# An Abstract Machine Semantics for Handlers

**Daniel Hillerström** Sam Lindley

Laboratory for Foundations of Computer Science
The University of Edinburgh

March 22, 2017

Scottish Programming Language Seminar, St. Andrews

# The Links programming language

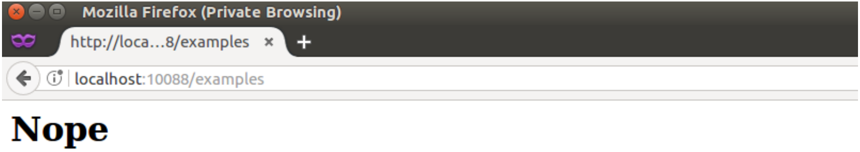
A bit of background

- Originally developed by Cooper, Lindley, Wadler, and Yallop (2006).
- Single source functional language for multi-tier web programming.
- Like JavaScript, but with ML semantics...

# The Links programming language

A bit of background

- Originally developed by Cooper, Lindley, Wadler, and Yallop (2006).
- Single source functional language for multi-tier web programming.
- Like JavaScript, but with ML semantics. . .




. . . however with worse error messages.

# The Links programming language

A bit of background

- Originally developed by Cooper, Lindley, Wadler, and Yallop (2006).
- Single source functional language for multi-tier web programming.
- Like JavaScript, but with ML semantics...

Handlers in Links

- Retrofitted with algebraic effects and handlers (Hillerström 2015).
- Server-side handlers run on top of a CEK machine.
- Client-side handlers are CPS translated.

# The Links programming language

A bit of background

- Originally developed by Cooper, Lindley, Wadler, and Yallop (2006).
- Single source functional language for multi-tier web programming.
- Like JavaScript, but with ML semantics...

Handlers in Links

- Retrofitted with algebraic effects and handlers (Hillerström 2015).
- Server-side handlers run on top of a CEK machine.
- Client-side handlers are CPS translated.

# A calculus of handlers

Fine-grain call-by-value lambda calculus

$$
\begin{array}{lrcl}
\text{Values} & V, W & ::= & x \mid \lambda x.\, M \\[1em]
\text{Computations} & M, N & ::= & V\, W \\
& & \mid & \textbf{return}\ V \\
& & \mid & \textbf{let}\ x \leftarrow M\ \textbf{in}\ N \\
& & \mid & \textbf{do}\ \ell\ V \\
& & \mid & \textbf{handle}\ M\ \textbf{with}\ H \\[1em]
\text{Handlers} & H & ::= & \{\textbf{return}\ x \mapsto M\} \\
& & \mid & \{\ell\ x\ k \mapsto M\} \uplus H
\end{array}
$$

Small-step semantics for handlers

$$
\begin{array}{rcl}
\textbf{handle}\ (\textbf{return}\ V)\ \textbf{with}\ H & \leadsto & N[V/x], \quad \text{if } \{\textbf{return}\ x \mapsto N\} \in H \\
\textbf{handle}\ \mathcal{E}[\textbf{do}\ \ell\ V]\ \textbf{with}\ H & \leadsto & N[V/x, \lambda y.\, \textbf{handle}\ \mathcal{E}[\textbf{return}\ y]\ \textbf{with}\ H/k] \\
& & \text{if } \{\ell\ x\ k \mapsto N\} \in H
\end{array}
$$

# Drunk coin tossing

```
let c1 ← do Choose () in
if c1 then
  let c2 ← do Choose () in
  if c2 then
     return Heads ()
   else
     return Tails ()
else
  do Fail ()
```

# Drunk coin tossing

```
handle
  let c1 ← do Choose () in
  if c1 then
    let c2 ← do Choose () in
    if c2 then
       return Heads ()
     else
       return Tails ()
  else
    do Fail ()
with
{ return x  ↦ return Some x
  Fail () k ↦ return None   }
```

# Drunk coin tossing

```
handle
  handle
    let c1 ← do Choose () in
    if c1 then
      let c2 ← do Choose () in
      if c2 then
         return Heads ()
       else
         return Tails ()
    else
      do Fail ()
  with
  { return x   ↦ return Some x
    Fail () k ↦ return None    }
with
{ return x     ↦ return x
  Choose () k ↦ k true }
```

# Drunk coin tossing

```
handle
  handle
    let c1 ← do Choose () in
    if c1 then
      let c2 ← do Choose () in
      if c2 then
         return Heads ()
       else
         return Tails ()
    else
      do Fail ()
  with
  { return x  ↦ return Some x
    Fail () k ↦ return None   }
with
{ return x    ↦ return x
  Choose () k ↦ k true }
```

evaluates to Some Heads

# Drunk coin tossing

```
handle
  handle
    let c1 ← do Choose () in
    if c1 then
      let c2 ← do Choose () in
      if c2 then
         return Heads ()
       else
         return Tails ()
    else
      do Fail ()
  with
  { return x  ↦ return Some x
    Fail () k ↦ return None   }
with
{ return x    ↦ return [x]
  Choose () k ↦ k true ++ k false }
```

evaluates to `[Some Heads, Some Tails, None]`

# CEK 101 (Felleisen and Friedman 1987)

A CEK machine operates on configurations of the shape $\langle C \mid E \mid K \rangle$, where:

- Control $C$ is the expression being evaluated ($M$)
- Environment $E$ binds the free variables ($\gamma$)
- Continuation $K$ instructs the machine what to do next ($\kappa$)

# CEK 101: Syntax and semantics

Abstract machine syntax

$$
\begin{array}{lrcl}
\text{Configurations} & \mathcal{C} & ::= & \langle M \mid \gamma \mid \kappa \rangle \\
\text{Value environments} & \gamma & ::= & \emptyset \mid \gamma[x \mapsto v] \\
\text{Values} & v, w & ::= & (\gamma, \lambda x.\, M) \mid \kappa \\
\text{Continuations} & \kappa & ::= & [\,] \mid \phi :: \kappa \\
\text{Continuation frames} & \phi & ::= & (\gamma, x, N)
\end{array}
$$

Abstract machine semantics

$$
\begin{array}{rcll}
M & \longrightarrow & \langle M \mid \emptyset \mid \kappa_0 \rangle & \\
\langle V\ W \mid \gamma \mid \kappa \rangle & \longrightarrow & \langle M \mid \gamma'[x \mapsto [\![W]\!]\gamma] \mid \kappa \rangle, & \text{if } [\![V]\!]\gamma = (\gamma', \lambda x.\, M) \\
\langle \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N \mid \gamma \mid \kappa \rangle & \longrightarrow & \langle M \mid \gamma \mid (\gamma, x, N) :: \kappa \rangle &
\end{array}
$$

# CEK 101: Syntax and semantics

Abstract machine syntax

$$
\begin{array}{lrcl}
\text{Configurations} & \mathcal{C} & ::= & \langle M \mid \gamma \mid \kappa \rangle \\
\text{Value environments} & \gamma & ::= & \emptyset \mid \gamma[x \mapsto v] \\
\text{Values} & v, w & ::= & (\gamma, \lambda x.\, M) \mid \kappa \\
\text{Continuations} & \kappa & ::= & [\,] \mid \phi :: \kappa \\
\text{Continuation frames} & \phi & ::= & (\gamma, x, N)
\end{array}
$$

Abstract machine semantics

$$
\begin{array}{rcll}
M & \longrightarrow & \langle M \mid \emptyset \mid \kappa_0 \rangle & \\
\langle V\ W \mid \gamma \mid \kappa \rangle & \longrightarrow & \langle M \mid \gamma'[x \mapsto [\![W]\!]\gamma] \mid \kappa \rangle, & \text{if } [\![V]\!]\gamma = (\gamma', \lambda x.\, M) \\
\langle \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N \mid \gamma \mid \kappa \rangle & \longrightarrow & \langle M \mid \gamma \mid (\gamma, x, N) :: \kappa \rangle &
\end{array}
$$

# CEK 101: Syntax and semantics

Abstract machine syntax

$$
\begin{array}{lrl}
\text{Configurations} & \mathcal{C} & ::= \langle M \mid \gamma \mid \kappa \rangle \\
\text{Value environments} & \gamma & ::= \emptyset \mid \gamma[x \mapsto v] \\
\text{Values} & v, w & ::= (\gamma, \lambda x.\, M) \mid \kappa \\
\text{Continuations} & \kappa & ::= [\,] \mid \phi :: \kappa \\
\text{Continuation frames} & \phi & ::= (\gamma, x, N)
\end{array}
$$

Abstract machine semantics

$$
\begin{array}{rcll}
M & \longrightarrow & \langle M \mid \emptyset \mid \kappa_0 \rangle & \\
\langle V\ W \mid \gamma \mid \kappa \rangle & \longrightarrow & \langle M \mid \gamma'[x \mapsto [\![W]\!]\gamma] \mid \kappa \rangle, & \text{if } [\![V]\!]\gamma = (\gamma', \lambda x.\, M) \\
\langle \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N \mid \gamma \mid \kappa \rangle & \longrightarrow & \langle M \mid \gamma \mid (\gamma, x, N) :: \kappa \rangle &
\end{array}
$$

# CEK 101: Syntax and semantics

Abstract machine syntax

$$
\begin{array}{lrl}
\text{Configurations} & \mathcal{C} & ::= \langle M \mid \gamma \mid \kappa \rangle \\
\text{Value environments} & \gamma & ::= \emptyset \mid \gamma[x \mapsto v] \\
\text{Values} & v, w & ::= (\gamma, \lambda x.\, M) \mid \kappa \\
\text{Continuations} & \kappa & ::= [\,] \mid \phi :: \kappa \\
\text{Continuation frames} & \phi & ::= (\gamma, x, N)
\end{array}
$$

Abstract machine semantics

$$
\begin{array}{rcll}
M & \longrightarrow & \langle M \mid \emptyset \mid \kappa_0 \rangle & \\
\langle V\; W \mid \gamma \mid \kappa \rangle & \longrightarrow & \langle M \mid \gamma'[x \mapsto [\![W]\!]\gamma] \mid \kappa \rangle, & \text{if } [\![V]\!]\gamma = (\gamma', \lambda x.\, M) \\
\langle \textbf{let}\; x \leftarrow M \;\textbf{in}\; N \mid \gamma \mid \kappa \rangle & \longrightarrow & \langle M \mid \gamma \mid (\gamma, x, N) :: \kappa \rangle &
\end{array}
$$

