# On the expressive power of user-defined effects: effect handlers, monadic reflection, and delimited control

Work in Progress

Ohad Kammar
<ohad.kammar@cs.ox.ac.uk>
joint work with
Yannick Forster, Sam Lindley, and Matija Pretnar

The 4th Southern REPLS
Imperial College
27 September 2016

# Effect oriented programming

## Native effects

- I/O.
- Mutable state.
- Randomness and non-determinism.

## User-defined effects

- Parsing.
- Constraint solving.
- Proof-search tactics.
- Redefine existing effects?

# A brief history of functional programming and effects

## A love-hate story

Effects are harmful...

- Disallow useful compiler optimisations.
- Break referential transparency.
- Depart from the $\lambda$-calculus ($\beta\eta$-equality, confluence).

But are useful!

## A rift and a bridge

ML and Scheme vs. Haskell.
Monads [Moggi'89, Wadler'91], now in Haskell, ML, Scheme, F$*$, C++...

# Algebraic effects

## Monad issues

- No interface for effects.
- Compositionality and modularity issues.
- Steep learning curve.

## Plotkin-Power-Pretnar-Bauer

- Add effect operations to Moggi's theory [Plotkin-Power'02,'03].
- Add exception handlers, and more generally, effect handlers [Plotkin-Pretnar'09].
- Programming with algebraic effects and handlers [Bauer-Pretnar'15].

# Delimited control

### Goto on steroids

Effect handlers are a new kind of delimited control effect.
But(!):

- Clean denotational semantics. [Plotkin-Pretnar'09]
- Clean program logic. [Pretnar's thesis]
- Clean meta-theory: strong normalisation and type-and-effect systems [K-Lindley-Oury'13], unrestricted polymorphism [K-Pretnar'16], . . .

# Basic research question

## Monads, handlers, and delimited control

- How do these abstractions compare?
- How to compare these abstractions?

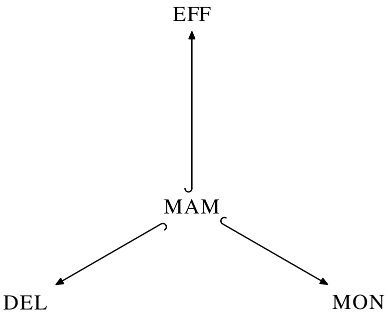
## Macro expressibility [Fellisen'90]

- Expressive power of Turing-complete languages.
- Computability and complexity reductions are too crude.
- Macro translations:
  - Keep shared fragment identical.
  - Compositional a.k.a. local a.k.a. homomorphic.

