

Recursion in two level type theories for effects

Work in progress

Rasmus Ejlers Møgelberg
IT University of Copenhagen

Two level type theories

- Moggi introduced monads as general framework for modelling effects
- 2-level type theories based on distinction between computation types and value types
- This has been used in Levy's Call-By-Push-Value (CBPV)
- And in recent work with Alex Simpson on parametricity
- Today: Recursion in 2-level type theories

Two levels of types

- Value types

$$A, B ::= FA \mid A \rightarrow B \mid \underline{A} \multimap \underline{B}$$

- Computation types

$$\underline{A}, \underline{B} ::= FA \mid A \rightarrow \underline{B}$$

- Computation types \subseteq Value types

Terms

- Terms in context

$$\Gamma \mid - \vdash t : A$$

$$\Gamma \mid x : \underline{B} \vdash t : \underline{A} .$$

Terms

- Terms in context

$$\Gamma \mid - \vdash t : A \qquad \Gamma \mid x : \underline{B} \vdash t : \underline{A} .$$

- Terms

$$t, s ::= x \mid \lambda x : A. t \mid s(t) \mid \lambda^\circ x : \underline{A}. t$$

Typing rules

$$\overline{\Gamma, x:A \mid - \vdash x:A}$$

$$\overline{\Gamma \mid x:\underline{A} \vdash x:\underline{A}}$$

$$\frac{\Gamma, x:A \mid \Delta \vdash t:B}{\Gamma \mid \Delta \vdash \lambda x:A. t: A \rightarrow B}$$

$$\frac{\Gamma \mid x:\underline{A} \vdash t:\underline{B}}{\Gamma \mid - \vdash \lambda^\circ x:\underline{A}. t: \underline{A} \multimap \underline{B}}$$

$$\frac{\Gamma \mid \Delta \vdash s: A \rightarrow B \quad \Gamma \mid - \vdash t: A}{\Gamma \mid \Delta \vdash s(t): B}$$

$$\frac{\Gamma \mid - \vdash s: \underline{A} \multimap \underline{B} \quad \Gamma \mid \Delta \vdash t: \underline{A}}{\Gamma \mid \Delta \vdash s(t): \underline{B}}$$

Typing rules, monadic type

$$\frac{\Gamma \mid - \vdash t : \mathbf{B}}{\Gamma \mid - \vdash !t : \mathbf{FB}}$$

$$\frac{\Gamma \mid \Delta \vdash t : \mathbf{FB} \quad \Gamma, x : \mathbf{B} \mid - \vdash u : \underline{\underline{\mathbf{A}}}}{\Gamma \mid \Delta \vdash \text{let } !x \text{ be } t \text{ in } u : \underline{\underline{\mathbf{A}}}}$$

Algebraic operations

- Specialise the theory to printing by adding

$$\text{print}_a : \underline{A} \rightarrow \underline{A}$$

for each computation type \underline{A} and each a in alphabet A

Algebraic operations

- Specialise the theory to printing by adding

$$\text{print}_a : \underline{A} \rightarrow \underline{A}$$

for each computation type \underline{A} and each a in alphabet A

- Or to global state by adding

$$\text{lookup}_{\underline{A}} : (\text{nat} \rightarrow \underline{A}) \rightarrow \text{loc} \rightarrow \underline{A}$$

$$\text{update}_{\underline{A}} : \underline{A} \rightarrow \text{nat} \rightarrow \text{loc} \rightarrow \underline{A}$$

for each computation type \underline{A}

Maps commuting with operations

- Maps of homomorphism type $f: \underline{A} \rightarrow \underline{B}$ commute with operations
- Examples:
 - $\text{print}_a(f(x)) = f(\text{print}_a(x))$
 - $\text{lookup}_{\underline{B}, l}(\lambda x: \text{nat. } f(t)) = f(\text{lookup}_{\underline{A}, l}(\lambda x: \text{nat. } t))$

Models

- \mathcal{V} cartesian closed, T strong monad on \mathcal{V}

$$\begin{array}{c} \mathcal{V}^T \\ F \uparrow \downarrow U \\ \mathcal{V} \end{array}$$

