Handlers in Action

Ohad Kammar
<ohad.kammar@cl.cam.ac.uk>

Higher-Order Programming and Effects (HOPE)
September 8, 2012
joint with
Sam Lindley and Nicolas Oury

Isaac Newton Trust
Computer Laboratory
sicsa*
THE UNIVERSITY of EDINBURGH informatics lfcs
Laboratory for Foundations of Computer Science

Exception handlers

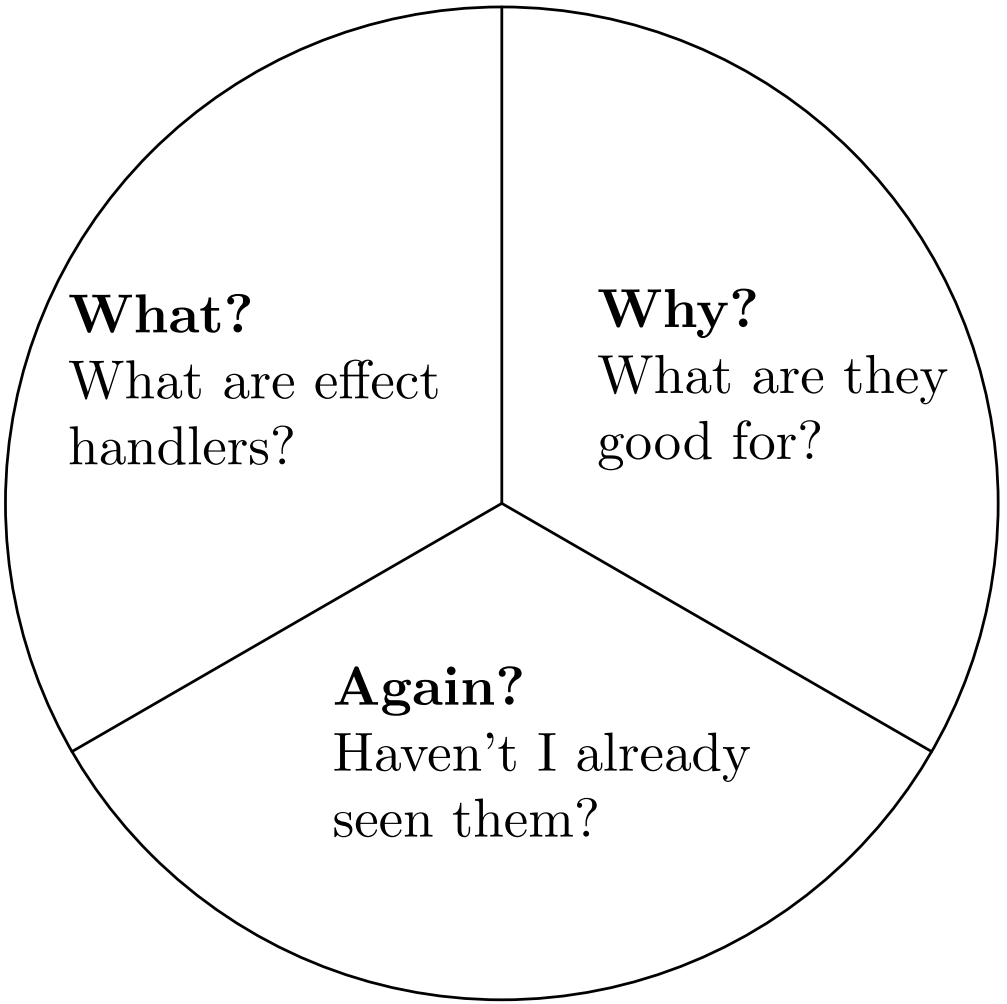
handle
  if $(\texttt{get}\ \ell) = 0$
  then `raise` *DivideByZero*
  else $42\ /\ (\texttt{get}\ \ell)$
with
  DivideByZero $\mapsto 0$
  e $\mapsto$ `raise` e

