Algebraic Effects and Handlers

Ningning Xie

University of Toronto
OPLSS 2025

1. Algebraic effects

1. Algebraic effects

1.3. Generic effects

Generalizing one boolean location

Recall: one boolean location

signatures: $put_t : 1$, $put_f : 1$, $get : 2$

examples: $put_t(m)$, $put_f(n)$, $get(m, n)$

Here is a question:

How to generalize it to locations which can store natural numbers?

Parameterized operations

locations of natural numbers

signatures: $\mathit{update} : \mathit{loc} \times \mathit{nat} \rightsquigarrow 1$, $\mathit{lookup} : \mathit{loc} \rightsquigarrow \mathit{nat}$

- Read *update* as "put *nat* to *loc*, and continue with the single continuation".
- Read *lookup* as "get the value from *loc*, and carry on with a continuation depending on the value read".

examples:

- $\mathit{update}((l, 42), m)$

Parameterized operations

locations of natural numbers

signatures: $update : loc \times nat \rightsquigarrow 1$, $lookup : loc \rightsquigarrow nat$

- Read *update* as "put *nat* to *loc*, and continue with the single continuation".
- Read *lookup* as "get the value from *loc*, and carry on with a continuation depending on the value read".

examples:

- $update((l, 42), m)$
- $lookup(l, m1, \ldots)$ — an infinitary operation!

Parameterized arguments

locations of natural numbers

signatures: $update : loc \times nat \rightsquigarrow 1$, $lookup : loc \rightsquigarrow nat$

- Read *update* as "put *nat* to *loc*, and continue with the single continuation".
- Read *lookup* as "get the value from *loc*, and carry on with a continuation depending on the value read".

examples:

- $update((l, 42), \lambda x : 1.\ m)$
- $lookup(l, \lambda x : nat.\ m)$ — with parameterized arguments

Parameterized arguments

locations of natural numbers

signatures: $\mathit{update} : \mathit{loc} \times \mathit{nat} \rightsquigarrow 1$, $\mathit{lookup} : \mathit{loc} \rightsquigarrow \mathit{nat}$

- Read *update* as "put *nat* to *loc*, and continue with the single continuation".
- Read *lookup* as "get the value from *loc*, and carry on with a continuation depending on the value read".

examples:

- $\mathit{update}((l, 42), \lambda x : 1.\ m)$
- $\mathit{lookup}(l, \lambda x : \mathit{nat}.\ m)$ — with parameterized arguments

axioms:

- $\mathit{update}((l, m), \mathit{update}((l, n), p)) = \mathit{update}((l, n), p)$
- $\mathit{lookup}(l, \lambda x.\ \mathit{lookup}(l, \lambda y.\ f(x, y))) = \mathit{lookup}(l, \lambda x.\ f(x, x))$
- ......

Algebraicity

Programming counterpart of algebraicity:

Evaluation context $\quad E ::= \Box \mid E\ n \mid (\lambda x.\ m)\ E$

For any operation $op : p \rightsquigarrow n$, we have:

$$E[op(p, \lambda x : n.\ m)] = op(p, \lambda x : n.\ E[m])$$

Algebraicity

Programming counterpart of algebraicity:

$$\textbf{Evaluation context} \qquad E ::= \Box \mid E\ n \mid (\lambda x.\ m)\ E$$

For any operation $op : p \rightsquigarrow n$, we have:

$$E[op(p, \lambda x : n.\ m)] = op(p, \lambda x : n.\ E[m])$$

Example

- $lookup(l, \lambda x.\ m)\ n = lookup(l, \lambda x.\ m\ n)$
- $(\lambda y.\ m')\ lookup(l, \lambda x.\ m) = lookup(l, \lambda x.\ (\lambda y.\ m')\ m)$

Generic effects

algebraic operations

signatures: $update : loc \times nat \rightsquigarrow 1$, $lookup : loc \rightsquigarrow nat$

examples:

- $update((l, 42), \lambda x : 1.\ m)$
- $lookup(l, \lambda x : nat.\ m)$

Generic effects

algebraic operations

signatures: $\mathit{update} : \mathit{loc} \times \mathit{nat} \rightsquigarrow 1$, $\mathit{lookup} : \mathit{loc} \rightsquigarrow \mathit{nat}$

examples:

