# No value restriction is needed for algebraic effects and handlers

Ohad Kammar
<ohad.kammar@cs.ox.ac.uk>
joint work with Sean Moss and Matija Pretnar

LOLA 2016
Syntax and Semantics of Low-Level Languages
10 July 2016

# Value restriction

## Identity crisis

$$
\begin{array}{llll}
\textbf{let } id1 = (\textbf{fun } x \to x) & \textbf{in} & (* \ id1 & : \forall\alpha.\ \alpha \to \alpha *) \\
\textbf{let } id2 = id1(id1) & \textbf{in} & (* \ id2 & : \_\alpha \to \_\alpha *) \\
\qquad id2(id2) & & (* \ \texttt{TYPE ERROR: The type} & \\
& & \quad \texttt{variable } \_\alpha \texttt{ occurs} & \\
& & \quad \texttt{inside } \_\alpha \to \_\alpha *) &
\end{array}
$$

## Reason

Unrestricted, would type

$$
\begin{array}{lll}
\textbf{let } r = \textbf{ref } [\,] \textbf{ in} & (* \ r : \forall\alpha.\alpha \text{ list ref} & *) \\
\quad r := [true]; & (* \ \texttt{specialise } \alpha := \text{bool} & *) \\
\quad 0 :: !r & (* \ \texttt{specialise } \alpha := \text{int} & *)
\end{array}
$$

as int list

# Some history

## Three crucial ingredients

- Computational effects
- Polymorphism
- Call-by-value

# Some history

## Three crucial ingredients

- Computational effects
- Polymorphism
- Call-by-value

Moggi ['89] $\lambda_c$-calculus

# Some history

## Three crucial ingredients

- Computational effects
- Polymorphism
- Call-by-value

Hindley ['69]-Milner ['78]-Damas ['85]

# Some history

## Three crucial ingredients

- Computational effects
- Polymorphism
- Call-by-value

Leroy ['93], and recently Haskell:

- **let** : **poly**morphic call-by-**name**
- $\gg=$: **mono**morphic call-by-**value**

# Goals

Combine:

- Computational **algebraic** effects
- Polymorphism
- Call-by-value

in a sound, unrestricted, Hindley-Milner type system.

# Contribution

Extend Pretnar's ['15] core calculus of effect handlers with:

1. Standard Hindley-Milner polymorphism
   type variables, type schemes, let-generalization
   (no value restriction)
2. Polymorphic type soundness (in Twelf)
3. Robustness evidence
   effect annotations, subtyping, shallow handlers.
4. Comparison with ref cells.
5. Comparison with dynamically scoped cells.
6. Sound denotational model.

For 1–5, see draft: `http://arxiv.org/abs/1605.06938`

# Algebraic effects and handlers

## Algebraic effect operations

- get : unit $\to$ int
- set : int $\to$ unit

$$\textbf{let } inc = \textbf{fun } \_ \to \text{set}(1 + \text{get}()) \textbf{ in } \ldots$$

generally:

- op : $P \to A$

## Effect handlers

$$H := \textbf{handler } \{\text{get}(\_; k) \mapsto k(5) \quad \text{set}(s; k) \mapsto k()\}$$

$$\textbf{with } H \textbf{ handle } inc(); \; inc(); \; \text{get}() \quad \rightsquigarrow^* 5$$

# Simulating global state locally

## Real state

$$H_{ST} := \textbf{handler}\ \{\quad x \mapsto \textbf{fun}\ \_ \to x$$
$$\mathrm{get}(\_; k) \mapsto \textbf{fun}\ s \to k\ s\ s$$
$$\mathrm{set}(s'; k) \mapsto \textbf{fun}\ \_ \to k\ ()\ s'\}$$

Syntactic sugar:

$$\langle c, s\rangle := (\textbf{with}\ H_{ST}\ \textbf{handle}\ c)\ s$$

Define:

$$\langle \mathrm{get}(), s\rangle \overset{st}{\rightsquigarrow} \langle s, s\rangle \qquad \langle \mathrm{set}(s'), s\rangle \overset{st}{\rightsquigarrow} \langle (), s'\rangle$$

$$\frac{\langle c_1, s\rangle \overset{st}{\rightsquigarrow} \langle c_1', s'\rangle}{\langle \textbf{let}\ x = c_1\ \textbf{in}\ c_2, s\rangle \overset{st}{\rightsquigarrow} \langle \textbf{let}\ x = c_1'\ \textbf{in}\ c_2, s'\rangle} \qquad \cdots$$