- Where \mathcal{V}^T category of algebras $\xi: TX \rightarrow X$ for monad T
- Model value types $\mathcal{V}[[A]] \in \mathcal{V}$
- Model computation types $\mathcal{V}^T[[\underline{B}]] \in \mathcal{V}^T$
- $\mathcal{V}[[\underline{B}]] = U(\mathcal{V}^T[[\underline{B}]])$

Models II

- More generally adjunctions

$$\begin{array}{ccc} & \mathcal{C} & \\ F \uparrow & \left(+ \right) & \downarrow U \\ & \mathcal{V} & \end{array}$$

satisfying certain conditions (e.g. $C(\underline{X}, \underline{Y})$ should be an object of \mathcal{V}) can be used.

Example: Printing

$$\begin{array}{c} \mathbf{PA} \\ A^* \times (-) \left(\begin{array}{c} \uparrow \\ - \\ \downarrow \end{array} \right) U \\ \mathbf{Set} \end{array}$$

- **PA** cat. of sets with left A^* action $(X, \cdot : A^* \times X \rightarrow X)$

Example: Printing

$$\begin{array}{c} \mathbf{PA} \\ A^{\star} \times (-) \left(\begin{array}{c} \uparrow \\ - \\ \downarrow \end{array} \right) U \\ \mathbf{Set} \end{array}$$

- **PA** cat. of sets with left A^{\star} action ($X, \cdot : A^{\star} \times X \rightarrow X$)
- \rightarrow interpreted as ordinary maps in sets, \dashv as homomorphisms

Example: Printing

$$\begin{array}{c} \mathbf{PA} \\ A^* \times (-) \left(\begin{array}{c} \uparrow \\ - \\ \downarrow \end{array} \right) U \\ \mathbf{Set} \end{array}$$

- **PA** cat. of sets with left A^* action ($X, \cdot : A^* \times X \rightarrow X$)
- \rightarrow interpreted as ordinary maps in sets, \rightarrow as homomorphisms
- $\text{print}_a : \underline{X} \rightarrow \underline{X}$ is interpreted as $a \cdot (-)$

Polymorphism and effects

- Type theory for (parametric) polymorphism and effects
- Value types

$$A, B ::= X \mid A \rightarrow B \mid \forall X. A \mid \underline{X} \mid \underline{A} \multimap \underline{B} \mid \forall \underline{X}. A$$

- Computation types

$$\underline{A}, \underline{B} ::= A \rightarrow \underline{B} \mid \forall X. \underline{A} \mid \underline{X} \mid \forall \underline{X}. \underline{A} .$$

- F can be encoded using parametricity:

$$FA = \forall \underline{X}. (A \rightarrow \underline{X}) \rightarrow \underline{X}.$$

Inductive and coinductive types in PE

- Parametric polymorphism allows for encoding of (co)inductive value and computation types

Inductive and coinductive types in PE

- Parametric polymorphism allows for encoding of (co)inductive value and computation types
- e.g. inductive computation types

$$\mu^\circ \underline{X}. \underline{A} \stackrel{\text{def}}{=} \forall \underline{X}. (\underline{A} \multimap \underline{X}) \rightarrow \underline{X} \quad (\underline{X} \text{ +ve in } \underline{A})$$

Inductive and coinductive types in PE

- Parametric polymorphism allows for encoding of (co)inductive value and computation types
- e.g. inductive computation types

$$\mu^\circ \underline{X}. \underline{A} \stackrel{\text{def}}{=} \forall \underline{X}. (\underline{A} \multimap \underline{X}) \rightarrow \underline{X} \quad (\underline{X} \text{ +ve in } \underline{A})$$

- Generalise encodings from combination of recursion and parametric polymorphism

Inductive and coinductive types in PE

- Parametric polymorphism allows for encoding of (co)inductive value and computation types
- e.g. inductive computation types

$$\mu^\circ \underline{X}. \underline{A} \stackrel{\text{def}}{=} \forall \underline{X}. (\underline{A} \multimap \underline{X}) \rightarrow \underline{X} \quad (\underline{X} \text{ +ve in } \underline{A})$$

