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

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

Exception handlers

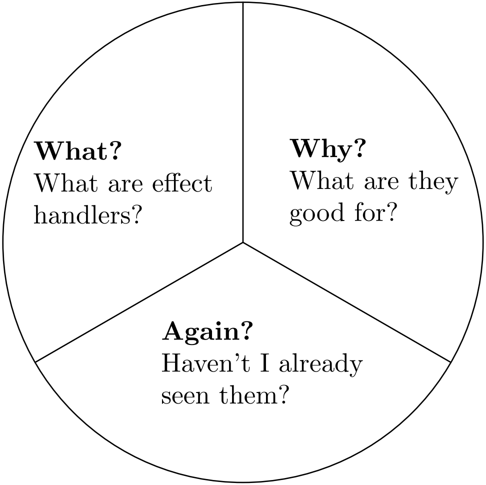
handle
  if $(\mathtt{get}\ \ell) = 0$
  then `raise` *DivideByZero*
  else $42\ /\ (\mathtt{get}\ \ell)$
with
  DivideByZero $\mapsto 0$
  e $\mapsto$ `raise` *e*

  return *x* $\mapsto$ `display` *x*

Effect handlers

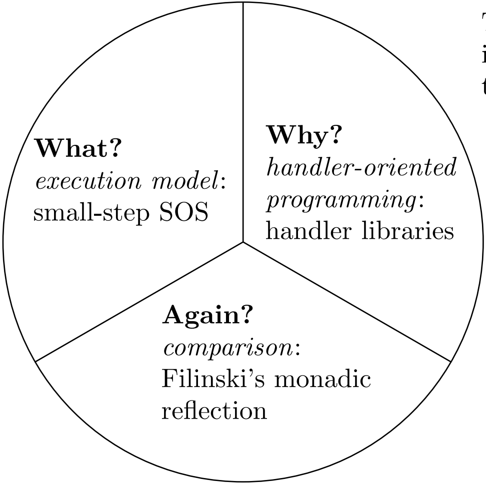
handle
  if $(\mathtt{get}\ \ell) = 0$
  then `raise` *DivideByZero*
  else $42\ /\ (\mathtt{get}\ \ell)$
with
  `raise` *DivideByZero* $k \mapsto 0$
  `raise` e $k \mapsto$ `raise` e
  `get` l $k \mapsto k\ (1)$
  return $x \mapsto$ `display` x

Addressed questions



Contribution

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

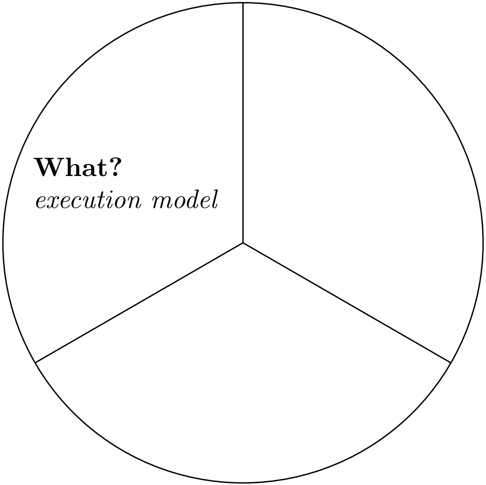


Two implementation techniques:

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

Goal

Facilitate operational discussion.


What?
execution model

Algebraic effects

Operations, parameter types, arities

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

For example:

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

Usage

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

For example:

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

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

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

For example:

$$\begin{aligned} &\texttt{raise}\ \mathit{DivideByZero} && k \mapsto 0 \\ &\texttt{raise}\ e && k \mapsto \texttt{raise}\ e \\ &\texttt{lookup}\ l && k \mapsto k\ (1) \\ &\textbf{return}\ x \mapsto \texttt{display}\ x \end{aligned}$$

λ_{eff}-calculus

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

λ_{eff}-calculus

Reduction rules

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

λ_{eff}-calculus

Reduction rules

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

More generally:

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

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

Reduction rules (cntd.)

handle
  lookup ℓ $(\lambda i \rightarrow$
    if $i = 0$
    then M_1
    else M_2
)
with
  …
  lookup l $k \mapsto k\ (1)$

$\xrightarrow{\text{op}}$

$(\lambda i \rightarrow$
  handle
    if $i = 0$
    then M_1
    else M_2
  with H
$)\ (1)$

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

handle op V $(\lambda x \rightarrow M)$
with
  …
  op p $k \mapsto N$
  …

$\xrightarrow{\text{op}} N[V/p, (\lambda x \rightarrow \textbf{handle } M \textbf{ with } H)/k]$

Reduction rules (cntd.)

