!] &= \underline{\lambda} z\ ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \underline{\mathbf{let}}\ r = \mathbf{fun}\ s\ \underline{\mathbf{in}}\ [\![N_\ell]\!] \mathbin{\overline{@}} {\uparrow}ks)_{\ell\in\mathcal{L}}; \\
&\qquad\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \underline{\mathbf{let}}\ (k' \mathbin{\underline{::}} h' \mathbin{\underline{::}} ks') = ks\ \underline{\mathbf{in}} \\
&\ \mathbf{vmap}\ (\underline{\lambda}\langle p, s\rangle\ (k \mathbin{\underline{::}} ks).k\ \langle p, h' \mathbin{\underline{::}} k' \mathbin{\underline{::}} s\rangle\ ks)\ y\ ks' \\
\top[\![M]\!] &= [\![M]\!] \mathbin{\overline{@}} ((\underline{\lambda} x\ ks.x) \mathbin{\overline{::}} (\underline{\lambda} z\ ks.\mathbf{absurd}\ z) \mathbin{\overline{::}} {\uparrow}[\,])
\end{aligned}
$$

$[\![-]\!]$ produces no statically eliminatable administrative redexes

$$
\top[\![\mathbf{return}\ \langle\rangle]\!] \;=\; (\overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\kappa \mathbin{\underline{@}} \langle\rangle \mathbin{\underline{@}} {\downarrow}\kappa) \mathbin{\overline{@}} ((\underline{\lambda} x\ ks.x) \mathbin{\overline{::}} (\underline{\lambda} z\ ks.\mathbf{absurd}\ z) \mathbin{\overline{::}} {\uparrow}[\,])
$$

Higher-order CPS for handlers (II)

Two kinds of reductions: $\overline{\textit{static}}$ and $\underline{\textit{dynamic}}$.

$$
\begin{aligned}
\llbracket \mathbf{return}\ V \rrbracket &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\kappa \mathbin{\underline{@}} \llbracket V \rrbracket \mathbin{\underline{@}} {\downarrow}\kappa s \\
\llbracket \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N \rrbracket &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\llbracket M \rrbracket \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.\llbracket N \rrbracket \mathbin{\overline{@}} (\kappa \mathbin{\overline{::}} {\uparrow}ks)) \mathbin{\overline{::}} \kappa s) \\
\llbracket \mathbf{do}\ \ell\ V \rrbracket &= \overline{\lambda}\kappa \mathbin{\overline{::}} \eta \mathbin{\overline{::}} \kappa s.\eta \mathbin{\underline{@}} (\ell\ \langle \llbracket V \rrbracket, \eta \mathbin{\underline{::}} \kappa \mathbin{\underline{::}} [\,]\rangle) \mathbin{\underline{@}} {\downarrow}\kappa s \\
\llbracket \mathbf{handle}\ M\ \mathbf{with}\ H \rrbracket &= \overline{\lambda}\kappa s.\llbracket M \rrbracket \mathbin{\overline{@}} (\llbracket H^{\mathrm{ret}} \rrbracket \mathbin{\overline{::}} \llbracket H^{\mathrm{ops}} \rrbracket \mathbin{\overline{::}} \kappa s) \\
\llbracket \{\mathbf{return}\ x \mapsto N\} \rrbracket &= \underline{\lambda} x\, ks.\underline{\mathbf{let}}\ (h \mathbin{\underline{::}} ks') = ks\ \underline{\mathbf{in}}\ \llbracket N \rrbracket \mathbin{\overline{@}} {\uparrow}ks' \\
\llbracket \{(\ell\ p\ r \mapsto N_\ell)_{\ell \in \mathcal{L}}\} \rrbracket &= \underline{\lambda} z\, ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \underline{\mathbf{let}}\ r = \mathbf{fun}\ s\ \underline{\mathbf{in}}\ \llbracket N_\ell \rrbracket \mathbin{\overline{@}} {\uparrow}ks)_{\ell \in \mathcal{L}}; \\
&\qquad\qquad\qquad\quad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \underline{\mathbf{let}}\ (k' \mathbin{\underline{::}} h' \mathbin{\underline{::}} ks') = ks\ \underline{\mathbf{in}} \\
&\ \mathbf{vmap}\ (\underline{\lambda}\langle p, s\rangle\ (k \mathbin{\underline{::}} ks).k\ \langle p, h' \mathbin{\underline{::}} k' \mathbin{\underline{::}} s\rangle\ ks)\ y\ ks' \\
\top\llbracket M \rrbracket &= \llbracket M \rrbracket \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.x) \mathbin{\overline{::}} (\underline{\lambda} z\, ks.\mathbf{absurd}\ z) \mathbin{\overline{::}} {\uparrow}[\,])
\end{aligned}
$$