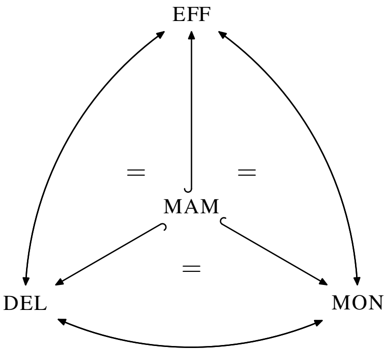
# Contribution: bird's eye

MAM

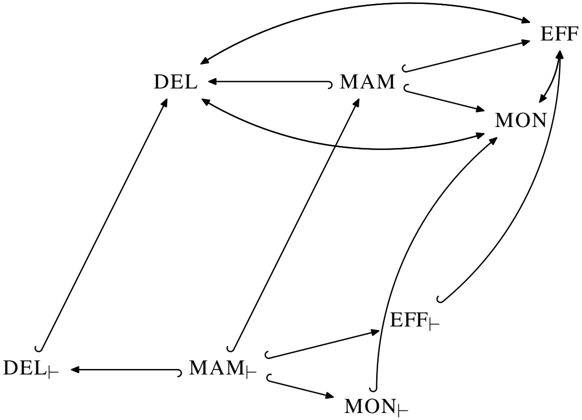
# Contribution: bird's eye



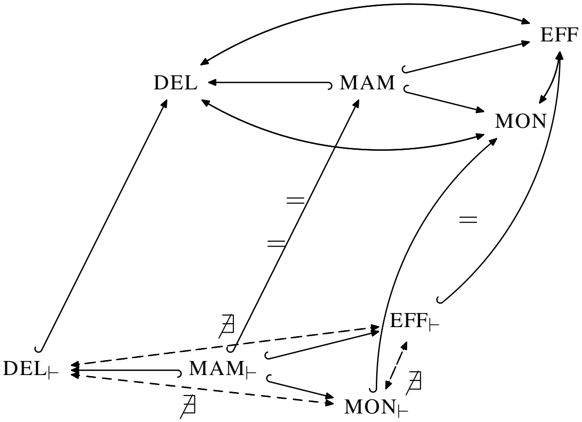
# Contribution: bird's eye



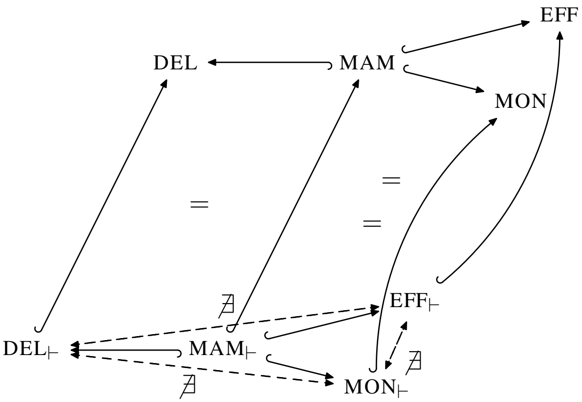
# Contribution: bird's eye



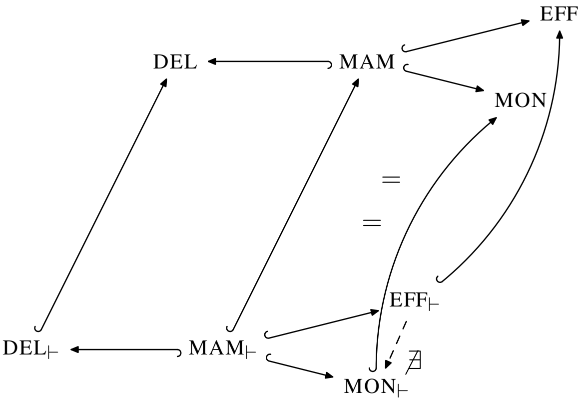
# Contribution: bird's eye



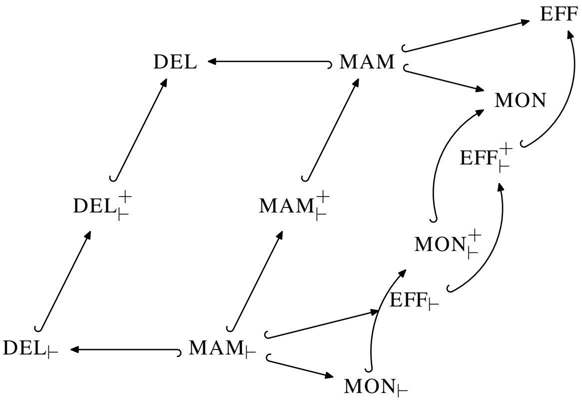
# Contribution: bird's eye



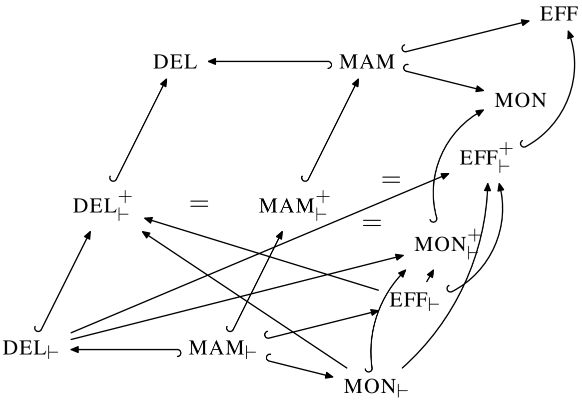
# Contribution: bird's eye



# Contribution: bird's eye



# Contribution: bird's eye



# Talk structure

- Short tutorial on algebraic effects, monadic reflection, and delimited control

Rationale: Most fiddly task was simplifying and unifying calculi.

- Time pertmitting: discuss the negative result.