Then:

$$\frac{\langle c_1, s\rangle \overset{st}{\rightsquigarrow} \langle c_1', s'\rangle}{\langle c_1, s\rangle \rightsquigarrow^{+} \langle c_1', s'\rangle}$$

# Handlers

## Handlers summary

- Control effect that expresses real effects
- Generalise exception handlers

## Other perspectives

- Folds over free monads
- Command-response trees [Hancock-Setzer'00]
- A variant of monadic reflection [Filinski'94,96,99,10]
- Structured delimited control
  Bauer's thesis:

$$\frac{\text{handlers}}{\text{delimited control}} = \frac{\text{while loops}}{\text{goto}}$$

# Hindley-Milner type system (summary)

## Just add schemes

- Extend types with type variables: $\alpha$
- Add type schemes: $\forall \alpha_1 \cdots \alpha_n . A$
- Add type generalisation:

$$\frac{\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m; \Gamma \vdash c : A \,!\, \Sigma \qquad \alpha_1, \ldots, \alpha_n \vdash \Gamma, \Sigma}{\alpha_1, \ldots, \alpha_n; \Gamma \vdash c : (\forall \beta_1 \cdots \beta_m . A) \,!\, \Sigma} (\textsc{Gen})$$

E.g.:

$$\begin{aligned} H_{ST} := \textbf{handler}\ \{ \quad & x \mapsto \textbf{fun}\ \_ \to x \\ & \mathsf{get}(\_; k) \mapsto \textbf{fun}\ s \to k\ s\ s \\ & \mathsf{set}(s'; k) \mapsto \textbf{fun}\ \_ \to k\ ()\ s' \} \end{aligned}$$

$$H_{ST} : \forall \alpha, \beta . \alpha \,!\, \{\mathsf{get} : \mathsf{unit} \to \beta, \mathsf{set} : \beta \to \mathsf{unit}\} \Rightarrow (\beta \to \alpha \,!\, \emptyset) \,!\, \emptyset$$

# Safety

**Theorem**

*If $\vdash c : A \,!\, \Sigma$ holds, then either:*

(i) $c \leadsto c'$ *for some* $\vdash c' : A \,!\, \Sigma$;

(ii) $c = v$ *for some* $\vdash v : A$; *or*

(iii) $c = \mathrm{op}(v; y.\, c')$ *for some* $(\mathrm{op} : A_{\mathrm{op}} \to B_{\mathrm{op}}) \in \Sigma$, $\vdash v : A_{\mathrm{op}}$, *and* $y : B_{\mathrm{op}} \vdash c' : A \,!\, \Sigma$.

*In particular, when $\Sigma = \emptyset$, evaluation will not get stuck before returning a value.*

**Proof**

Formalised in Twelf[1]. $\square$

Robust under calculus variations:

effect annotations, subtyping and instances, shallow handlers.

---

[1] https://github.com/matijapretnar/twelf-eff/tree/val-restriction-local-sig

# Evaluation, following Leroy's thesis

## Feature interaction

$$\textbf{let}\,\mathrm{imp\_map} = \textbf{fun}\ f\ xs \to$$
$$\textbf{with}\ H_{ST}\ \textbf{handle}\ (\textbf{foldl}\ \ (\textbf{fun}\ x \to \mathsf{set}(f\ \ x :: \mathsf{get}\ ())\ ()\ xs;$$
$$\mathrm{reverse}(\mathsf{get}\ ()))$$
$$[\,]\ (*\ \text{initial state}\ *)\ \textbf{in}\ \ldots$$

$$\mathrm{imp\_map} : \forall\alpha\beta.(\alpha \to \beta \,!\, \Sigma) \to (\alpha\ \mathsf{list} \to \beta\ \mathsf{list} \,!\, \Sigma) \,!\, \emptyset$$

for any $\Sigma$.