  return $x \mapsto$ `display` x

Effect handlers

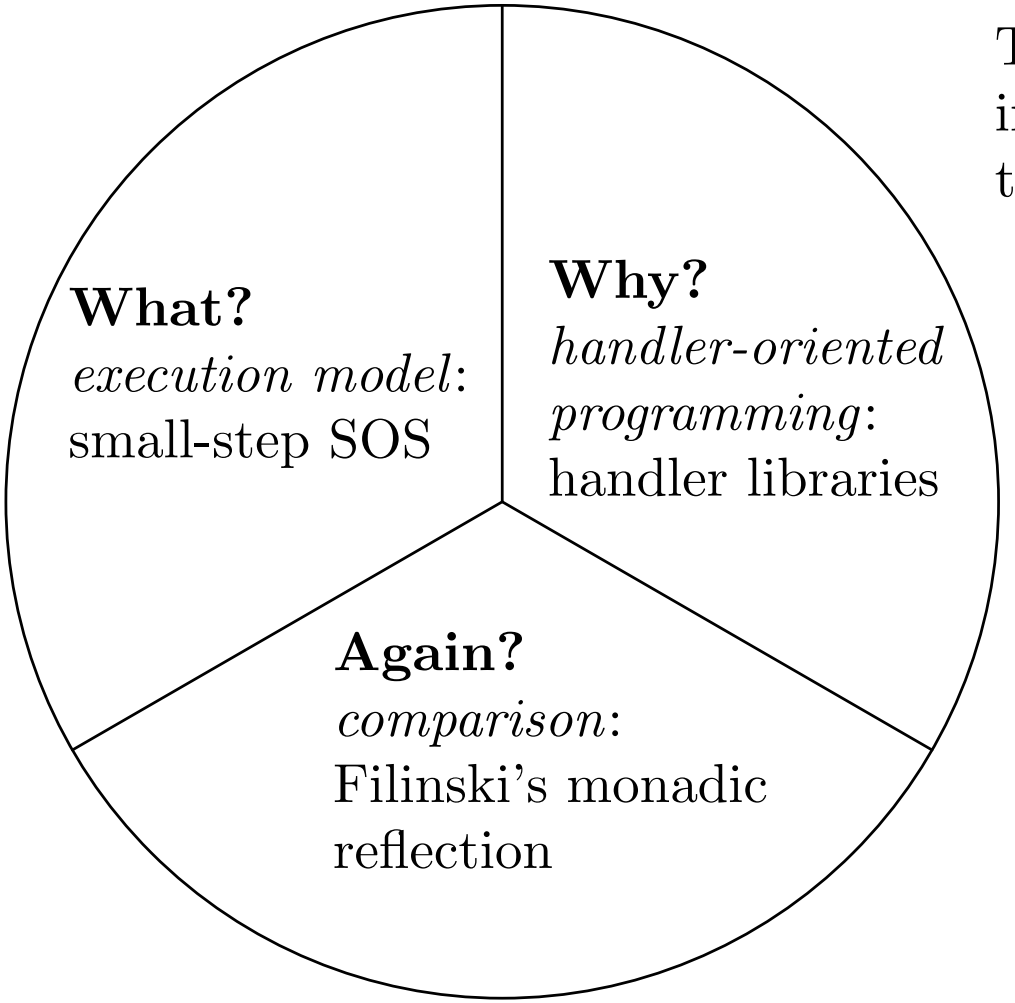
handle
  if $(\texttt{get}\ \ell) = 0$
  then `raise` *DivideByZero*
  else $42\ /\ (\texttt{get}\ \ell)$
with
  `raise` *DivideByZero* $k \mapsto 0$
  `raise` e $k \mapsto \texttt{raise}\ e$
  `get` l $k \mapsto k\ (1)$
  return $x \mapsto \texttt{display}\ x$