# The lambda calculus with effects (MAM)

## Syntax

| | | | |
|---|---|---|---|
| $V, W$ | $::=$ | | values |
| | | $x$ | variable |
| | $\mid$ | $()$ | unit value |
| | $\mid$ | $(V_1, V_2)$ | pairing |
| | $\mid$ | $\mathbf{inj}_i\ V$ | variant constructor |
| | $\mid$ | $\{M\}$ | thunk |
| $M, N$ | $::=$ | | computations |
| | | $\mathbf{split}(V, x_1.x_2.M)$ | pattern matching: product |
| | $\mid$ | $\mathbf{case}_0(V)$ | void |
| | $\mid$ | $\mathbf{case}(V, \mathbf{inj}_1\ x_1.M_1$ <br> $, \mathbf{inj}_2\ x_2.M_2)$ | variants |
| | $\mid$ | $V!$ | force |
| | $\mid$ | $\mathbf{return}\ \ V$ | returner |
| | $\mid$ | $\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N$ | sequencing (monadic bind) |
| | $\mid$ | $\lambda x.M$ | function abstraction |
| | $\mid$ | $M\ \ V$ | function application |
| | $\mid$ | $\langle M_1, M_2 \rangle$ | computation pair |
| | $\mid$ | $\mathbf{prj}_i\ M$ | projection |

# The lambda calculus with effects (MAM)

## Syntax (CBPV)

| | | | |
|---|---|---|---|
| $V, W$ | $::=$ | | values |
| | | $x$ | variable |
| | $\mid$ | $()$ | unit value |
| | $\mid$ | $(V_1, V_2)$ | pairing |
| | $\mid$ | $\mathbf{inj}_i\ V$ | variant constructor |
| | $\mid$ | $\{M\}$ | thunk |
| $M, N$ | $::=$ | | computations |
| | | $\mathbf{split}(V, x_1.x_2.M)$ | pattern matching: product |
| | $\mid$ | $\mathbf{case_0}(V)$ | void |
| | $\mid$ | $\mathbf{case}(V, \mathbf{inj}_1\ x_1.M_1$ $, \mathbf{inj}_2\ x_2.M_2)$ | variants |
| | $\mid$ | $V!$ | force |
| | $\mid$ | $\mathbf{return}\ \ V$ | returner |
| | $\mid$ | $\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N$ | sequencing (monadic bind) |
| | $\mid$ | $\lambda x.M$ | function abstraction |
| | $\mid$ | $M\ \ V$ | function application |
| | $\mid$ | $\langle M_1, M_2 \rangle$ | computation pair |
| | $\mid$ | $\mathbf{prj}_i\ M$ | projection |

# The lambda calculus with effects (MAM)

## Operational semantics

**Reduction frames and contexts**

| | | | |
|---|---|---|---|
| $\mathcal{B}$ | $::=$ | $\textbf{let } x \leftarrow [\,] \textbf{ in } N \mid [\,]\ V \mid \textbf{prj}_i\,[\,]$ | basic frames |
| $\mathcal{F}$ | $::=$ | $\mathcal{B}$ | computation frames |
| $\mathcal{C}$ | $::=$ | | evaluation context |
| | | $[\ \ ]$ | hole |
| | | $\mathcal{C}[\mathcal{F}[\ \ ]]$ | layered frame |

**Beta reduction** $\boxed{M \rightsquigarrow_\beta M'}$

$$
\begin{array}{lrcl}
(\beta.\times) & \textbf{split}(\,(V_1, V_2), x_1.x_2.M) & \rightsquigarrow_\beta & M[V_1/x_1, V_2/x_2] \\
(\beta.+) & \textbf{case}(\textbf{inj}_i\ V, \textbf{inj}_1\ x_1.M_1, \textbf{inj}_2\ x_2.M_2) & \rightsquigarrow_\beta & M_i[V/x_i] \\
(\beta.U) & \{M\}! & \rightsquigarrow_\beta & M \\
(\beta.F) & \textbf{let } x \leftarrow \textbf{return } V \textbf{ in } M & \rightsquigarrow_\beta & M[V/x] \\
(\beta.\to) & (\lambda x.M)\ V & \rightsquigarrow_\beta & M[V/x] \\
(\beta.\&) & \textbf{prj}_i\,\langle M_1, M_2\rangle & \rightsquigarrow_\beta & M_i
\end{array}
$$

**Reduction** $\boxed{M \rightsquigarrow M'}$

$$\frac{M \rightsquigarrow_\beta M'}{\mathcal{C}[M] \rightsquigarrow \mathcal{C}[M']}$$