- $\mathit{update}((l, 42), \lambda x : 1.\ m)$
- $\mathit{lookup}(l, \lambda x : \mathit{nat}.\ m)$

generic effects

$$\frac{m : \mathit{loc} \times \mathit{nat}}{\mathit{gen\_update}\ m : 1} \qquad \frac{n : \mathit{loc}}{\mathit{gen\_lookup}\ n : \mathit{nat}}$$

examples:

- $\mathit{gen\_update}((l, 42)) : 1$
- $\mathit{gen\_lookup}(l) : \mathit{nat}$

Generic effects

algebraic operations

signatures: $update : loc \times nat \rightsquigarrow 1,\ lookup : loc \rightsquigarrow nat$

examples:

- $update((l, 42), \lambda x : 1.\ m)$
- $lookup(l, \lambda x : nat.\ m)$

generic effects

$$\frac{m : loc \times nat}{gen\_update\ m : 1} \qquad \frac{n : loc}{gen\_lookup\ n : nat}$$

examples:

- $gen\_update((l, 42)) : 1$
- $gen\_lookup(l) : nat$

Question: are they equivalent?

Generic effects

$$\begin{array}{rcl} \textbf{algebraic operations} & & \textbf{generic effects} \\ \mathit{gen\_update}(l, 42) & = & \mathit{update}((l, 42), \lambda x.\ x) \\ \mathit{update}((l, 42), \lambda x.\ m) & = & \textbf{let } x = \mathit{gen\_update}((l, 42)) \textbf{ in } m \end{array}$$

Generic effects

$$
\begin{array}{rcl}
\textbf{algebraic operations} & & \textbf{generic effects} \\
\mathit{gen\_update}(l, 42) & = & \mathit{update}((l, 42), \lambda x.\ x) \\
\mathit{update}((l, 42), \lambda x.\ m) & = & \textbf{let } x = \mathit{gen\_update}((l, 42)) \textbf{ in } m
\end{array}
$$

Applied Categorical Structures **11:** 69–94, 2003.

Algebraic Operations and Generic Effects

GORDON PLOTKIN and JOHN POWER★
School of Informatics, University of Edinburgh, King's Buildings, Edinburgh EH9 3JZ, Scotland, U.K.

Abstract. Given a complete and cocomplete symmetric monoidal closed category V and a symmetric monoidal V-category C with cotensors and a strong V-monad T on C, we investigate axioms under which an $Ob\,C$-indexed family of operations of the form $\alpha_x : (Tx)^v \to (Tx)^w$ provides semantics for algebraic operations on the computational λ-calculus. We recall a definition for which we have elsewhere given adequacy results, and we show that an enrichment of it is equivalent to a

A small calculus

expressions

$e := c \mid x \mid \lambda x.\ e \mid e_1\ e_2 \mid \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid op\ e$

$v := c \mid \lambda x.\ e$

evaluation context $E ::= \Box \mid E\ e \mid v\ E \mid \textbf{if}\ E\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid op\ E$

reduction

$$
\begin{array}{lcl}
(\lambda x.\ e)\ v & \longrightarrow & e[x := v] \\
\textbf{if True then}\ e_2\ \textbf{else}\ e_3 & \longrightarrow & e_2 \\
\textbf{if False then}\ e_2\ \textbf{else}\ e_3 & \longrightarrow & e_3
\end{array}
$$

An example program

effect signature

$$\mathit{choose} : () \rightsquigarrow \mathit{Bool} \qquad \mathit{fail} : () \rightsquigarrow a$$

drunk coin tossing

$$\mathit{toss}\ () = \mathbf{if}\ \mathit{choose}\ ()\ \mathbf{then}\ \mathit{Heads}\ \mathbf{else}\ \mathit{Tails}$$

An example program

effect signature

$$\mathit{choose} : () \rightsquigarrow \mathit{Bool} \qquad \mathit{fail} : () \rightsquigarrow a$$

drunk coin tossing

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

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

How to destruct effect operations?