Addressed questions



Contribution

- functional language with handlers
- sound type-and-effect system

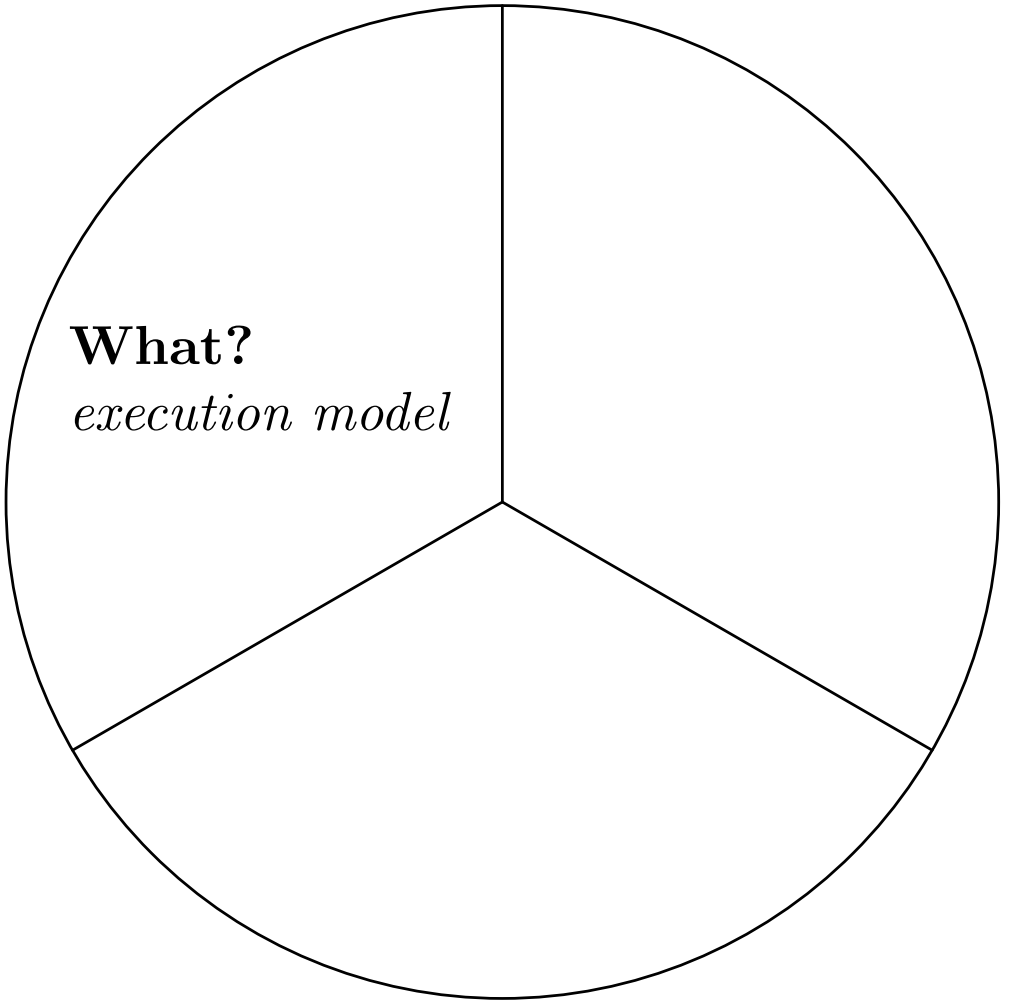


Two implementation techniques:

- free monads in Haskell;
- (delimited) control operators in SML and Racket;

Goal

Facilitate operational discussion.


What?
execution model

Algebraic effects

Operations, parameter types, arities

$$\mathtt{op} : \mathit{Pa} \to \mathit{Ar}$$

For example:

$$\begin{aligned}
\mathtt{lookup} &: \mathit{Loc} &&\to \mathit{Integer} \\
\mathtt{update} &: (\mathit{Loc}, \mathit{Integer}) &&\to \mathit{Unit} \\
\mathtt{raise} &: \mathit{Exception} &&\to \mathit{Empty}
\end{aligned}$$

Usage

$$\mathtt{op}\ V\ (\lambda x \to M)$$

For example:

$$\mathtt{lookup}\ \ell\ (\lambda i \to \mathtt{update}\ (\ell, i+1)\ (\lambda\_ \to ()))$$

Algebraic effects (cntd.)

Generic effects

Another familiar variant:

$$\texttt{gen}\ V = \texttt{op}\ V\ (\lambda x \to x)$$

For example:

$$\begin{aligned}
\texttt{get}\ \ell &= \texttt{lookup}\ \ell\ (\lambda i \to i) \\
\texttt{set}\ (\ell, i) &= \texttt{update}\ (\ell, i)\ (\lambda _ \to ()) \\
\texttt{raise}\ e &= \texttt{raise}\ e\ (\lambda z \to \texttt{whatever}\ z)
\end{aligned}$$

λ_{eff}-calculus

Syntax

- Value terms V
- Computation terms

$$M ::= \ldots \mid \mathrm{op}\ V\ (\lambda x \to M) \mid \textbf{handle}\ M\ \textbf{with}\ H$$

- Handlers

$$H ::= \mathrm{op}\ p\ k \mapsto M$$
$$\ldots$$
$$\textbf{return}\ x \mapsto N$$

For example:

$$\texttt{raise}\ \textit{DivideByZero}\ k \mapsto 0$$
$$\texttt{raise}\ e \qquad k \mapsto \texttt{raise}\ e$$
$$\texttt{lookup}\ l \qquad k \mapsto k\ (1)$$
$$\textbf{return}\ x \mapsto \texttt{display}\ x$$

Reduction rules

handle
  if $(\texttt{get}\ \ell) = 0$
  then `raise` *DivideByZero*
  else $42\ /\ (\texttt{get}\ \ell)$
with
  `raise` *DivideByZero* *k* $\mapsto 0$
  `raise` *e* *k* $\mapsto$ `raise` *e*
  `lookup` *l* *k* $\mapsto$ *k* (1)
  return *x* $\mapsto$ `display` *x*