# The lambda calculus with effects (MAM)

## Types

| | | | |
|---|---|---|---|
| $E$ | $::=$ | | effects |
| | | $\emptyset$ | pure effect |
| $K$ | $::=$ | | kinds |
| | $\mid$ | **Eff** | effects |
| | | **Val** | values |
| | $\mid$ | $\mathbf{Comp}_E$ | $E$-computations |
| | $\mid$ | **Ctxt** | environments |
| $A, B$ | $::=$ | | value types |
| | | $\alpha$ | type variable |
| | $\mid$ | $1$ | unit |
| | $\mid$ | $A_1 \times A_2$ | value products |
| | $\mid$ | $0$ | empty |
| | $\mid$ | $A_1 + A_2$ | variant |
| | $\mid$ | $U_E C$ | thunks |
| $C, D$ | $::=$ | | computation types |
| | | $FA$ | returners |
| | $\mid$ | $A \to C$ | functions |
| | $\mid$ | $C_1 \,\&\, C_2$ | computation products |
| $\Theta$ | $::=$ | $\alpha_1, \ldots, \alpha_n$ | type environments |
| $\Gamma, \Delta$ | $::=$ | $x_1 : A_1, \ldots, x_n : A_n$ | environments |

# The lambda calculus with effects (MAM)

### Denotational semantics

Standard, using sets and functions. Using Hermida's ['93] lifting:

### Theorem (adequacy)

*Denotational equivalence implies contextural equivalence: for all* $\Theta; \Gamma \vdash_E P, Q : X$, *if* $[\![P]\!] = [\![Q]\!]$ *then* $P \simeq Q$.

### Corollary (soundness and strong normalisation)

*All well-typed closed ground returners reduce to a normal form: for all* $; \vdash_\emptyset M : FG$ *there exists some* $; \vdash V : G$ *such that* $[\![\mathbf{return}\ V]\!] = [\![M]\!]$ *and*

$$M \rightsquigarrow^\star \mathbf{return}\ V$$

# Effect handlers

## Syntax [K-Lindley-Oury'13]

$$
\begin{array}{llll}
M, N & ::= & \ldots & \text{computations} \\
 & | & \mathrm{op}\ V & \quad \text{operation call} \\
 & | & \textbf{handle}\ M\ \textbf{with}\ H & \quad \text{handling construct} \\
H & ::= & & \text{handlers} \\
 & & \{\textbf{return}\ x \mapsto M\} & \quad \text{return clause} \\
 & | & H \uplus \{\mathrm{op}\, p\, k \mapsto N\} & \quad \text{operation clause} \\
 & & & \text{(where } \mathrm{op} \text{ does not occur in } H\text{)}
\end{array}
$$

## Operational semantics

**Reduction frames and contexts**

$$\begin{array}{llll} \cdots & \mathcal{F} & ::= \ldots & \text{computation frame} \\ & & \mid \quad \textbf{handle } [\ \ ] \textbf{ with } H & \\ & \mathcal{H} & ::= [\ \ ] \mid \mathcal{H}[\mathcal{B}[\ \ ]] & \text{hoisting context} \end{array}$$

**Beta reduction**

$$\cdots \quad (\mathit{handle}.F) \quad \textbf{handle } (\textbf{return } V) \textbf{ with } H \rightsquigarrow_\beta M[V/x]$$
$$\text{where } H^{\textbf{return}} = \lambda x.M$$

$$(\mathit{handle}.\mathrm{op}) \quad \textbf{handle } \mathcal{H}[\mathrm{op}\ V] \textbf{ with } H \rightsquigarrow_\beta$$
$$N[V/p, \{\lambda x.\textbf{handle } \mathcal{H}[\textbf{return } x] \textbf{ with } H\}/k]$$
$$\text{where } H^{\mathrm{op}} = \lambda p\, k.N \text{ and } x \notin FV(H, \mathcal{H})$$

# Effect handlers

## Types

$$
\begin{array}{llll}
E & ::= \ldots & & \text{effects} \\
 & \mid & \{\mathrm{op} : A \to B\} \uplus E & \quad \text{arity assignment} \\
K & ::= \ldots & & \text{kinds} \\
 & \mid & \mathbf{Hndlr} & \quad \text{handlers} \\
R & ::= A \;{}^{E}\!\!\Rightarrow^{E'}\; C & & \text{handler types} \quad \cdots
\end{array}
$$