## Unrestricted polymorphism

$$\textbf{let}\ id = (\textbf{fun}\ f \to f)(\textbf{fun}\ x \to x)\ \textbf{in}\ \ldots$$

$$id : \forall\alpha(\alpha \to \alpha \,!\, \emptyset)$$

## Reference cells

We believe they are not expressible.

$H_{ST}$ simulates dynamically scoped state.

# Denotational soundness

- **Relativise** Seely's System F fibrational models: **[Altenkirch et al.'14, Ulmer'68]**

$$J : \text{Types} \to \text{Schemes}$$

$$\text{Weakening} \dashv_J \forall\vec{\alpha} : \text{Types} \to \text{Schemes}$$

$$\frac{W\Gamma \longrightarrow JA}{\Gamma \longrightarrow \forall\vec{\alpha}A}$$

- Postulate a universal set $\mathcal{U} \neq \emptyset$
- Construct a relational set-theoretic model [Harper and Mitchell'93]
- Define a free fibred monad $T_{\vec{\alpha}}$

### Theorem

*The canonical morphism* $T_{\vec{\alpha}}\forall\vec{\beta}.\tau \to \forall\vec{\beta}.T_{\vec{\alpha}\times\vec{\beta}}\tau$ *is invertible.*

# Contribution

Extend Pretnar's ['15] core calculus of effect handlers with:

1. Standard Hindley-Milner polymorphism
   type variables, type schemes, let-generalization
   (no value restriction)
2. Polymorphic type soundness (in Twelf)
3. Robustness evidence
   effect annotations, subtyping, shallow handlers.
4. Comparison with ref cells.
5. Comparison with dynamically scoped cells.
6. Sound denotational model.

For 1–5, see draft: `http://arxiv.org/abs/1605.06938`

# Conclusion

## Takeaway message

- Reach beyond the value restriction
- New perspectives via algebraic effects
- Redrawn the boundary of safe polymorphism

## Further work

- Reference cells
- Delimited control
- Algorithmic concerns: inference, principal types
- Usability: effect polymorphism

# Untyped Eff

**Syntax**

| | | |
|---|---|---|
| $v ::=$ | | value |
| | $x$ | variable |
| $\mid$ | $\mathbf{true} \mid \mathbf{false}$ | boolean constants |
| $\mid$ | $\mathbf{fun}\ x \to c$ | function |
| $\mid$ | $h$ | handler |
| $h ::=$ | | handler |
| | $\mathbf{handler}\ \{ x \mapsto c_r,$ | return clause |
| | $\mathrm{op}_1(x; k) \mapsto c_1, \ldots, \mathrm{op}_n(x; k) \mapsto c_n \}$ | operation clauses |
| $c ::=$ | | computation |
| | $v$ | return value |
| $\mid$ | $\mathbf{let}\ x = c_1\ \mathbf{in}\ c_2$ | sequencing |
| $\mid$ | $\mathrm{op}(v; y.\, c)$ | operation call |
| $\mid$ | $\mathbf{if}\ v\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2$ | conditional |
| $\mid$ | $v_1\ v_2$ | application |
| $\mid$ | $\mathbf{with}\ v\ \mathbf{handle}\ c$ | handling |

## Semantics (part 1)

$$\frac{c_1 \rightsquigarrow c_1'}{\textbf{let } x = c_1 \textbf{ in } c_2 \rightsquigarrow \textbf{let } x = c_1' \textbf{ in } c_2}$$

$$\frac{}{\textbf{let } x = v \textbf{ in } c \rightsquigarrow c[v/x]} \qquad \frac{}{\textbf{if true then } c_1 \textbf{ else } c_2 \rightsquigarrow c_1}$$

$$\frac{}{\textbf{if false then } c_1 \textbf{ else } c_2 \rightsquigarrow c_2} \qquad \frac{}{(\textbf{fun } x \to c)\, v \rightsquigarrow c[v/x]}$$

$$\frac{}{\textbf{let } x = \mathrm{op}(v; y.\, c_1) \textbf{ in } c_2 \rightsquigarrow \mathrm{op}(v; y.\, \textbf{let } x = c_1 \textbf{ in } c_2)}\,(\textrm{DO-OP})$$