$\llbracket - \rrbracket$ produces no statically eliminatable administrative redexes

$$
\begin{aligned}
\top\llbracket \mathbf{return}\ \langle\rangle \rrbracket &= (\overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\kappa \mathbin{\underline{@}} \langle\rangle \mathbin{\underline{@}} {\downarrow}\kappa) \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.x) \mathbin{\overline{::}} (\underline{\lambda} z\, ks.\mathbf{absurd}\ z) \mathbin{\overline{::}} {\uparrow}[\,]) \\
&= (\underline{\lambda} x\, ks.x) \mathbin{\underline{@}} \langle\rangle \mathbin{\underline{@}} {\downarrow}((\underline{\lambda} z\, ks.\mathbf{absurd}\ z) \mathbin{\overline{::}} {\uparrow}[\,])
\end{aligned}
$$

Higher-order CPS for handlers (II)

Two kinds of reductions: $\overline{\textit{static}}$ and $\underline{\textit{dynamic}}$.

$$
\begin{aligned}
[\![\mathbf{return}\ V]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\kappa \mathbin{\underline{@}} [\![V]\!] \mathbin{\underline{@}} {\downarrow}\kappa s \\
[\![\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.[\![M]\!] \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.[\![N]\!] \mathbin{\overline{@}} (\kappa \mathbin{\overline{::}} {\uparrow}ks)) \mathbin{\overline{::}} \kappa s) \\
[\![\mathbf{do}\ \ell\ V]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \eta \mathbin{\overline{::}} \kappa s.\eta \mathbin{\underline{@}} (\ell\ \langle [\![V]\!], \eta \mathbin{\underline{::}} \kappa \mathbin{\underline{::}} [\,]\rangle) \mathbin{\underline{@}} {\downarrow}\kappa s \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= \overline{\lambda}\kappa s.[\![M]\!] \mathbin{\overline{@}} ([\![H^{\mathrm{ret}}]\!] \mathbin{\overline{::}} [\![H^{\mathrm{ops}}]\!] \mathbin{\overline{::}} \kappa s) \\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \underline{\lambda} x\, ks.\underline{\mathbf{let}}\ (h \mathbin{\underline{::}} ks') = ks\ \underline{\mathbf{in}}\ [\![N]\!] \mathbin{\overline{@}} {\uparrow}ks' \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell \in \mathcal{L}}\}]\!] &= \underline{\lambda} z\, ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \underline{\mathbf{let}}\ r = \mathbf{fun}\, s\ \underline{\mathbf{in}}\ [\![N_\ell]\!] \mathbin{\overline{@}} {\uparrow}ks)_{\ell \in \mathcal{L}}; \\
&\qquad\qquad\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \underline{\mathbf{let}}\ (k' \mathbin{\underline{::}} h' \mathbin{\underline{::}} ks') = ks\ \underline{\mathbf{in}} \\
&\quad\ \mathbf{vmap}\ (\underline{\lambda}\langle p, s\rangle\ (k \mathbin{\underline{::}} ks).k\ \langle p, h' \mathbin{\underline{::}} k' \mathbin{\underline{::}} s\rangle\ ks)\ y\ ks' \\
\top[\![M]\!] &= [\![M]\!] \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.x) \mathbin{\overline{::}} (\underline{\lambda} z\, ks.\mathbf{absurd}\ z) \mathbin{\overline{::}} {\uparrow}[\,])
\end{aligned}
$$