$(\lambda i \rightarrow$
handle
if $i = 0$
then M_1
else M_2
with H
$)\ (1)$

$\xrightarrow{\beta}{}^{*}$

handle M_2 **with** H

Reduction rules (cntd.)

$$\textbf{handle} \quad 42 \,/\, (\texttt{get}\ \ell) \quad \textbf{with}\ H \xrightarrow{\text{hoist, op, }\beta\text{, arithmetic}}{}^{*} \textbf{handle}\ 42\ \textbf{with}\ H$$

Reduction rules (cntd.)

$$\begin{array}{l}\textbf{handle } 42 \textbf{ with} \\ \quad \ldots \\ \quad \textbf{return } x \mapsto \texttt{display } x\end{array} \xrightarrow{\text{handler return}} \texttt{display } 42$$

More generally:

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

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)

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

- Handlers

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

λ_{eff}-calculus

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.)

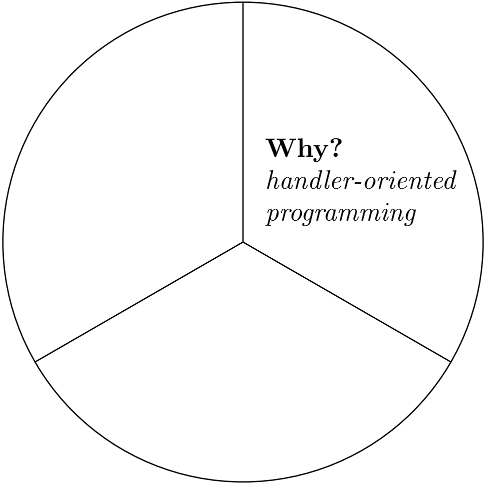
- Handler type judgements $\Gamma \vdash H : R$:

$$\frac{\begin{array}{c}\Gamma, p : Pa, k : U_E(Ar \to C) \vdash_E M : C \\ \dots \\ \Gamma, x : A \vdash_E N : C\end{array}}{\begin{array}{l}\Gamma \vdash \text{op}\ p\ k \mapsto M \\ \quad \dots \\ \quad \textbf{return}\ x \mapsto N : A \overset{\{\text{op}:Pa \to Ar, \dots\}}{\Rightarrow}{}^{E}\ C\end{array}}$$

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

$$\begin{aligned}
&m = \textbf{do}\\
&\quad \mathit{fruit} \leftarrow \mathit{chooseFruit}\\
&\quad \mathit{form} \leftarrow \mathit{chooseForm}\\
&\quad \textbf{return}\ \$\ \mathit{form} +\!\!+\ \mathit{fruit}
\end{aligned}$$

Individually:

$$\begin{aligned}
&\mathit{bothFruit} :: [\mathit{String}]\\
&\mathit{bothFruit} = [\texttt{"apple"}, \texttt{"orange"}]\\
&\mathit{randomForm} :: \mathit{IO}\ \mathit{String}\\
&\mathit{randomForm} = \textbf{do}\\
&\qquad\qquad x \leftarrow \mathit{getStdRandom}\ \mathit{random}\\
&\qquad\qquad \textbf{if}\ x\ \textbf{then return}\ \texttt{"raw "}\\
&\qquad\qquad\quad \textbf{else return}\ \texttt{"cooked "}
\end{aligned}$$

User-defined effects

In Haskell

$$
\begin{aligned}
&m = \textbf{do} \\
&\quad \mathit{fruit} \leftarrow \mathit{chooseFruit} \\
&\quad \mathit{form} \leftarrow \mathit{chooseForm} \\
&\quad \textbf{return}\ \$\ \mathit{form} +\!\!+ \mathit{fruit}
\end{aligned}
$$