Reduction rules

handle
  if $(\texttt{lookup}\ \ell$
    $(\lambda i \to i))$
    $= 0$
  then M_1
  else M_2
with H

Reduction rules

$$\begin{array}{l}\textbf{handle}\\ \quad \textbf{if } (\texttt{lookup } \ell\\ \qquad (\lambda i \to i))\\ \qquad = 0\\ \qquad \textbf{then } M_1\\ \qquad \textbf{else } M_2\\ \textbf{with } H\end{array} \xrightarrow{\text{hoist}} \begin{array}{l}\textbf{handle}\\ \quad \texttt{lookup } \ell\ (\lambda i \to\\ \qquad \textbf{if } i\\ \qquad\\ \qquad = 0\\ \qquad \textbf{then } M_1\\ \qquad \textbf{else } M_2\\ \quad)\\ \textbf{with } H\end{array}$$

More generally:

$$\mathcal{H}[\text{op } V\ (\lambda x \to M)] \xrightarrow{\text{hoist}} \text{op } V\ (\lambda x \to \mathcal{H}[M])$$

for hoisting frames $\mathcal{H}[-]$ with $x \notin FV(\mathcal{H})$.

Reduction rules (cntd.)

$$
\begin{array}{l}
\textbf{handle} \\
\quad \texttt{lookup}\ \ell\ (\lambda i \to \\
\qquad \textbf{if}\ i = 0 \\
\qquad \textbf{then}\ M_1 \\
\qquad \textbf{else}\ M_2 \\
\quad) \\
\textbf{with} \\
\quad \ldots \\
\quad \texttt{lookup}\ l\ k \mapsto k\ (1)
\end{array}
\quad \xrightarrow{\text{op}} \quad
\begin{array}{l}
(\lambda i \to \\
\quad \textbf{handle} \\
\qquad \textbf{if}\ i = 0 \\
\qquad \textbf{then}\ M_1 \\
\qquad \textbf{else}\ M_2 \\
\quad \textbf{with}\ H \\
)\ (1)
\end{array}
$$

More generally, for handler H satisfying $x \notin FV(H)$:

$$
\begin{array}{l}
\textbf{handle}\ \text{op}\ V\ (\lambda x \to M) \\
\textbf{with} \\
\quad \ldots \\
\quad \text{op}\ p\ k \mapsto N \\
\quad \ldots
\end{array}
\quad \xrightarrow{\text{op}} N[V/p, (\lambda x \to \textbf{handle}\ M\ \textbf{with}\ H)/k]
$$

Reduction rules (cntd.)

$$
\begin{array}{l}
(\lambda i \rightarrow \\
\quad \textbf{handle} \\
\quad\quad \textbf{if } i = 0 \\
\quad\quad \textbf{then } M_1 \\
\quad\quad \textbf{else } M_2 \\
\quad \textbf{with } H \\
)\ (1)
\end{array}
\quad \xrightarrow{\beta}{}^{*} \quad
\textbf{handle } M_2 \textbf{ with } H
$$

Reduction rules (cntd.)

handle
42 / (get ℓ)
with H

$\xrightarrow{\text{hoist, op, } \beta\text{, arithmetic}}{}^{*}$ **handle** 42 **with** H

Reduction rules (cntd.)

$$\textbf{handle}\ 42\ \textbf{with}\ \ldots\ \textbf{return}\ x \mapsto \texttt{display}\ x \xrightarrow{\text{handler return}} \texttt{display}\ 42$$

More generally:

$$\textbf{handle}\ V\ \textbf{with}\ \ldots\ \textbf{return}\ x \mapsto N \xrightarrow{\text{handler return}} N\,[V\,/\,x]$$

λ_{eff}-calculus

Type-and-effect system

- Value types $A, B ::= \ldots \mid U_E\, C$.
- Computation types C.
- Effect signatures: (with Pa and Ar value types)

$$E ::= \{\mathrm{op} : Pa \to Ar, \ldots\}$$

- Handlers

$$R ::= A\ ^{E}\!\Rightarrow^{E'}\ C$$

Type-and-effect system (cntd.)