**Computation typing** $\boxed{\Theta; \Gamma \vdash_E M : C}$ $(\Theta \vdash_k \Gamma : \mathbf{Ctxt}, E : \mathbf{Eff}, C : \mathbf{Comp}_E)$

$$
\ldots \quad \frac{(\mathrm{op} : A \to B) \in E \qquad \Theta; \Gamma \vdash V : A}{\Theta; \Gamma \vdash_E \mathrm{op}\ V : FB} \quad \frac{\Theta; \Gamma \vdash_E M : FA \qquad \Theta; \Gamma \vdash H : A \;{}^{E}\!\!\Rightarrow^{E'}\; C}{\Theta; \Gamma \vdash_{E'} \mathbf{handle}\ M\ \mathbf{with}\ H : C}
$$

**Handler typing** $\boxed{\Theta; \Gamma \vdash H : R}$ $(\Theta \vdash_k \Gamma : \mathbf{Ctxt}, R : \mathbf{Hndlr})$

$$
\frac{\begin{array}{c} E = \{\mathrm{op}_i : A_i \to B_i\}_i \\ H = \{\mathbf{return}\ x \mapsto M\} \uplus \{\mathrm{op}_i\ p\ k \mapsto N_i\}_i \\ {[\Theta; \Gamma, p : A_i, k : U_{E'}(B_i \to C) \vdash_{E'} N_i : C]_i} \qquad \Theta; \Gamma, x : A \vdash_{E'} M : C \end{array}}{\Theta; \Gamma \vdash H : A \;{}^{E}\!\!\Rightarrow^{E'}\; C}
$$

## Denotational semantics

Using free monads for a signature. Using a folklore lifting [cf. K'14] we have **adequacy**, **soundness** and **strong normalisation**.

# Monadic reflection

## Syntax [Filinski'94-10]

$$\begin{array}{llll} T & ::= & \mathbf{mon}(M, N) & \text{monads} \\ M, N & ::= & \ldots & \text{computations} \\ & \mid & \hat{\mu}(N) & \quad \text{reflect} \\ & \mid & [N]^{T} & \quad \text{reify} \end{array}$$

# Monadic reflection

## Operational semantics

**Reduction frames and contexts**

$$\begin{array}{lll} \mathcal{F} & ::= \mathcal{B} \mid [[\ ]]^{T} & \text{computation frames} \\ \mathcal{H} & ::= [\ \ ] \mid \mathcal{H}[\mathcal{B}[\ \ ]] & \text{hoisting contexts} \quad \cdots \end{array}$$

**Beta reduction**

$$\begin{array}{lll} \cdots \quad (\mathit{reify}) & [\mathbf{return}\ V]^{T} \rightsquigarrow_{\beta} N_u V \\ (\mathit{reflect}) & [\mathcal{H}[\hat{\mu}(N)]]^{T} \rightsquigarrow_{\beta} N_b\ \{N\}\ \{(\lambda x.[\mathcal{H}[\mathbf{return}\ x]]^{T})\} \end{array}$$

for every $T = \mathbf{mon}(N_u, N_b)$.

# Monadic reflection

## Types

$$\begin{aligned} E \; ::= & \; \ldots & \text{effects} \\ & E \prec \langle \alpha.C, N, M \rangle & \text{layered monad} \end{aligned}$$

**Monad typing** $\boxed{\Theta \vdash_m T : E}$ $(\Theta \vdash_k E : \mathbf{Eff})$

$$\frac{\Theta, \alpha; \vdash_E N_u : \alpha \to C \qquad \Theta, \alpha, \beta; \vdash_E N_b : U_E C \to U_E(\alpha \to C[\beta/\alpha]) \to C[\beta/\alpha]}{\Theta \vdash_m \mathbf{mon}(N_u, N_b) : E \prec \langle \alpha.C, N_u, N_b \rangle}$$

**Computation typing** $\boxed{\Theta; \Gamma \vdash_E M : C}$ $(\Theta \vdash_k \Gamma : \mathbf{Ctxt}, E : \mathbf{Eff}, C : \mathbf{Comp}_E)$

$$\ldots \quad \frac{\Theta \vdash_m T : E \prec \langle \alpha.C, N_u, N_b \rangle \qquad \Theta; \Gamma \vdash_{E \prec \langle \alpha.C, N_u, N_b \rangle} N : A}{\Theta; \Gamma \vdash_E [N]^T : C[A/\alpha]}$$

$$\frac{\Theta; \Gamma \vdash_E N : C[A/\alpha]}{\Theta; \Gamma \vdash_{E \prec \langle \alpha.C, N_u, N_b \rangle} \hat{\mu}(N) : FA}$$

## Caveats

Essentially top level monad declarations to avoid dependent types.
Requires type variables.