Combined:

$$
\begin{aligned}
&\mathit{result} :: \mathit{IO}\ [\mathit{String}] \\
&\mathit{result} = \mathit{runListT}\ \ m \\
&\mathit{chooseFruit}\ = \mathit{ListT}\ \$\ \textbf{return}\ \mathit{bothFruit} \\
&\mathit{chooseForm} = \mathit{lift}\qquad \$\ \mathit{randomForm}
\end{aligned}
$$

Horizontal composition

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

Handler-oriented programming

Vertical composition

$$
\begin{array}{l}
\mathit{handleListProbV} :: \mathit{IO}\ [\mathit{String}] \\
\mathit{handleListProbV} = \\
\quad \textbf{handle} \\
\quad\quad (\textbf{handle do} \\
\quad\quad\quad \mathit{fruit} \leftarrow \mathit{chooseFruit} \\
\quad\quad\quad \mathit{form} \leftarrow \mathit{chooseForm} \\
\quad\quad\quad \textbf{return}\ \$\ \mathit{form} +\!\!+\ \mathit{fruit}) \\
\quad\quad\quad (\mathit{ChooseFruit} \mapsto \\
\quad\quad\quad\quad (\lambda p\ k \rightarrow \textbf{do}\ \mathit{xs} \leftarrow k\ \texttt{"apple"} \\
\quad\quad\quad\quad\quad\quad\quad\quad \mathit{ys} \leftarrow k\ \texttt{"orange"} \\
\quad\quad\quad\quad\quad\quad\quad\quad \textbf{return}\ (\mathit{xs} +\!\!+\ \mathit{ys})) \triangleleft \mathit{ChooseForm} \multimap\!\!\triangleleft\ \mathit{Empty}, \\
\quad\quad\quad \lambda x \rightarrow \textbf{return}\ [x])) \\
\quad\quad (\mathit{ChooseForm} \mapsto (\lambda p\ k \rightarrow \textbf{do}\ \{ v \leftarrow \mathit{randomForm}; k\ v \}) \\
\quad\quad\quad \triangleleft\ \mathit{Empty}, \textbf{return})
\end{array}
$$

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*

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

Our contribution

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

Implementation: free monads

Concretely

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

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

Consequently:

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

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

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:

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

- More expressive types (effect polymorphism) $\implies$ 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.)

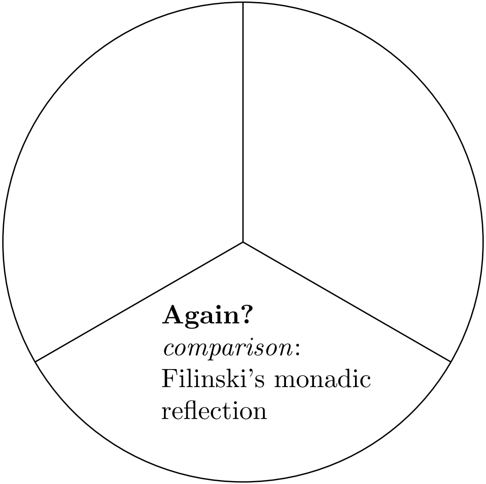
The **handle** construct **handle** M **with** H

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

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

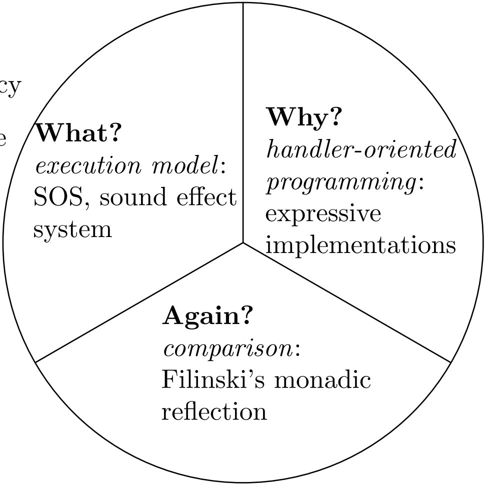
- **shift0** captures the hoisting context, concatenating it with M
- use the effect binding from top of the stack to execute op

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