An example program

effect signature

$$\mathit{choose} : () \rightsquigarrow \mathit{Bool} \qquad \mathit{fail} : () \rightsquigarrow a$$

drunk coin tossing

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

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

How to destruct effect operations?

Next:

- effect handlers

2. Effect handlers

2. Effect handlers

2.1. Introductions

What is an effect handler?

- Effect "destructors"
- A (modular) interpretation of effect operations
- Resumable exception handlers
- A fold over the computational tree
- Structured delimited control operator
- A morphism between (free) algebras

Syntax

$$\textbf{handle } \{op \mapsto \lambda x\ k.\ e_1, return \mapsto \lambda x.\ e_2\}\ e$$

- e: the computation being handled

Syntax

$$\textbf{handle}\ \{op \mapsto \lambda x\ k.\ e_1, return \mapsto \lambda x.\ e_2\}\ e$$

- e: the computation being handled
- $op \mapsto \lambda x\ k.\ e_1$: handles the operation op
  - x binds the operation argument
  - k binds the delimited continuation
  - evaluates e_1

Syntax

$$\textbf{handle}\ \{op \mapsto \lambda x\ k.\ e_1, return \mapsto \lambda x.\ e_2\}\ e$$

- e: the computation being handled
- $op \mapsto \lambda x\ k.\ e_1$: handles the operation op
  - x binds the operation argument
  - k binds the delimited continuation
  - evaluates e_1
- $return \mapsto \lambda x.\ e_2$: applies when the computation returns a value
  - x binds the value
  - evaluates e_2

What's the delimited continuation?

The delimited continuation captures

- from where the operation is performed
- to where the handler is in the evaluation context

What's the delimited continuation?

The delimited continuation captures

- from where the operation is performed
- to where the handler is in the evaluation context

$$\overbrace{\textbf{handle}\ h\ E}^{\textit{delimited continuation}}\ [\mathit{op}\ v] \qquad \text{where } \mathit{op} \# E$$

What's the delimited continuation?

The delimited continuation captures

- from where the operation is performed
- to where the handler is in the evaluation context

$$\overbrace{\textbf{handle}\ h\ E}^{\textit{delimited continuation}}\ [\mathit{op}\ v] \qquad \text{where } \mathit{op} \# E$$

- x binds v
- k binds ???

What's the delimited continuation?

The delimited continuation captures

- from where the operation is performed
- to where the handler is in the evaluation context

$$\overbrace{\textbf{handle}\ h\ E}^{\textit{delimited continuation}}\ [op\ v] \qquad \text{where } op \# E$$

- x binds v
- k binds ??? $\lambda x.\ \textbf{handle}\ h\ E[x]$

Examples

effect handler

$$
\mathit{maybeFail} = \{\\
\quad \mathit{fail} \mapsto \lambda x\ k.\ \textbf{Nothing}\ ,\\
\quad \mathit{return} \mapsto \lambda x.\ \textbf{Just}\ \ x\ \}
$$

examples

$$\textbf{handle}\ \mathit{maybeFail}\ 42 \longrightarrow \textbf{Just}\ 42$$

$$\textbf{handle}\ \mathit{maybeFail}\ (\mathit{fail}\ ()) \longrightarrow \textbf{Nothing}$$

Examples

effect handler

$$
\begin{aligned}
&\mathit{maybeFail} = \{\\
&\quad \mathit{fail} \mapsto \lambda x\ k.\ \textbf{Nothing}\ ,\\
&\quad \mathit{return} \mapsto \lambda x.\ \textbf{Just}\ \ x\ \}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{trueChoice} = \{\\
&\quad \mathit{choose} \mapsto \lambda x\ k.\ k\ \textbf{True},\\
&\quad \mathit{return} \mapsto \lambda x.\ x\ \}
\end{aligned}
$$

examples

$\textbf{handle}\ \mathit{maybeFail}\ 42 \longrightarrow \textbf{Just}\ 42$

$\textbf{handle}\ \mathit{maybeFail}\ (\mathit{fail}\ ()) \longrightarrow \textbf{Nothing}$