# A CEK machine with handlers

The trick: augment the configuration space and enrich the continuation structure

$$
\begin{array}{lrcl}
\text{Configurations} & \mathcal{C} & ::= & \langle M \mid \gamma \mid \sigma \rangle \\
\\
\text{Value environments} & \gamma & ::= & \emptyset \mid \gamma[x \mapsto v] \\
\text{Values} & v, w & ::= & (\gamma, \lambda x.\, M) \mid \kappa \\
\\
\text{Continuations} & \kappa & ::= & [\,] \mid \phi :: \kappa \\
\text{Continuation frames} & \phi & ::= & (\gamma, x, N)
\end{array}
$$

# A CEK machine with handlers

The trick: augment the configuration space and enrich the continuation structure

| | | |
|---|---|---|
| Configurations | $\mathcal{C}$ | $::= \langle M \mid \gamma \mid \kappa \rangle$ |
| | | $\mid\ \langle M \mid \gamma \mid \kappa \mid \kappa' \rangle_{\mathsf{op}}$ |
| Value environments | $\gamma$ | $::= \emptyset \mid \gamma[x \mapsto v]$ |
| Values | $v, w$ | $::= (\gamma, \lambda x.\, M) \mid \kappa$ |
| Continuations | $\kappa$ | $::= [\,] \mid \delta :: \kappa$ |
| Continuation frames | $\delta$ | $::= (\sigma, \chi)$ |
| Pure continuations | $\sigma$ | $::= [\,] \mid \phi :: \sigma$ |
| Pure continuation frames | $\phi$ | $::= (\gamma, x, N)$ |
| Handler closures | $\chi$ | $::= (\gamma, H)$ |

Intuition: $\kappa'$ is a list of handlers which forwarded some operation

# Drunken coin tossing in an abstract machine

Machine transitions

$$M \longrightarrow \langle M \mid \emptyset \mid \kappa_0 \rangle$$

The example program

```
M :=
 handle
  handle
   let c1 ⟵ do Choose () in
   if c1 then
    let c2 ⟵ do Choose () in
    if c2 then
      return Heads ()
    else
      return Tails ()
   else
     do Fail ()
  with (H_fail)
  { return x   ⟼ return Some x
    Fail () k ⟼ return None   }
 with (H_true)
 { return x    ⟼ return x
   Choose () k ⟼ k true }
```

# Drunken coin tossing in an abstract machine

Machine transitions

$$\begin{array}{rl} M & \longrightarrow \quad \langle M \mid \emptyset \mid \kappa_0 \rangle \\ & \longrightarrow^+ \quad \langle \textbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \end{array}$$

The example program

```
M :=
 handle
  handle
   let c1 ← do Choose () in
   if c1 then
    let c2 ← do Choose () in
    if c2 then
      return Heads ()
    else
      return Tails ()
   else
     do Fail ()
  with (H_fail)
  { return x   ↦ return Some x
    Fail () k ↦ return None   }
 with (H_true)
 { return x    ↦ return x
   Choose () k ↦ k true }
```

# Drunken coin tossing in an abstract machine

Machine transitions

$$
\begin{aligned}
M &\longrightarrow \langle M \mid \emptyset \mid \kappa_0 \rangle \\
&\longrightarrow^+ \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
&\longrightarrow \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] \rangle_{\mathsf{op}}
\end{aligned}
$$

The example program

```
M :=
 handle
  handle
   let c1 ⟵ do Choose () in
   if c1 then
    let c2 ⟵ do Choose () in
    if c2 then
      return Heads ()
    else
      return Tails ()
   else
     do Fail ()
  with (H_fail)
  { return x   ↦ return Some x
    Fail () k ↦ return None   }
 with (H_true)
 { return x    ↦ return x
   Choose () k ↦ k true }
```

# Drunken coin tossing in an abstract machine

Machine transitions

$$
\begin{array}{lll}
M & \longrightarrow & \langle M \mid \emptyset \mid \kappa_0 \rangle \\
 & \longrightarrow^{+} & \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
 & \longrightarrow & \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] \rangle_{\mathbf{op}} \\
 & \longrightarrow & \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] +\!\!+ [(\sigma_2, H_{\mathsf{fail}})] \rangle_{\mathbf{op}}
\end{array}
$$

The example program

```
M :=
 handle
  handle
   let c1 ← do Choose () in
   if c1 then
    let c2 ← do Choose () in
    if c2 then
      return Heads ()
    else
      return Tails ()
   else
     do Fail ()
  with (H_fail)
  { return x   ↦ return Some x
    Fail () k ↦ return None   }
 with (H_true)
 { return x    ↦ return x
   Choose () k ↦ k true }
```

# Drunken coin tossing in an abstract machine

Machine transitions

$$
\begin{aligned}
M &\longrightarrow \langle M \mid \emptyset \mid \kappa_0 \rangle \\
&\longrightarrow^{+} \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
&\longrightarrow \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] \rangle_{\mathbf{op}} \\
&\longrightarrow \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] +\!\!+ [(\sigma_2, H_{\mathsf{fail}})] \rangle_{\mathbf{op}} \\
&\longrightarrow \langle k\ \mathsf{true} \mid \emptyset[k \mapsto [(\sigma_2, H_{\mathsf{fail}})] +\!\!+ [(\sigma_1, H_{\mathsf{true}})] \mid \kappa_0 \rangle \\
&\qquad \text{where } H_{\mathsf{true}}(\mathsf{Choose}) = \{\mathsf{Choose}\ ()\ k \mapsto k\ \mathsf{true}\}
\end{aligned}
$$