**Semantics (part 2)**

For every

$h = \textbf{handler}\ \{x \mapsto c_r, \mathrm{op}_1(x;k) \mapsto c_1, \ldots, \mathrm{op}_n(x;k) \mapsto c_n\}$, define:

$$\frac{c \rightsquigarrow c'}{\textbf{with}\ h\ \textbf{handle}\ c \rightsquigarrow \textbf{with}\ h\ \textbf{handle}\ c'}$$

$$\frac{}{\textbf{with}\ h\ \textbf{handle}\ (v) \rightsquigarrow c_r[v/x]}$$

$$\frac{(1 \leq i \leq n)}{\textbf{with}\ h\ \textbf{handle}\ \mathrm{op}_i(v; y.\, c) \rightsquigarrow c_i[v/x, (\textbf{fun}\ y \rightarrow \textbf{with}\ h\ \textbf{handle}\ c)/k]}$$

$$\frac{(\mathrm{op} \notin \{\mathrm{op}_1, \ldots, \mathrm{op}_n\})}{\textbf{with}\ h\ \textbf{handle}\ \mathrm{op}(v; y.\, c) \rightsquigarrow \mathrm{op}(v; y.\, \textbf{with}\ h\ \textbf{handle}\ c)}$$

# Eff types and effects

**Types**

| | | | | |
|---|---|---|---|---|
| value type | $A, B$ | $::=$ | $\alpha$ | type variable |
| | | $\mid$ | bool | boolean type |
| | | $\mid$ | $A \to \underline{C}$ | function type |
| | | $\mid$ | $\underline{C} \Rightarrow \underline{D}$ | handler type |
| computation type | $\underline{C}, \underline{D}$ | $::=$ | $A \,!\, \Sigma$ | |
| scheme | | | $\forall \vec{\alpha}.A$ | |
| effect signatures | $\Sigma$ | $::=$ | $\{\mathrm{op}_1 : A_1 \to B_1, \ldots, \mathrm{op}_n : A_n \to B_n\}$ | |

# Kind system

**Well-formed value types:**

$$\frac{\alpha \in \Theta}{\Theta \vdash \alpha} \qquad \frac{}{\Theta \vdash \mathsf{bool}} \qquad \frac{\Theta \vdash A \qquad \Theta \vdash \underline{C}}{\Theta \vdash A \to \underline{C}} \qquad \frac{\Theta \vdash \underline{C} \qquad \Theta \vdash \underline{D}}{\Theta \vdash \underline{C} \Rightarrow \underline{D}}$$

**Well-formed effect signatures, schemes, and computation types:**

$$\frac{[\Theta \vdash A_i \qquad \Theta \vdash B_i]_{1 \le i \le n}}{\Theta \vdash \{\mathsf{op}_1 : A_1 \to B_1, \ldots, \mathsf{op}_n : A_n \to B_n\}} \qquad \frac{\Theta, \vec{\alpha} \vdash A}{\Theta \vdash \forall \vec{\alpha}.A}$$

$$\frac{\Theta \vdash A \qquad \Theta \vdash \Sigma}{\Theta \vdash A \,!\, \Sigma}$$

**Well-formed polymorphic and monomorphic contexts:**

$$\frac{[\Theta \vdash \forall \vec{\alpha}.A]_{(x : \forall \vec{\alpha}.A) \in \Xi}}{\Theta \vdash \Xi} \qquad \frac{[\Theta \vdash A]_{(x : A) \in \Gamma}}{\Theta \vdash \Gamma}$$

# Type and effect system (part 1)

**Value judgements** $\boxed{\Theta; \Xi; \Gamma \vdash v : A}$**, assuming** $\Theta \vdash \Xi, \Gamma, A$**:**

$$\frac{(x : A) \in \Gamma}{\Theta; \Xi; \Gamma \vdash x : A} \qquad \frac{(x : \forall\vec{\alpha}.B) \in \Xi \qquad [\Theta \vdash A_i]_{1 \le i \le |\vec{\alpha}|}}{\Theta; \Xi; \Gamma \vdash x : B[A_i/\alpha_i]_{1 \le i \le |\vec{\alpha}|}}$$

$$\frac{}{\Theta; \Xi; \Gamma \vdash \mathbf{true} : \mathrm{bool}} \qquad \frac{}{\Theta; \Xi; \Gamma \vdash \mathbf{false} : \mathrm{bool}}$$