- Value type judgements $\Gamma \vdash V : C$.
- Computation type judgements $\Gamma \vdash_E M : C$:

$$\frac{\Gamma \vdash V : Pa \quad \Gamma, x : Ar \vdash_E M : C}{\Gamma \vdash_E \mathrm{op}\ V\ (\lambda x \to M) : C}(\mathrm{op} : Pa \to Ar \in E)$$

$$\frac{\Gamma \vdash_E M : FA \quad \Gamma \vdash H : A\ ^{E}\!\Rightarrow^{E'}\ C}{\Gamma \vdash_{E'} \textbf{handle}\ M\ \textbf{with}\ H : C}$$

Type-and-effect system (cntd.)

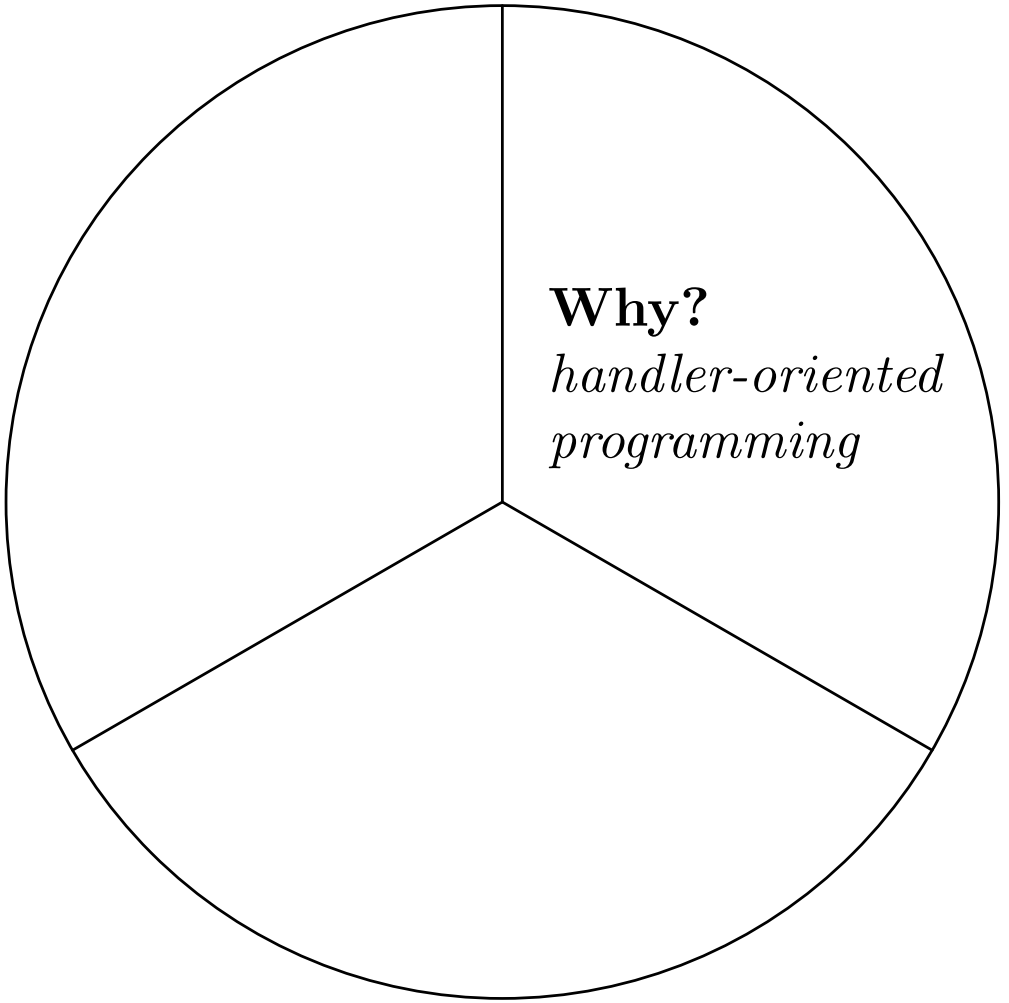
- Handler type judgements $\Gamma \vdash H : R$:

$$\frac{\begin{array}{c}\Gamma, p : Pa, k : U_E(Ar \to C) \vdash_E M : C \\ \cdots \\ \Gamma, x : A \vdash_E N : C\end{array}}{\Gamma \vdash \mathrm{op}\ p\ k \mapsto M} \qquad \cdots \qquad \mathbf{return}\ x \mapsto N : A\ ^{\{\mathrm{op}:Pa \to Ar, \ldots\}}\Rightarrow^{E}\ C$$

Note the placement of E's.

Type soundness

If $\vdash_{\{\}} M : FA$ then $M \to^* \mathrm{return} V$, for $\vdash V : A$.