The example program

```
M :=
 handle
  handle
   let c1 ⟵ do Choose () in
   if c1 then
    let c2 ⟵ do Choose () in
    if c2 then
      return Heads ()
    else
      return Tails ()
   else
     do Fail ()
  with (H_fail)
  { return x    ↦ return Some x
    Fail () k ↦ return None    }
 with (H_true)
 { return x     ↦ return x
   Choose () k ↦ k true }
```

# Drunken coin tossing in an abstract machine

Machine transitions

$$
\begin{array}{rl}
M & \longrightarrow \quad \langle M \mid \emptyset \mid \kappa_0 \rangle \\
& \longrightarrow^+ \quad \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
& \longrightarrow \quad \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] \rangle_{\mathsf{op}} \\
& \longrightarrow \quad \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] \mathbin{+\!\!+} [(\sigma_2, H_{\mathsf{fail}})] \rangle_{\mathsf{op}} \\
& \longrightarrow \quad \langle k\ \mathsf{true} \mid \emptyset[k \mapsto [(\sigma_2, H_{\mathsf{fail}})] \mathbin{+\!\!+} [(\sigma_1, H_{\mathsf{true}})] \mid \kappa_0 \rangle \\
& \qquad \text{where } H_{\mathsf{true}}(\mathsf{Choose}) = \{\mathsf{Choose}\ ()\ k \mapsto k\ \mathsf{true}\} \\
& \longrightarrow^+ \quad \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset[c1 \mapsto \mathsf{true}] \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle
\end{array}
$$

The example program

```
M :=
 handle
  handle
   let c1 ← do Choose () in
   if c1 then
    let c2 ← do Choose () in
    if c2 then
      return Heads ()
    else
      return Tails ()
   else
     do Fail ()
  with (H_fail)
  { return x   ↦ return Some x
    Fail () k ↦ return None   }
 with (H_true)
 { return x     ↦ return x
   Choose () k ↦ k true }
```

# Drunken coin tossing in an abstract machine

Machine transitions

$$
\begin{array}{rl}
M & \longrightarrow \langle M \mid \emptyset \mid \kappa_0 \rangle \\
& \longrightarrow^+ \langle \textbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
& \longrightarrow \langle \textbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] \rangle_{\mathbf{op}} \\
& \longrightarrow \langle \textbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] +\!\!+ [(\sigma_2, H_{\mathsf{fail}})] \rangle_{\mathbf{op}} \\
& \longrightarrow \langle k\ \mathsf{true} \mid \emptyset[k \mapsto [(\sigma_2, H_{\mathsf{fail}})] +\!\!+ [(\sigma_1, H_{\mathsf{true}})] \mid \kappa_0 \rangle \\
& \qquad \text{where } H_{\mathsf{true}}(\mathsf{Choose}) = \{\mathsf{Choose}\ ()\ k \mapsto k\ \mathsf{true}\} \\
& \longrightarrow^+ \langle \textbf{do}\ \mathsf{Choose}\ () \mid \emptyset[c1 \mapsto \mathsf{true}] \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
& \longrightarrow^+ \langle \textbf{return}\ \mathsf{Heads} \mid \gamma \mid ([\,], H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle
\end{array}
$$

The example program

```
M :=
 handle
  handle
   let c1 ← do Choose () in
   if c1 then
    let c2 ← do Choose () in
    if c2 then
      return Heads ()
    else
      return Tails ()
   else
     do Fail ()
  with (H_fail)
  { return x   ↦ return Some x
    Fail () k ↦ return None   }
 with (H_true)
 { return x    ↦ return x
   Choose () k ↦ k true }
```

# Drunken coin tossing in an abstract machine

Machine transitions

$$
\begin{array}{lll}
M & \longrightarrow & \langle M \mid \emptyset \mid \kappa_0 \rangle \\
 & \longrightarrow^{+} & \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
 & \longrightarrow & \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] \rangle_{\mathbf{op}} \\
 & \longrightarrow & \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] \mathbin{+\!\!+} [(\sigma_2, H_{\mathsf{fail}})] \rangle_{\mathbf{op}} \\
 & \longrightarrow & \langle k\ \mathsf{true} \mid \emptyset[k \mapsto [(\sigma_2, H_{\mathsf{fail}})] \mathbin{+\!\!+} [(\sigma_1, H_{\mathsf{true}})] \mid \kappa_0 \rangle \\
 & & \quad \text{where } H_{\mathsf{true}}(\mathsf{Choose}) = \{\mathsf{Choose}\ ()\ k \mapsto k\ \mathsf{true}\} \\
 & \longrightarrow^{+} & \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset[c1 \mapsto \mathsf{true}] \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
 & \longrightarrow^{+} & \langle \mathbf{return}\ \mathsf{Heads} \mid \gamma \mid ([\,], H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
 & \longrightarrow & \langle \mathbf{return}\ \mathsf{Some}\ x \mid \emptyset[x \mapsto \mathsf{Heads}] \mid ([\,], H_{\mathsf{true}}) :: \kappa_0 \rangle \\
 & & \quad \text{where } H_{\mathsf{fail}}(\mathbf{return}) = \{\mathbf{return}\ x \mapsto \mathbf{return}\ \mathsf{Some}\ x\}
\end{array}
$$