$\textbf{handle}\ \mathit{trueChoice}\ 42 \longrightarrow 42$

$\textbf{handle}\ \mathit{trueChoice}\ (\mathit{toss}\ ()) \longrightarrow \mathit{Heads}$

Examples

effect handler

$$maybeFail = \{ \\ \quad fail \mapsto \lambda x\ k.\ \mathbf{Nothing}\ , \\ \quad return \mapsto \lambda x.\ \mathbf{Just}\ \ x\ \}$$

$$trueChoice = \{ \\ \quad choose \mapsto \lambda x\ k.\ k\ \mathbf{True}, \\ \quad return \mapsto \lambda x.\ x\ \}$$

$$allChoices = \{ \\ \quad choose \mapsto \lambda x\ k.\ k\ \mathbf{True} +\!\!+\ k\ \mathbf{False}, \\ \quad return \mapsto \lambda x.\ [x]\ \}$$

examples

$$\mathbf{handle}\ maybeFail\ 42 \longrightarrow \mathbf{Just}\ 42$$

$$\mathbf{handle}\ maybeFail\ (fail\ ()) \longrightarrow \mathbf{Nothing}$$

$$\mathbf{handle}\ trueChoice\ 42 \longrightarrow 42$$

$$\mathbf{handle}\ trueChoice\ (toss\ ()) \longrightarrow Heads$$

$$\mathbf{handle}\ allChoices\ 42 \longrightarrow [42]$$

$$\mathbf{handle}\ allChoices\ (toss\ ()) \longrightarrow [Heads, Tails]$$

Handler composition

effect composition

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

handler composition

$$
\begin{aligned}
& \textbf{handle}\ \mathit{allChoices}\ (\textbf{handle}\ \mathit{maybeFail}\ (\mathit{drunkToss}\ ())) \\
& \qquad \longrightarrow [\textbf{Just}\ \mathit{Heads},\ \textbf{Just}\ \mathit{Tails},\ \textbf{Nothing}\ ]
\end{aligned}
$$

Handler composition

effect composition

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

handler composition

$$\begin{aligned} & \textbf{handle}\ \mathit{allChoices}\ (\textbf{handle}\ \mathit{maybeFail}\ (\mathit{drunkToss}\ ())) \\ & \qquad \longrightarrow [\textbf{Just}\ \mathit{Heads},\ \textbf{Just}\ \mathit{Tails},\ \textbf{Nothing}\ ] \end{aligned}$$

$$\textbf{handle}\ \mathit{maybeFail}\ (\textbf{handle}\ \mathit{allChoices}\ (\mathit{drunkToss}\ ())) \longrightarrow \textbf{Nothing}$$

A small calculus

expressions

$$e := c \mid x \mid \lambda x.\ e \mid e_1\ e_2 \mid \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid \mathit{op}\ e$$

$$v := c \mid \lambda x.\ e$$

evaluation context $\quad E ::= \square \mid E\ e \mid v\ E \mid \textbf{if}\ E\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid \mathit{op}\ E$

reduction

$$\begin{array}{lcl} (\lambda x.\ e)\ v & \longrightarrow & e[x := v] \\ \textbf{if True then}\ e_2\ \textbf{else}\ e_3 & \longrightarrow & e_2 \\ \textbf{if False then}\ e_2\ \textbf{else}\ e_3 & \longrightarrow & e_3 \end{array}$$

A small calculus

expressions

$$e := c \mid x \mid \lambda x.\ e \mid e_1\ e_2 \mid \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid \mathit{op}\ e \mid \textbf{handle}\ h\ e$$

$$v := c \mid \lambda x.\ e$$

$$h := \{\mathit{op} \mapsto \lambda x\ k.\ e_1, \mathit{return} \mapsto \lambda x.\ e_2\}$$

evaluation context $\quad E ::= \square \mid E\ e \mid v\ E \mid \textbf{if}\ E\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid \mathit{op}\ E$

reduction

$$\begin{array}{lcl} (\lambda x.\ e)\ v & \longrightarrow & e[x := v] \\ \textbf{if True then}\ e_2\ \textbf{else}\ e_3 & \longrightarrow & e_2 \\ \textbf{if False then}\ e_2\ \textbf{else}\ e_3 & \longrightarrow & e_3 \end{array}$$