$$\frac{\Theta; \Xi; \Gamma, x : A \vdash c : \underline{C}}{\Theta; \Xi; \Gamma \vdash \mathbf{fun}\ x \to c : A \to \underline{C}}$$

$$\frac{\begin{array}{c}\Theta; \Xi; \Gamma, x : A \vdash c_r : B\,!\,\Sigma' \\ \left[(\mathrm{op}_i : A_i \to B_i) \in \Sigma \qquad \Theta; \Xi; \Gamma, x : A_i, k : B_i \to B\,!\,\Sigma' \vdash c_i : B\,!\,\Sigma'\right]_{1 \le i \le n} \\ \Sigma \setminus \{\mathrm{op}_i \mid 1 \le i \le n\} \subseteq \Sigma'\end{array}}{\Theta; \Xi; \Gamma \vdash \mathbf{handler}\ \{x \mapsto c_r, \mathrm{op}_1(x; k) \mapsto c_1, \ldots, \mathrm{op}_n(x; k) \mapsto c_n\} : A\,!\,\Sigma \Rightarrow B\,!\,\Sigma'}$$

# Type and effect system (part 2)

**Computation judgements** $\boxed{\Theta; \Xi; \Gamma \vdash c : A \,!\, \Sigma}$, **assuming** $\Theta \vdash \Xi, \Gamma, A$:

$$\frac{\Theta; \Xi; \Gamma \vdash v : A}{\Theta; \Xi; \Gamma \vdash v : A \,!\, \Sigma} \qquad \frac{\Theta; \Xi; \Gamma \vdash c_1 : (\forall \vec{\alpha}.A) \,!\, \Sigma \qquad \Theta; \Xi, x : \forall \vec{\alpha}.A; \Gamma \vdash c_2 : B \,!\, \Sigma}{\Theta; \Xi; \Gamma \vdash \mathbf{let}\ x = c_1\ \mathbf{in}\ c_2 : B \,!\, \Sigma}$$

$$\frac{(\mathrm{op} : A_{\mathrm{op}} \to B_{\mathrm{op}}) \in \Sigma \qquad \Theta; \Xi; \Gamma \vdash v : A_{\mathrm{op}} \qquad \Theta; \Xi; \Gamma, y : B_{\mathrm{op}} \vdash c : A \,!\, \Sigma}{\Theta; \Xi; \Gamma \vdash \mathrm{op}(v; y.\, c) : A \,!\, \Sigma}$$

$$\frac{\Theta; \Xi; \Gamma \vdash v : \mathrm{bool} \qquad \Theta; \Xi; \Gamma \vdash c_1 : \underline{C} \qquad \Theta; \Xi; \Gamma \vdash c_2 : \underline{C}}{\Theta; \Xi; \Gamma \vdash \mathbf{if}\ v\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 : \underline{C}}$$

$$\frac{\Theta; \Xi; \Gamma \vdash v_1 : A \to \underline{C} \qquad \Theta; \Xi; \Gamma \vdash v_2 : A}{\Theta; \Xi; \Gamma \vdash v_1\, v_2 : \underline{C}} \qquad \frac{\Theta; \Xi; \Gamma \vdash v : \underline{C} \Rightarrow \underline{D} \qquad \Theta; \Xi; \Gamma \vdash c : \underline{C}}{\Theta; \Xi; \Gamma \vdash \mathbf{with}\ v\ \mathbf{handle}\ c : \underline{D}}$$

# Type and effect system (part 3)

**Scheme judgement** $\boxed{\Theta; \Xi; \Gamma \vdash c : (\forall \vec{\alpha}.A) \,!\, \Sigma}$**, assuming** $\Theta \vdash \Xi, \Gamma, (\forall \vec{\alpha}.A), \Sigma$**:**

$$\frac{\Theta, \vec{\alpha}; \Xi; \Gamma \vdash c : A \,!\, \Sigma}{\Theta; \Xi; \Gamma \vdash c : (\forall \vec{\alpha}.A) \,!\, \Sigma}(\textsc{gen})$$