# Monadic reflection

**Denotational semantics**

Partial semantics due to the invalidity of the monad laws.
Using $TT$-lifting, we have **adequacy**, **soundness** and **strong normalisation** for terms whose semantics is defined.

**Lemma (Finite denotation property)**

*For any tuple $\theta = \langle X_\alpha \rangle_{\alpha \in \Theta}$ of finite sets, if the types $A$ and $C$ have well-defined denotations for $\theta$, they denote finite sets.*

# Delimited control

## Syntax

$$
\begin{array}{llll}
M, N & ::= & \ldots & \text{computations} \\
 & \mid & \mathbf{S_0} k.M & \text{shift-0} \\
 & \mid & \langle M | x.N \rangle & \text{reset}
\end{array}
$$

# Delimited control

## Operational semantics

**Reduction frames and contexts**

$$\begin{array}{lllll} \cdots & \mathcal{F} & ::= \ldots & & \text{computation frame} \\ & & \mid & \langle[\ \ ]|x.N\rangle & \\ & \mathcal{H} & ::= [\ \ ] \mid \mathcal{H}[\mathcal{B}[\ \ ]] & & \text{hoisting context} \end{array}$$

**Beta reduction**

$$\cdots \quad (\mathit{reset}) \quad \langle(\mathbf{return}\ V)|x.M\rangle \rightsquigarrow_\beta M[V/x]$$

$$(\mathit{shift}_0) \quad \langle\mathcal{H}[\mathbf{S_0} k.M]|x.N\rangle \rightsquigarrow_\beta M[\lambda y.\langle\mathcal{H}[\mathbf{return}\ y]|x.N\rangle/k]$$

# Delimited control

## Types [Danvy and Filinski, sketched]

$$\begin{aligned} E \; &::= \ldots \quad \text{effects} \\ &\mid \quad E, A \end{aligned}$$

**Computation typing** $\boxed{\Theta; \Gamma \vdash_E M : C}$ $(\Theta \vdash_k \Gamma : \mathbf{Ctxt}, E : \mathbf{Eff}, C : \mathbf{Comp}_E)$

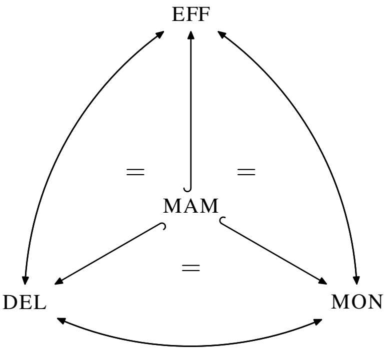
$$\ldots \quad \frac{\Theta; \Gamma, k : U_E(B \to FA) \vdash_E M : FA}{\Theta; \Gamma \vdash_{E,A} \mathbf{S_0} k.M : FB} \quad \frac{\Theta; \Gamma \vdash_{E,A} M : FA \qquad \Theta; \Gamma, x : A \vdash_E N : C}{\Theta; \Gamma \vdash_E \langle M | x.N \rangle : C}$$

# Delimited control

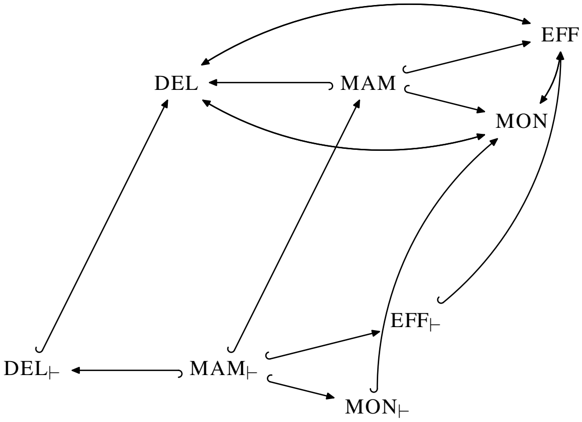
### Denotational semantics

An adequate denotational semantics is an open problem.
Perhaps [Atkey'06]'s parameterised monads?