A small calculus

expressions

$$e := c \mid x \mid \lambda x.\ e \mid e_1\ e_2 \mid \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid \mathit{op}\ e \mid \textbf{handle}\ h\ e$$

$$v := c \mid \lambda x.\ e$$

$$h := \{\mathit{op} \mapsto \lambda x\ k.\ e_1, \mathit{return} \mapsto \lambda x.\ e_2\}$$

evaluation context $E ::= \square \mid E\ e \mid v\ E \mid \textbf{if}\ E\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid \mathit{op}\ E \mid \textbf{handle}\ h\ E$

reduction

$$\begin{array}{lcl} (\lambda x.\ e)\ v & \longrightarrow & e[x := v] \\ \textbf{if True then}\ e_2\ \textbf{else}\ e_3 & \longrightarrow & e_2 \\ \textbf{if False then}\ e_2\ \textbf{else}\ e_3 & \longrightarrow & e_3 \end{array}$$

A small calculus

expressions

$$e := c \mid x \mid \lambda x.\ e \mid e_1\ e_2 \mid \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid \mathit{op}\ e \mid \textbf{handle}\ h\ e$$

$$v := c \mid \lambda x.\ e$$

$$h := \{\mathit{op} \mapsto \lambda x\ k.\ e_1, \mathit{return} \mapsto \lambda x.\ e_2\}$$

evaluation context $E ::= \Box \mid E\ e \mid v\ E \mid \textbf{if}\ E\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid \mathit{op}\ E \mid \textbf{handle}\ h\ E$

reduction

$$\begin{array}{lcll}
(\lambda x.\ e)\ v & \longrightarrow & e[x := v] & \\
\textbf{if True then}\ e_2\ \textbf{else}\ e_3 & \longrightarrow & e_2 & \\
\textbf{if False then}\ e_2\ \textbf{else}\ e_3 & \longrightarrow & e_3 & \\
\textbf{handle}\ h\ v & \longrightarrow & f\ v & \text{where } \mathit{return} \mapsto f \in h \\
\textbf{handle}\ h\ E[\mathit{op}\ v] & \longrightarrow & f\ v\ (\lambda x.\ \textbf{handle}\ h\ E[x]) & \text{where } \mathit{op} \mapsto f \in h, \mathit{op} \# E
\end{array}$$

Example: lightweight cooperative threads

effect signature

$$\mathit{yield} : () \rightsquigarrow () \qquad \mathit{fork} : (() \to ()) \rightsquigarrow ()$$

$$\begin{aligned}
&\mathit{scheduler}\ f = \textbf{handle}\ \{\\
&\qquad \mathit{yield} \mapsto \lambda x\ k.\ \mathit{enqueue}\ k; \mathit{next} \leftarrow \mathit{dequeue}\ (); \mathit{next}\ (),\\
&\qquad \mathit{fork} \mapsto \lambda g\ k.\ \mathit{enqueue}\ k; \mathit{scheduler}\ g,\\
&\qquad \mathit{return} \mapsto \lambda _.\ \mathit{next} \leftarrow \mathit{dequeue}\ (); \mathit{next}\ ()\ \}\ (f\ ())
\end{aligned}$$

Example: lightweight cooperative threads

effect signature

$$yield : () \rightsquigarrow () \qquad fork : (() \to ()) \rightsquigarrow ()$$

$$\begin{aligned}
&scheduler\ f = \textbf{handle}\ \ \{ \\
&\qquad yield \mapsto \lambda x\ k.\ enqueue\ k; next \leftarrow dequeue\ (); next\ (), \\
&\qquad fork \mapsto \lambda g\ k.\ enqueue\ k; scheduler\ g, \\
&\qquad return \mapsto \lambda_.\ next \leftarrow dequeue\ (); next\ ()\ \}\ (f\ ())
\end{aligned}$$

$$\begin{aligned}
scheduler\ (\lambda_.\ \ &print\ "A"; fork\ (\lambda_.\ print\ "B"; yield\ (); print\ "E"); \\
&print\ "C"; fork\ (\lambda_.\ print\ "D"; yield\ (); print\ "G"); print "F")
\end{aligned}$$

Question: What will be printed?