# Safety proof in detail

## Proof sketch (formalised in Twelf):

Prove progress and preservation by induction. Only interesting case is preservation, in the following step:

$$\dfrac{\dfrac{\dfrac{\overline{(\mathrm{op} : A_{\mathrm{op}} \to B_{\mathrm{op}}) \in \Sigma} \quad \dfrac{\vdots}{\Theta, \vec{\alpha} \vdash v : A_{\mathrm{op}}} \quad \dfrac{\vdots}{\Theta, \vec{\alpha}; y : B_{\mathrm{op}} \vdash c_1 : A \,!\, \Sigma}}{\Theta, \vec{\alpha} \vdash \mathrm{op}(v; y.\, c_1) : A!\Sigma}}{\Theta \vdash \mathrm{op}(v; y.\, c_1) : (\forall \vec{\alpha}. A) \,!\, \Sigma} \quad \dfrac{\vdots}{\Theta; x : \forall \vec{\alpha}. A \vdash c_2 : B \,!\, \Sigma}}{\Theta \vdash \mathbf{let}\ x = \mathrm{op}(v; y.\, c_1)\ \mathbf{in}\ c_2 : B \,!\, \Sigma} \qquad \rightsquigarrow$$

$$\dfrac{\overline{(\mathrm{op} : A_{\mathrm{op}} \to B_{\mathrm{op}}) \in \Sigma} \quad \dfrac{\vdots}{\Theta \vdash v : A_{\mathrm{op}}} \quad \dfrac{\dfrac{\dfrac{\vdots}{\Theta, \vec{\alpha}; y : B_{\mathrm{op}} \vdash c_1 : A \,!\, \Sigma}}{\Theta; y : B_{\mathrm{op}} \vdash c_1 : (\forall \vec{\alpha}. A) \,!\, \Sigma} \quad \dfrac{\vdots}{\Theta; x : \forall \vec{\alpha}. A \vdash c_2 : B \,!\, \Sigma}}{\Theta; y : B_{\mathrm{op}} \vdash \mathbf{let}\ x = c_1\ \mathbf{in}\ c_2 : B \,!\, \Sigma}}{\Theta \vdash \mathrm{op}(v; y.\, \mathbf{let}\ x = c_1\ \mathbf{in}\ c_2) : B \,!\, \Sigma}$$

# Programming with handlers (additional examples)

## Backtracking

Let $e :=$ $\textbf{if}\ \mathrm{toss}()\ \textbf{then}\ \textbf{if}\ \mathrm{toss}()\ \textbf{then}\ 1\ \textbf{else}\ 2\ \textbf{else}\ 3$ in

$$\textbf{handle}\ e\ \textbf{with handler}\ \{\mathrm{toss}(\_; k) \mapsto k\ \textbf{true}\} \rightsquigarrow^* 1$$

$$\begin{aligned}\textbf{handle}\ e\ \textbf{with handler}\ \{\ & x \mapsto \textbf{fun}\ \_ \to x \\ & \mathrm{toss}(\_; k) \mapsto \textbf{fun}\ b \to k\ b\ (\textbf{not}\ b) \\ & \}\ \textbf{true} \rightsquigarrow^* 2\end{aligned}$$

$$\begin{aligned}\textbf{handle}\ e\ \textbf{with handler}\ \{\ & x \mapsto [x] \\ & \mathrm{toss}(\_; k) \mapsto (k\ \textbf{true})@(k\ \textbf{false}) \\ & \} \rightsquigarrow^* 1\end{aligned}$$

## Delimited continuations

Taking

$$\begin{aligned} \mathbf{S_0}\ k.e &:= \mathrm{shift}_0\ (\mathbf{fun}\ k \to e) \\ \mathbf{reset}\ \ e &:= \mathbf{with\ handler}\ \{\mathrm{shift}_0(f; k) \mapsto f\ \ k\}\ \mathbf{handle}\ e \end{aligned}$$

simulates shift0/reset0:

$$\mathbf{reset}\ \ \mathcal{C}[\mathbf{S_0}\ k.e] \leadsto^* e[\mathbf{fun}\ x \to \mathbf{reset}\ \ \mathcal{C}[x]/k]$$

but our type system will not be able to type it.

## Images

- http://cfensi.files.wordpress.com/2014/01/frozen-let-it-go.png