Why?
handler-oriented
programming

User-defined effects

In Haskell

```
m = do
  fruit ← chooseFruit
  form ← chooseForm
  return $ form ++ fruit
```

Individually:

```
bothFruit :: [String]
bothFruit = ["apple","orange"]
randomForm :: IO String
randomForm = do
               x ← getStdRandom random
               if x then return "raw "
                    else return "cooked "
```

User-defined effects

In Haskell

```
m = do
  fruit ← chooseFruit
  form ← chooseForm
  return $ form ++ fruit
```

Combined:

```
result :: IO [String]
result = runListT m
chooseFruit = ListT $ return bothFruit
chooseForm = lift    $ randomForm
```

Handler-oriented programming

Horizontal composition

$$
\begin{aligned}
&\textbf{handle}\ (\textbf{do}\\
&\qquad\qquad \mathit{fruit} \leftarrow \mathit{chooseFruit}\\
&\qquad\qquad \mathit{form} \leftarrow \mathit{chooseForm}\\
&\qquad\qquad \textbf{return}\ \$\ \mathit{form} +\!\!+ \mathit{fruit})\\
&\quad (\mathit{ChooseFruit} \mapsto (\lambda p\ k \rightarrow \textbf{do}\ \mathit{xs} \leftarrow k\ \texttt{"apple"}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{ys} \leftarrow k\ \texttt{"orange"}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{return}\ (\mathit{xs} +\!\!+ \mathit{ys}))\triangleleft\\
&\quad \mathit{ChooseForm} \mapsto (\lambda p\ k \rightarrow \textbf{do}\ \{v \leftarrow \mathit{randomForm}; k\ v\})\triangleleft\\
&\quad \mathit{Empty},\\
&\quad \lambda x \rightarrow \textbf{return}\ [x])
\end{aligned}
$$

Handler-oriented programming

Vertical composition

```
handleListProbV :: IO [String]
handleListProbV =
  handle
    (handle do
       fruit ← chooseFruit
       form ← chooseForm
       return $ form ++ fruit)
       (ChooseFruit ↦
          (λp k → do xs ← k "apple"
                     ys ← k "orange"
                     return (xs ++ ys)) ◁ ChooseForm ⊣ Empty,
           λx → return [x]))
    (ChooseForm ↦ (λp k → do {v ← randomForm; k v}) ◁ Empty, return)
```

Handler-oriented programming

Evaluation

Is it better than monads? We don't know!

Bauer's thesis [private communication]

> *My experience with eff convinces me that we have*

$$\text{“}\textit{effects} + \textit{handlers}\text{”} \;:\; \text{“}\textit{delimited continuations}\text{”}$$
$$=$$
$$\text{“}\textit{while}\text{”} \;:\; \text{“}\textit{goto}\text{”}$$

Our contribution

Facilitate investigation: libraries in Haskell, SML, and Racket.

Implementation: free monads

Concretely

For $E = \{\texttt{raise} : \mathit{Exception} \to \mathit{Empty}, \texttt{lookup} : \mathit{Loc} \to \mathit{Integer}\}$:

$$\begin{aligned}\textbf{data}\ \mathit{Comp}\ a = {} & \mathit{Return}\ a \\ \mid {} & \mathit{Raise}\ \mathit{Exception} \\ \mid {} & \mathit{Lookup}\ (\mathit{Loc}, \mathit{Integer} \to \mathit{Comp}\ a)\end{aligned}$$

Consequently:

$$\begin{aligned}\texttt{raise}\ e &= \mathit{Raise}\ e \\ \texttt{lookup}\ \ell\ m &= \mathit{Lookup}\ (\ell, m)\end{aligned}$$

$$\begin{aligned}&\textbf{handle}\ (\mathit{Return}\ a) && \mathit{raiseC}\ \mathit{lookupC}\ \mathit{returnC} = \mathit{returnC}\ a \\ &\textbf{handle}\ (\mathit{Raise}\ e) && \mathit{raiseC}\ \mathit{lookupC}\ \mathit{returnC} = \mathit{raiseC}\ e \\ &\textbf{handle}\ (\mathit{Lookup}\ (\ell, m)) && \mathit{raiseC}\ \mathit{lookupC}\ \mathit{returnC} = \mathit{lookupC}\ \ell\ m\end{aligned}$$

Implementation: free monads (cntd.)

Typed implementation:

Option 1

Use dynamic types and casts:

$$\textbf{data}\ \mathit{Comp}\ a = \mathit{Return}\ a \mid \mathit{App}\ (\mathit{Op}, \mathit{Dyn}, \mathit{Dyn} \to \mathit{Comp}\ a)$$

Implementation: free monads (cntd.)