Operational semantics zoo

deep handlers

$$\textbf{handle}\ h\ E[op\ v] \longrightarrow f\ v\ (\lambda x.\ \textbf{handle}\ h\ E[x]) \qquad \text{where } op \mapsto f \in h, op\#E$$

Operational semantics zoo

deep handlers

$$\textbf{handle}\ h\ E[\mathit{op}\ v] \longrightarrow f\ v\ (\lambda x.\ \textbf{handle}\ h\ E[x]) \qquad \text{where } \mathit{op} \mapsto f \in h, \mathit{op} \# E$$

shallow handlers

$$\textbf{handle}\ h\ E[\mathit{op}\ v] \longrightarrow f\ v\ (\lambda x.\ E[x]) \qquad \text{where } \mathit{op} \mapsto f \in h, \mathit{op} \# E$$

Operational semantics zoo

deep handlers

$$\textbf{handle}\ h\ E[op\ v] \longrightarrow f\ v\ (\lambda x.\ \textbf{handle}\ h\ E[x]) \qquad \text{where } op \mapsto f \in h, op\#E$$

shallow handlers

$$\textbf{handle}\ h\ E[op\ v] \longrightarrow f\ v\ (\lambda x.\ E[x]) \qquad \text{where } op \mapsto f \in h, op\#E$$

sheep handlers

$$\textbf{handle}\ h\ E[op\ v] \longrightarrow f\ v\ (\lambda h'.\ \lambda x.\ \textbf{handle}\ \ h'\ E[x]) \qquad \text{where } op \mapsto f \in h, op\#E$$

- like shallow handlers: the original handler need not be installed
- like deep handlers: a handler (not necessarily the original one) must be installed

Operational semantics zoo

deep handlers

$$\textbf{handle}\ h\ E[\mathit{op}\ v] \longrightarrow f\ v\ (\lambda x.\ \textbf{handle}\ h\ E[x]) \qquad \text{where } \mathit{op} \mapsto f \in h, \mathit{op}\#E$$

shallow handlers

$$\textbf{handle}\ h\ E[\mathit{op}\ v] \longrightarrow f\ v\ (\lambda x.\ E[x]) \qquad \text{where } \mathit{op} \mapsto f \in h, \mathit{op}\#E$$

sheep handlers

$$\textbf{handle}\ h\ E[\mathit{op}\ v] \longrightarrow f\ v\ (\lambda h'.\ \lambda x.\ \textbf{handle}\ \ h'\ E[x]) \qquad \text{where } \mathit{op} \mapsto f \in h, \mathit{op}\#E$$

- like shallow handlers: the original handler need not be installed
- like deep handlers: a handler (not necessarily the original one) must be installed

parameterized handlers

$$\textbf{handle}\ h\ s\ E[\mathit{op}\ v] \longrightarrow f\ s\ v\ (\lambda s'\ x.\ \textbf{handle}\ h\ s'\ E[x]) \qquad \text{where } \mathit{op} \mapsto f \in h, \mathit{op}\#E$$

Operational semantics zoo

deep handlers

$$\textbf{handle } h\ E[op\ v] \longrightarrow f\ v\ (\lambda x.\ \textbf{handle } h\ E[x]) \qquad \text{where } op \mapsto f \in h, op\#E$$

shallow handlers

$$\textbf{handle } h\ E[op\ v] \longrightarrow f\ v\ (\lambda x.\ E[x]) \qquad \text{where } op \mapsto f \in h, op\#E$$

sheep handlers

$$\textbf{handle } h\ E[op\ v] \longrightarrow f\ v\ (\lambda h'.\ \lambda x.\ \textbf{handle }\ h'\ E[x]) \qquad \text{where } op \mapsto f \in h, op\#E$$