The example program

```
M :=
 handle
  handle
   let c1 ← do Choose () in
   if c1 then
    let c2 ← do Choose () in
    if c2 then
      return Heads ()
    else
      return Tails ()
   else
     do Fail ()
  with (H_fail)
  { return x   ↦ return Some x
     Fail () k ↦ return None   }
 with (H_true)
 { return x    ↦ return x
   Choose () k ↦ k true }
```

# Drunken coin tossing in an abstract machine

Machine transitions

$$
\begin{array}{rl}
M & \longrightarrow \ \langle M \mid \emptyset \mid \kappa_0 \rangle \\
& \longrightarrow^+ \ \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
& \longrightarrow \ \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] \rangle_{\mathbf{op}} \\
& \longrightarrow \ \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] +\!\!+ [(\sigma_2, H_{\mathsf{fail}})] \rangle_{\mathbf{op}} \\
& \longrightarrow \ \langle k\ \mathsf{true} \mid \emptyset[k \mapsto [(\sigma_2, H_{\mathsf{fail}})] +\!\!+ [(\sigma_1, H_{\mathsf{true}})] \mid \kappa_0 \rangle \\
& \qquad \text{where } H_{\mathsf{true}}(\mathsf{Choose}) = \{\mathsf{Choose}\ ()\ k \mapsto k\ \mathsf{true}\} \\
& \longrightarrow^+ \ \langle \mathbf{do}\ \mathsf{Choose}\ () \mid \emptyset[c1 \mapsto \mathsf{true}] \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
& \longrightarrow^+ \ \langle \mathbf{return}\ \mathsf{Heads} \mid \gamma \mid ([\,], H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
& \longrightarrow \ \langle \mathbf{return}\ \mathsf{Some}\ x \mid \emptyset[x \mapsto \mathsf{Heads}] \mid ([\,], H_{\mathsf{true}}) :: \kappa_0 \rangle \\
& \qquad \text{where } H_{\mathsf{fail}}(\mathbf{return}) = \{\mathbf{return}\ x \mapsto \mathbf{return}\ \mathsf{Some}\ x\} \\
& \longrightarrow \ \langle \mathbf{return}\ x \mid \emptyset[x \mapsto \mathsf{Some}\ x] \mid \kappa_0 \rangle \\
& \qquad \text{where } H_{\mathsf{true}}(\mathbf{return}) = \{\mathbf{return}\ x \mapsto \mathbf{return}\ x\}
\end{array}
$$

The example program

```
M :=
 handle
  handle
   let c1 ← do Choose () in
   if c1 then
    let c2 ← do Choose () in
    if c2 then
      return Heads ()
    else
      return Tails ()
   else
     do Fail ()
  with (H_fail)
  { return x   ↦ return Some x
     Fail () k ↦ return None   }
 with (H_true)
 { return x    ↦ return x
   Choose () k ↦ k true }
```

# Drunken coin tossing in an abstract machine

Machine transitions

$$
\begin{aligned}
M \;&\longrightarrow\; \langle M \mid \emptyset \mid \kappa_0 \rangle \\
&\longrightarrow^+\; \langle \textbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
&\longrightarrow\; \langle \textbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] \rangle_{\mathbf{op}} \\
&\longrightarrow\; \langle \textbf{do}\ \mathsf{Choose}\ () \mid \emptyset \mid (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \mid [\,] +\!\!+ [(\sigma_2, H_{\mathsf{fail}})] \rangle_{\mathbf{op}} \\
&\longrightarrow\; \langle k\ \mathsf{true} \mid \emptyset[k \mapsto [(\sigma_2, H_{\mathsf{fail}})] +\!\!+ [(\sigma_1, H_{\mathsf{true}})]] \mid \kappa_0 \rangle \\
&\qquad \text{where } H_{\mathsf{true}}(\mathsf{Choose}) = \{\mathsf{Choose}\ ()\ k \mapsto k\ \mathsf{true}\} \\
&\longrightarrow^+\; \langle \textbf{do}\ \mathsf{Choose}\ () \mid \emptyset[c1 \mapsto \mathsf{true}] \mid (\sigma_2, H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
&\longrightarrow^+\; \langle \textbf{return}\ \mathsf{Heads} \mid \gamma \mid ([\,], H_{\mathsf{fail}}) :: (\sigma_1, H_{\mathsf{true}}) :: \kappa_0 \rangle \\
&\longrightarrow\; \langle \textbf{return}\ \mathsf{Some}\ x \mid \emptyset[x \mapsto \mathsf{Heads}] \mid ([\,], H_{\mathsf{true}}) :: \kappa_0 \rangle \\
&\qquad \text{where } H_{\mathsf{fail}}(\textbf{return}) = \{\textbf{return}\ x \mapsto \textbf{return}\ \mathsf{Some}\ x\} \\
&\longrightarrow\; \langle \textbf{return}\ x \mid \emptyset[x \mapsto \mathsf{Some}\ x] \mid \kappa_0 \rangle \\
&\qquad \text{where } H_{\mathsf{true}}(\textbf{return}) = \{\textbf{return}\ x \mapsto \textbf{return}\ x\} \\
&\longrightarrow^+\; \langle \textbf{return}\ \mathsf{Some}\ \mathrm{x} \mid \emptyset[x \mapsto \mathsf{Heads}] \mid [\,] \rangle
\end{aligned}
$$