$[\![-]\!]$ produces no statically eliminatable administrative redexes

$$
\begin{aligned}
\top[\![\mathbf{return}\ \langle\rangle]\!] &= (\overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\kappa \mathbin{\underline{@}} \langle\rangle \mathbin{\underline{@}} {\downarrow}\kappa) \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.x) \mathbin{\overline{::}} (\underline{\lambda} z\, ks.\mathbf{absurd}\, z) \mathbin{\overline{::}} {\uparrow}[\,]) \\
&= (\underline{\lambda} x\, ks.x) \mathbin{\underline{@}} \langle\rangle \mathbin{\underline{@}} {\downarrow}((\underline{\lambda} z\, ks.\mathbf{absurd}\, z) \mathbin{\overline{::}} {\uparrow}[\,]) \\
&= (\underline{\lambda} x\, ks.x) \mathbin{\underline{@}} \langle\rangle \mathbin{\underline{@}} ((\underline{\lambda} z\, ks.\mathbf{absurd}\, z) \mathbin{\overline{::}} [\,])
\end{aligned}
$$

Higher-order CPS for handlers (II)

Two kinds of reductions: $\overline{\textit{static}}$ and $\underline{\textit{dynamic}}$.

$$
\begin{aligned}
[\![\textbf{return } V]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\kappa \mathbin{\underline{@}} [\![V]\!] \mathbin{\underline{@}} \downarrow\kappa s \\
[\![\textbf{let } x \leftarrow M \textbf{ in } N]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.[\![M]\!] \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.[\![N]\!] \mathbin{\overline{@}} (\kappa \mathbin{\overline{::}} \uparrow ks)) \mathbin{\overline{::}} \kappa s) \\
[\![\textbf{do } \ell\ V]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \eta \mathbin{\overline{::}} \kappa s.\eta \mathbin{\underline{@}} (\ell\ \langle [\![V]\!], \eta \mathbin{\underline{::}} \kappa \mathbin{\underline{::}} []\rangle) \mathbin{\underline{@}} \downarrow\kappa s \\
[\![\textbf{handle } M \textbf{ with } H]\!] &= \overline{\lambda}\kappa s.[\![M]\!] \mathbin{\overline{@}} ([\![H^{\mathrm{ret}}]\!] \mathbin{\overline{::}} [\![H^{\mathrm{ops}}]\!] \mathbin{\overline{::}} \kappa s) \\
[\![\{\textbf{return } x \mapsto N\}]\!] &= \underline{\lambda} x\, ks.\underline{\textbf{let}}\ (h \mathbin{\underline{::}} ks') = ks \mathbin{\underline{\textbf{in}}} [\![N]\!] \mathbin{\overline{@}} \uparrow ks' \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell \in \mathcal{L}}\}]\!] &= \underline{\lambda} z\, ks.\textbf{case } z\ \{(\ell\ \langle p, s\rangle \mapsto \underline{\textbf{let}}\ r = \textbf{fun}\, s \mathbin{\underline{\textbf{in}}} [\![N_\ell]\!] \mathbin{\overline{@}} \uparrow ks)_{\ell \in \mathcal{L}}; \\
&\qquad\qquad\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \underline{\textbf{let}}\ (k' \mathbin{\underline{::}} h' \mathbin{\underline{::}} ks') = ks \mathbin{\underline{\textbf{in}}} \\
&\quad\ \textbf{vmap}\ (\underline{\lambda}\langle p, s\rangle\ (k \mathbin{\underline{::}} ks).k\ \langle p, h' \mathbin{\underline{::}} k' \mathbin{\underline{::}} s\rangle\ ks)\ y\ ks' \\
\top[\![M]\!] &= [\![M]\!] \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.x) \mathbin{\overline{::}} (\underline{\lambda} z\, ks.\textbf{absurd } z) \mathbin{\overline{::}} \uparrow[])
\end{aligned}
$$