Option 2

Use GADTs and proxy types:

$$\begin{aligned}&\textbf{data}\ \mathit{Comp}\ e\ a :: \star\ \textbf{where}\\ &\quad \mathit{Ret}\ :: a \to \mathit{Comp}\ e\ a\\ &\quad \mathit{App} :: \mathit{Witness}\ \mathrm{op}\ e \to \mathrm{op} \to \mathit{Param}\ \mathrm{op} \to\\ &\qquad (\mathit{Arity}\ \mathrm{op} \to \mathit{Comp}\ e\ a) \to \mathit{Comp}\ e\ a\end{aligned}$$

- More expressive types (effect polymorphism)
  $\Longrightarrow$ code reuse.
- Technicalities suggest *row polymorphisms* as more suitable.

Get it from:

`https://github.com/slindley/effect-handlers`

Implementation: delimited control

Primitive control operators

shift0, **reset0**:

$$\textbf{reset0}\ (\mathcal{E}[\textbf{shift0}\ (\lambda k \to M)]) \to M\ [(\lambda x \to \textbf{reset0}\ (\mathcal{E}[x]))\ /\ k]$$

Compare with the derived:

$$\begin{aligned}\textbf{handle}\ \mathcal{H}[\mathrm{op}\ V\ (\lambda x \to M)]\ \textbf{with}\ \ldots\ \mathrm{op}\ p\ k \mapsto N \ldots \\ \to N\ [V\ /\ p, (\lambda x \to \textbf{handle}\ \mathcal{H}[M]\ \textbf{with}\ H)\ /\ k]\end{aligned}$$

Implementation: delimited control (cntd.)

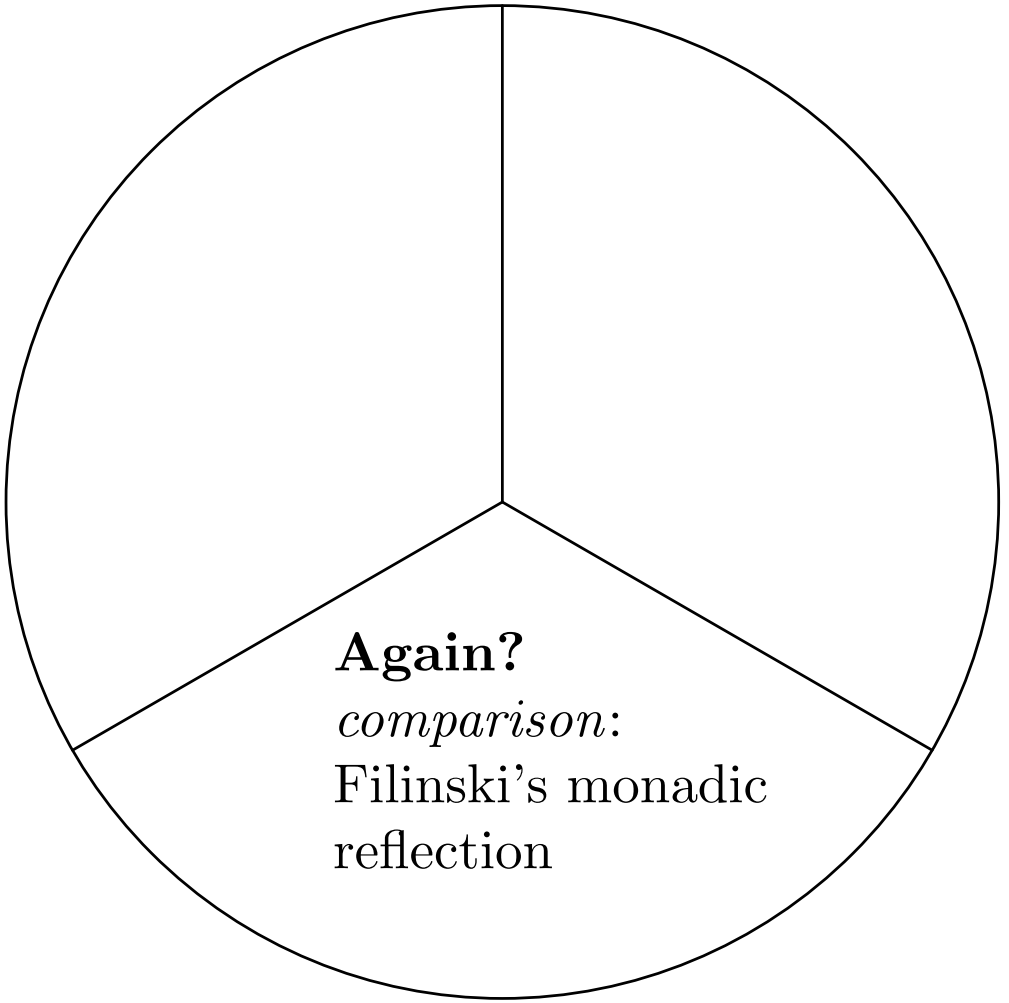
The **handle** construct **handle** M **with** H