The example program

```
M :=
 handle
  handle
   let c1 ← do Choose () in
   if c1 then
    let c2 ← do Choose () in
    if c2 then
      return Heads ()
    else
      return Tails ()
   else
     do Fail ()
  with (H_fail)
  { return x   ↦ return Some x
    Fail () k ↦ return None   }
 with (H_true)
 { return x     ↦ return x
   Choose () k ↦ k true }
```

# A semantics for CEK with handlers

Installing a handler and running a computation to completion

$$
\begin{aligned}
\langle \textbf{handle}\ M\ \textbf{with}\ H \mid \gamma \mid \kappa \rangle &\longrightarrow \langle M \mid \gamma \mid ([\,], (\gamma, H)) :: \kappa \rangle \\
\langle \textbf{return}\ V \mid \gamma \mid ([\,], (\gamma', H)) :: \kappa \rangle &\longrightarrow \langle M \mid \gamma'[x \mapsto [\![V]\!]\gamma] \mid \kappa \rangle, \\
&\qquad \text{if } H(\textbf{return}) = \{\textbf{return}\ x \mapsto M\}
\end{aligned}
$$

Operation invocation, forwarding, handling, and stack reconstruction

$$
\begin{aligned}
\langle \textbf{do}\ \ell\ V \mid \gamma \mid \kappa \rangle &\longrightarrow \langle \textbf{do}\ \ell\ V \mid \gamma \mid \kappa \mid [\,] \rangle_{\textsf{op}} \\
\langle \textbf{do}\ \ell\ V \mid \gamma \mid (\sigma, (\gamma', H)) :: \kappa \mid \kappa' \rangle_{\textsf{op}} &\longrightarrow \langle \textbf{do}\ \ell\ V \mid \gamma \mid \kappa \mid \kappa' +\!\!+ [(\sigma, (\gamma', H))] \rangle_{\textsf{op}}, \\
&\qquad \text{if } H(\ell) = \emptyset \\
\langle \textbf{do}\ \ell\ V \mid \gamma \mid (\sigma, (\gamma', H)) :: \kappa \mid \kappa' \rangle_{\textsf{op}} &\longrightarrow \langle M \mid \gamma'[x \mapsto [\![V]\!]\gamma, k \mapsto \kappa' +\!\!+ [(\sigma, (\gamma', H))]] \mid \kappa \rangle, \\
&\qquad \text{if } H(\ell) = \{\ell\ x\ k \mapsto M\} \\
\langle V\ W \mid \gamma \mid \kappa \rangle &\longrightarrow \langle \textbf{return}\ W \mid \gamma \mid \kappa' +\!\!+ \kappa \rangle, \ \text{if } [\![V]\!]\gamma = \kappa'
\end{aligned}
$$

# A semantics for CEK with handlers

Installing a handler and running a computation to completion

$$\begin{aligned}
\langle \textbf{handle } M \textbf{ with } H \mid \gamma \mid \kappa \rangle &\longrightarrow \langle M \mid \gamma \mid ([\,], (\gamma, H)) :: \kappa \rangle \\
\langle \textbf{return } V \mid \gamma \mid ([\,], (\gamma', H)) :: \kappa \rangle &\longrightarrow \langle M \mid \gamma'[x \mapsto [\![V]\!]\gamma] \mid \kappa \rangle, \\
&\qquad \text{if } H(\textbf{return}) = \{\textbf{return } x \mapsto M\}
\end{aligned}$$

Operation invocation, forwarding, handling, and stack reconstruction

$$\begin{aligned}
\langle \textbf{do } \ell \; V \mid \gamma \mid \kappa \rangle &\longrightarrow \langle \textbf{do } \ell \; V \mid \gamma \mid \kappa \mid [\,] \rangle_{\mathsf{op}} \\
\langle \textbf{do } \ell \; V \mid \gamma \mid (\sigma, (\gamma', H)) :: \kappa \mid \kappa' \rangle_{\mathsf{op}} &\longrightarrow \langle \textbf{do } \ell \; V \mid \gamma \mid \kappa \mid \kappa' +\!\!+ [(\sigma, (\gamma', H))] \rangle_{\mathsf{op}}, \\
&\qquad \text{if } H(\ell) = \emptyset \\
\langle \textbf{do } \ell \; V \mid \gamma \mid (\sigma, (\gamma', H)) :: \kappa \mid \kappa' \rangle_{\mathsf{op}} &\longrightarrow \langle M \mid \gamma'[x \mapsto [\![V]\!]\gamma, k \mapsto \kappa' +\!\!+ [(\sigma, (\gamma', H))]] \mid \kappa \rangle, \\
&\qquad \text{if } H(\ell) = \{\ell \; x \; k \mapsto M\} \\
\langle V \; W \mid \gamma \mid \kappa \rangle &\longrightarrow \langle \textbf{return } W \mid \gamma \mid \kappa' +\!\!+ \kappa \rangle, \; \text{if } [\![V]\!]\gamma = \kappa'
\end{aligned}$$