$[\![-]\!]$ produces no statically eliminatable administrative redexes

$$
\begin{aligned}
\top[\![\textbf{return } \langle\rangle]\!] &= (\overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\kappa \mathbin{\underline{@}} \langle\rangle \mathbin{\underline{@}} \downarrow\kappa) \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.x) \mathbin{\overline{::}} (\underline{\lambda} z\, ks.\textbf{absurd } z) \mathbin{\overline{::}} \uparrow[]) \\
&= (\underline{\lambda} x\, ks.x) \mathbin{\underline{@}} \langle\rangle \mathbin{\underline{@}} \downarrow((\underline{\lambda} z\, ks.\textbf{absurd } z) \mathbin{\overline{::}} \uparrow[]) \\
&= (\underline{\lambda} x\, ks.x) \mathbin{\underline{@}} \langle\rangle \mathbin{\underline{@}} ((\underline{\lambda} z\, ks.\textbf{absurd } z) \mathbin{\overline{::}} []) \rightsquigarrow^{+} \langle\rangle
\end{aligned}
$$

Correctness

Definition (Translation of evaluation contexts)

$$
\begin{aligned}
[\![[\,]]\!] &= \overline{\lambda}\kappa s.\ \kappa s \\
[\![\mathbf{let}\ x \leftarrow \mathcal{E}\ \mathbf{in}\ N]\!] &= \overline{\lambda}\kappa \mathbin{\overline{::}} \kappa s.\ [\![\mathcal{E}]\!] \mathbin{\overline{@}} ((\underline{\lambda} x\, ks.[\![N]\!] \mathbin{\overline{@}} (k \mathbin{\overline{::}} \uparrow ks)) \mathbin{\overline{::}} \kappa s) \\
[\![\mathbf{handle}\ \mathcal{E}\ \mathbf{with}\ H]\!] &= \overline{\lambda}\kappa s.\ [\![\mathcal{E}]\!] \mathbin{\overline{@}} ([\![H^{\mathrm{ret}}]\!] \mathbin{\overline{::}} [\![H^{\mathrm{ops}}]\!] \mathbin{\overline{::}} \kappa s)
\end{aligned}
$$

Lemma (Decomposition)

The characteristic property of the CPS translation on evaluation contexts

$$[\![\mathcal{E}[M]]\!] \mathbin{\overline{@}} (V \mathbin{\overline{::}} VS) = [\![M]\!] \mathbin{\overline{@}} ([\![\mathcal{E}]\!] \mathbin{\overline{@}} (V \mathbin{\overline{::}} VS))$$

Theorem (Simulation)

If $M \rightsquigarrow N$ *then* $\top[\![M]\!] \rightsquigarrow^{+} \rightsquigarrow_a^{*} \top[\![N]\!]$.

Conclusions

In summary

- handlers provide a modular abstraction for user-defined computational effects,
- first full CPS translations for handlers,
- the first-order curried CPS is neat, but inpractical,
- and the first-order uncurried CPS is naïve,
- refectified by a higher-order uncurried CPS.

Full details in the paper along with

- a discussion on the implementation of the higher-order CPS,
- a sketch of a typed higher-order CPS translation,
- an adaptation to accommodate *shallow* handlers,
- and a discussion on adapting the translation to a language like OCaml.

An implementation is available in Links

`https://github.com/links-lang/links`

References

Danvy, Olivier and Andrzej Filinski (1990). "Abstracting Control". In: *LISP and Functional Programming*, pp. 151–160.

Danvy, Olivier and Lasse R. Nielsen (2003). "A first-order one-pass CPS transformation". In: *Theor. Comput. Sci.* 308.1-3, pp. 239–257.

Cooper, Ezra et al. (2006). "Links: Web Programming Without Tiers". In: *FMCO*. Vol. 4709. LNCS. Springer, pp. 266–296.

Materzok, Marek and Dariusz Biernacki (2012). "A Dynamic Interpretation of the CPS Hierarchy". In: *APLAS*. Vol. 7705. LNCS. Springer, pp. 296–311.

Hillerström, Daniel and Sam Lindley (2016). "Liberating effects with rows and handlers". In: *TyDe@ICFP*. ACM, pp. 15–27.