- like shallow handlers: the original handler need not be installed
- like deep handlers: a handler (not necessarily the original one) must be installed

parameterized handlers (exercise: express parameterized handlers as deep handlers)

$$\textbf{handle } h\ s\ E[op\ v] \longrightarrow f\ s\ v\ (\lambda s'\ x.\ \textbf{handle } h\ s'\ E[x]) \qquad \text{where } op \mapsto f \in h, op\#E$$

Example: shallow handlers

A classic demand-driven Unix pipeline operator [Hillerstrom and Lindley]

$pipe(p, c) = \textbf{handle}\ \{Await \mapsto \lambda x\ k.\ copipe(k, p)\}\ (c\ ())$

$copipe(c, p) = \textbf{handle}\ \{Yield \mapsto \lambda x\ k.\ pipe(k, \lambda\_.\ c\ x)\}\ (p\ ())$

Example: shallow handlers

A classic demand-driven Unix pipeline operator [Hillerstrom and Lindley]

$$pipe(p, c) = \textbf{handle}\ \{Await \mapsto \lambda x\ k.\ copipe(k, p)\}\ (c\ ())$$

$$copipe(c, p) = \textbf{handle}\ \{Yield \mapsto \lambda x\ k.\ pipe(k, \lambda\_.\ c\ x)\}\ (p\ ())$$

As an example:

$$ones = \lambda\_.\ Yield\ 1; ones\ ()$$

$$pipe(ones, \lambda\_.\ Await\ ())$$

Operational semantics zoo (cont'd)

Question: what if I want my operation to be handled by the second innermost handler?

Operational semantics zoo (cont'd)

Question: what if I want my operation to be handled by the second innermost handler?

masking (also known as lifting)

expressions $\quad e := \ldots \mid \mathit{lift}[\mathit{op}]\ e$

$\textbf{handle}\ h\ E[\mathit{op}\ v] \longrightarrow f\ v\ (\lambda x.\ \textbf{handle}\ h\ E[x]) \qquad$ where $\mathit{op} \mapsto f \in h$, $0\text{-}\mathit{free}(\mathit{op}, E)$

$$\frac{}{0\text{-}\mathit{free}(\mathit{op}, \square)} \qquad \frac{n\text{-}\mathit{free}(\mathit{op}, E)}{n+1\text{-}\mathit{free}(\mathit{op}, \mathit{lift}[\mathit{op}]\ E)} \qquad \frac{n+1\text{-}\mathit{free}(\mathit{op}, E)}{n\text{-}\mathit{free}(\mathit{op}, \textbf{handle}\ \{\mathit{op} \mapsto e\}\ E)}$$

As an example:

$???\text{-}\mathit{free}(\mathit{op}, \textbf{handle}\ \{\mathit{op} \mapsto e\}\ (\mathit{lift}[\mathit{op}]\ \square))$

Named handlers

named handlers [Xie et al. 2022]

$\textbf{handler}\ h\ (\lambda x.\ e) \longrightarrow \textbf{handle}_\ell\ h\ (e\,[x := \ell]\,) \qquad \text{fresh } \ell$

$\textbf{handle}_\ell\ h\ E[\ \mathit{op}\ \ell\ \ v] \longrightarrow f\ \ v\ (\lambda x.\ \textbf{handle}_{\,\ell}\ h\ E[x]) \qquad \text{where } \mathit{op} \mapsto f \in h$

Named handlers

named handlers [Xie et al. 2022]

$$\textbf{handler}\ h\ (\lambda x.\ e) \longrightarrow \textbf{handle}_\ell\ h\ (e\,[x := \ell]\,) \qquad \text{fresh } \ell$$

$$\textbf{handle}_\ell\ h\ E[\,op\ \ell\ \ v] \longrightarrow f\ v\ (\lambda x.\ \textbf{handle}_\ell\ h\ E[x]) \qquad \text{where } op \mapsto f \in h$$

As an example:

$$read\ x = \{Read \mapsto \lambda\_\ k.\ k\ x\}$$

$$\textbf{handler}\ (read\ 21)\ (\lambda y.\ \textbf{handler}\ (read\ 42)\ (\lambda z.\ (Read\ y\ ()) + (Read\ z\ ())))$$