# A semantics for CEK with handlers

Installing a handler and running a computation to completion

$$\begin{aligned}
\langle \textbf{handle}\ M\ \textbf{with}\ H \mid \gamma \mid \kappa \rangle &\longrightarrow \langle M \mid \gamma \mid ([\,], (\gamma, H)) :: \kappa \rangle \\
\langle \textbf{return}\ V \mid \gamma \mid ([\,], (\gamma', H)) :: \kappa \rangle &\longrightarrow \langle M \mid \gamma'[x \mapsto [\![V]\!]\gamma] \mid \kappa \rangle, \\
&\qquad \text{if } H(\textbf{return}) = \{\textbf{return}\ x \mapsto M\}
\end{aligned}$$

Operation invocation, forwarding, handling, and stack reconstruction

$$\begin{aligned}
\langle \textbf{do}\ \ell\ V \mid \gamma \mid \kappa \rangle &\longrightarrow \langle \textbf{do}\ \ell\ V \mid \gamma \mid \kappa \mid [\,] \rangle_{\mathsf{op}} \\
\langle \textbf{do}\ \ell\ V \mid \gamma \mid (\sigma, (\gamma', H)) :: \kappa \mid \kappa' \rangle_{\mathsf{op}} &\longrightarrow \langle \textbf{do}\ \ell\ V \mid \gamma \mid \kappa \mid \kappa' +\!\!+ [(\sigma, (\gamma', H))] \rangle_{\mathsf{op}}, \\
&\qquad \text{if } H(\ell) = \emptyset \\
\langle \textbf{do}\ \ell\ V \mid \gamma \mid (\sigma, (\gamma', H)) :: \kappa \mid \kappa' \rangle_{\mathsf{op}} &\longrightarrow \langle M \mid \gamma'[x \mapsto [\![V]\!]\gamma, k \mapsto \kappa' +\!\!+ [(\sigma, (\gamma', H))]] \mid \kappa \rangle, \\
&\qquad \text{if } H(\ell) = \{\ell\ x\ k \mapsto M\} \\
\langle V\ W \mid \gamma \mid \kappa \rangle &\longrightarrow \langle \textbf{return}\ W \mid \gamma \mid \kappa' +\!\!+ \kappa \rangle, \ \text{if } [\![V]\!]\gamma = \kappa'
\end{aligned}$$

# A semantics for CEK with handlers

Installing a handler and running a computation to completion

$$
\begin{aligned}
\langle \textbf{handle}\ M\ \textbf{with}\ H \mid \gamma \mid \kappa\rangle &\longrightarrow \langle M \mid \gamma \mid ([\,], (\gamma, H)) :: \kappa\rangle \\
\langle \textbf{return}\ V \mid \gamma \mid ([\,], (\gamma', H)) :: \kappa\rangle &\longrightarrow \langle M \mid \gamma'[x \mapsto [\![V]\!]\gamma] \mid \kappa\rangle, \\
&\qquad \text{if } H(\textbf{return}) = \{\textbf{return}\ x \mapsto M\}
\end{aligned}
$$

Operation invocation, forwarding, handling, and stack reconstruction

$$
\begin{aligned}
\langle \textbf{do}\ \ell\ V \mid \gamma \mid \kappa\rangle &\longrightarrow \langle \textbf{do}\ \ell\ V \mid \gamma \mid \kappa \mid [\,]\rangle_{\textsf{op}} \\
\langle \textbf{do}\ \ell\ V \mid \gamma \mid (\sigma, (\gamma', H)) :: \kappa \mid \kappa'\rangle_{\textsf{op}} &\longrightarrow \langle \textbf{do}\ \ell\ V \mid \gamma \mid \kappa \mid \kappa' \mathbin{+\!\!+} [(\sigma, (\gamma', H))]\rangle_{\textsf{op}}, \\
&\qquad \text{if } H(\ell) = \emptyset \\
\langle \textbf{do}\ \ell\ V \mid \gamma \mid (\sigma, (\gamma', H)) :: \kappa \mid \kappa'\rangle_{\textsf{op}} &\longrightarrow \langle M \mid \gamma'[x \mapsto [\![V]\!]\gamma, k \mapsto \kappa' \mathbin{+\!\!+} [(\sigma, (\gamma', H))]] \mid \kappa\rangle, \\
&\qquad \text{if } H(\ell) = \{\ell\ x\ k \mapsto M\} \\
\langle V\ W \mid \gamma \mid \kappa\rangle &\longrightarrow \langle \textbf{return}\ W \mid \gamma \mid \kappa' \mathbin{+\!\!+} \kappa\rangle,\ \text{if } [\![V]\!]\gamma = \kappa'
\end{aligned}
$$