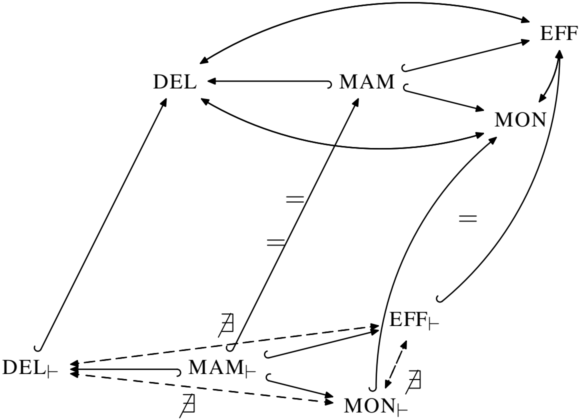
# Untyped translations



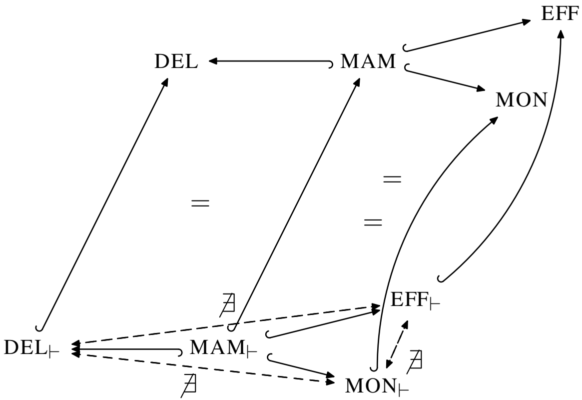
# Typed translations



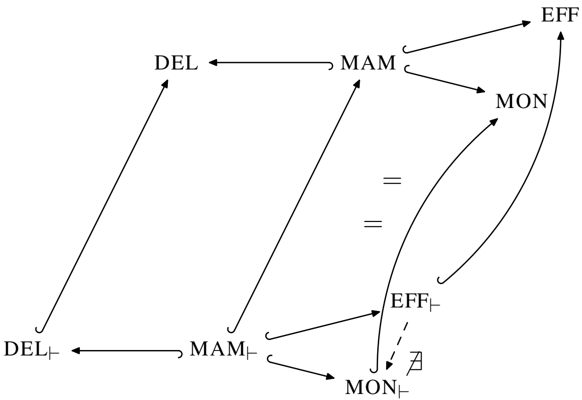
# Typed translations



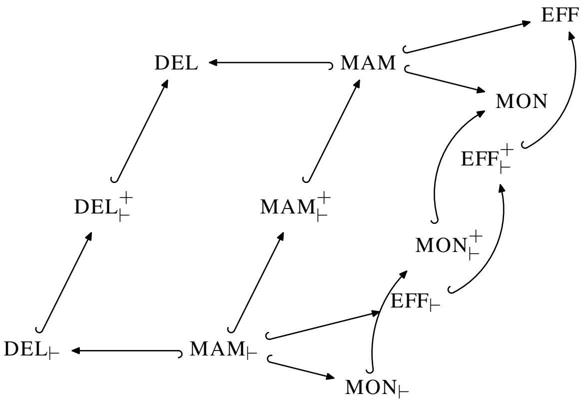
# Typed translations



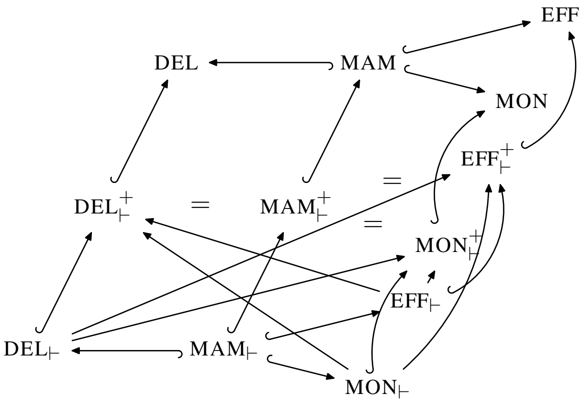
# Typed translations



# Richer types



# Richer types



# Richer types

## Conjectures

- Handlers + polymorphic arities express delimited control and monadic reflection
- Delimited control + effect polymorphism express effect handlers and monadic reflection.
- Parameterised monadic reflection and polymorphism express effect handlers and delimited control.

# Summary and conclusions

- Rigorous set-up for comparing user-defined effect abstractions.
- New and folklore macro translations.
- Inexpressivity result via a denotational invariant.
- Type system extensions accepting the translations.