- *push* current effect operation bindings from H onto a stack.
- **reset0** (M)

Effect operation op V $(\lambda x \to M)$

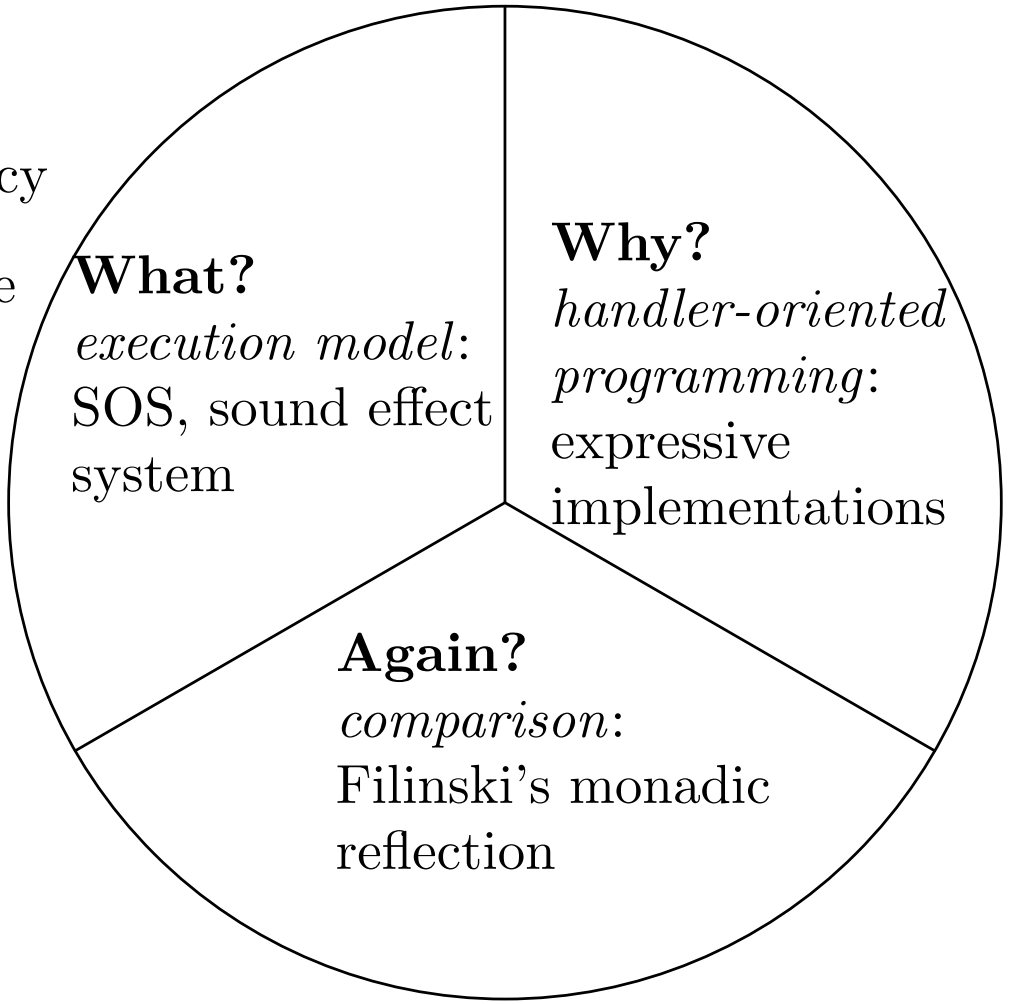
- **shift0** captures the hoisting context, concatenating it with M
- use the effect binding from top of the stack to execute op

and **return** is straightforward.


Again?
comparison:
Filinski's monadic
reflection

Conclusion

- Adequacy
- Non-free effects



- Evaluation
- Dynamic effect generation
- Performance

- delimited control

Try it, and join the discussion!

Images

- http://www.agriaffaires.co.uk/img_583/telescopic-handler/telescopic-handler.jpg
- http://ginavivinetto.files.wordpress.com/2008/09/chelsea-handler.jpg