# A semantics for CEK with handlers

Installing a handler and running a computation to completion

$$\begin{aligned}
\langle \textbf{handle } M \textbf{ with } H \mid \gamma \mid \kappa \rangle &\longrightarrow \langle M \mid \gamma \mid ([\,], (\gamma, H)) :: \kappa \rangle \\
\langle \textbf{return } V \mid \gamma \mid ([\,], (\gamma', H)) :: \kappa \rangle &\longrightarrow \langle M \mid \gamma'[x \mapsto [\![V]\!]\gamma] \mid \kappa \rangle, \\
&\quad\;\; \text{if } H(\textbf{return}) = \{\textbf{return } x \mapsto M\}
\end{aligned}$$

Operation invocation, forwarding, handling, and stack reconstruction

$$\begin{aligned}
\langle \textbf{do } \ell\ V \mid \gamma \mid \kappa \rangle &\longrightarrow \langle \textbf{do } \ell\ V \mid \gamma \mid \kappa \mid [\,] \rangle_{\textsf{op}} \\
\langle \textbf{do } \ell\ V \mid \gamma \mid (\sigma, (\gamma', H)) :: \kappa \mid \kappa' \rangle_{\textsf{op}} &\longrightarrow \langle \textbf{do } \ell\ V \mid \gamma \mid \kappa \mid \kappa' +\!\!+ [(\sigma, (\gamma', H))] \rangle_{\textsf{op}}, \\
&\quad\;\; \text{if } H(\ell) = \emptyset \\
\langle \textbf{do } \ell\ V \mid \gamma \mid (\sigma, (\gamma', H)) :: \kappa \mid \kappa' \rangle_{\textsf{op}} &\longrightarrow \langle M \mid \gamma'[x \mapsto [\![V]\!]\gamma, k \mapsto \kappa' +\!\!+ [(\sigma, (\gamma', H))]] \mid \kappa \rangle, \\
&\quad\;\; \text{if } H(\ell) = \{\ell\ x\ k \mapsto M\} \\
\langle V\ W \mid \gamma \mid \kappa \rangle &\longrightarrow \langle \textbf{return } W \mid \gamma \mid \kappa' +\!\!+ \kappa \rangle, \ \text{if } [\![V]\!]\gamma = \kappa'
\end{aligned}$$

# A semantics for CEK with handlers

Installing a handler and running a computation to completion

$$\begin{array}{rcl}\langle \textbf{handle}\ M\ \textbf{with}\ H \mid \gamma \mid \kappa\rangle & \longrightarrow & \langle M \mid \gamma \mid ([\,], (\gamma, H)) :: \kappa\rangle \\ \langle \textbf{return}\ V \mid \gamma \mid ([\,], (\gamma', H)) :: \kappa\rangle & \longrightarrow & \langle M \mid \gamma'[x \mapsto [\![V]\!]\gamma] \mid \kappa\rangle, \\ & & \text{if } H(\textbf{return}) = \{\textbf{return}\ x \mapsto M\}\end{array}$$

Operation invocation, forwarding, handling, and stack reconstruction

$$\begin{array}{rcl}\langle \textbf{do}\ \ell\ V \mid \gamma \mid \kappa\rangle & \longrightarrow & \langle \textbf{do}\ \ell\ V \mid \gamma \mid \kappa \mid [\,]\rangle_{\textsf{op}} \\ \langle \textbf{do}\ \ell\ V \mid \gamma \mid (\sigma, (\gamma', H)) :: \kappa \mid \kappa'\rangle_{\textsf{op}} & \longrightarrow & \langle \textbf{do}\ \ell\ V \mid \gamma \mid \kappa \mid \kappa' +\!\!+ [(\sigma, (\gamma', H))]\rangle_{\textsf{op}}, \\ & & \text{if } H(\ell) = \emptyset \\ \langle \textbf{do}\ \ell\ V \mid \gamma \mid (\sigma, (\gamma', H)) :: \kappa \mid \kappa'\rangle_{\textsf{op}} & \longrightarrow & \langle M \mid \gamma'[x \mapsto [\![V]\!]\gamma, k \mapsto \kappa' +\!\!+ [(\sigma, (\gamma', H))]] \mid \kappa\rangle, \\ & & \text{if } H(\ell) = \{\ell\ x\ k \mapsto M\} \\ \langle V\ W \mid \gamma \mid \kappa\rangle & \longrightarrow & \langle \textbf{return}\ W \mid \gamma \mid \kappa' +\!\!+ \kappa\rangle,\ \text{if } [\![V]\!]\gamma = \kappa'\end{array}$$

# Soundness of our CEK machine

**Theorem (Simulation)**

*If* $M \rightsquigarrow N$ *then* $M \longrightarrow^{+} N$.

See Hillerström and Lindley (2016) for the details.

# Conclusion and future work

In summary

- Augmented the configuration space of CEK
- Enriched the structure of continuations
- Showed that our machine simulates the operational semantics

Future work

- Relate the server-side abstract machine and the client-side CPS translation
- Support for multihandlers

# References I


Felleisen, Matthias and Daniel P. Friedman (1987). "Control Operators, the SECD-machine, and the $\lambda$-Calculus". In: *The Proceedings of the Conference on Formal Description of Programming Concepts III, Ebberup, Denmark*. Elsevier, pp. 193–217.

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

Hillerström, Daniel (2015). "Handlers for Algebraic Effects in Links". MA thesis. Scotland: The University of Edinburgh. URL: http://project-archive.inf.ed.ac.uk/msc/20150206/msc_proj.pdf.

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