- Generalise encodings from combination of recursion and parametric polymorphism
- Fixed points + inductive and coinductive types give general recursive types

Inductive and coinductive types in PE

- Parametric polymorphism allows for encoding of (co)inductive value and computation types
- e.g. inductive computation types

$$\mu^\circ \underline{X}. \underline{A} \stackrel{\text{def}}{=} \forall \underline{X}. (\underline{A} \multimap \underline{X}) \rightarrow \underline{X} \quad (\underline{X} \text{ +ve in } \underline{A})$$

- Generalise encodings from combination of recursion and parametric polymorphism
- Fixed points + inductive and coinductive types give general recursive types
- Can this be done for general effects?

How to add recursion?

- Would like to add fixpoint operator $\text{fix}_X: (X \rightarrow X) \rightarrow X$
- Should X be a value type or computation type?

How to add recursion?

- Would like to add fixpoint operator $\text{fix}_X: (X \rightarrow X) \rightarrow X$
- Should X be a value type or computation type?
- Would also like reasoning principles such as the uniformity principle:

$$h \circ f = g \circ h \Rightarrow h(\text{fix } f) = \text{fix } g$$

(if $h(\perp) = \perp$)

How to add recursion?

- Would like to add fixpoint operator $\text{fix}_X: (X \rightarrow X) \rightarrow X$
- Should X be a value type or computation type?
- Would also like reasoning principles such as the uniformity principle:

$$h \circ f = g \circ h \Rightarrow h(\text{fix } f) = \text{fix } g$$

(if $h(\perp) = \perp$)

- This suggests three notions of maps: \rightarrow , \dashv as above and $\circ\rightarrow$ for strict maps

Examples of monads

- Models are monads on \mathbf{Cpo} (complete partial orders)

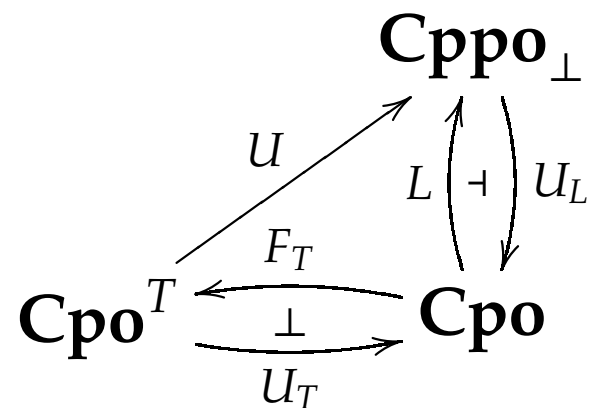
Examples of monads

- Models are monads on **Cpo** (complete partial orders)
- Examples:
 - State and recursion $TX = (S \times X)_{\perp}^S$ (where $S = \omega^L$ discrete cpo of states)
 - Printing and recursion: $TX = \mu Z. (A \times Z + X)_{\perp}$ (where A is alphabet)
 - Commutative combination of print and recursion: $TX = (A^{\star} \times X)_{\perp}$

Examples of monads

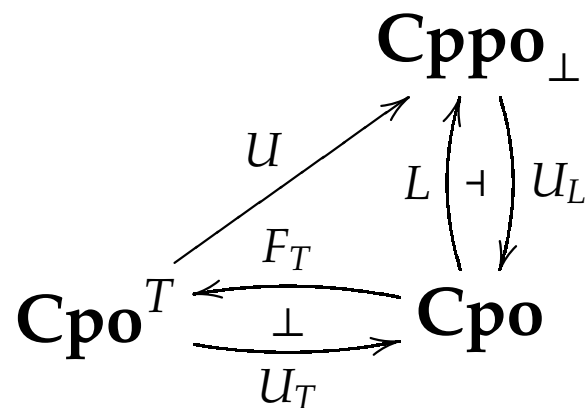
- Models are monads on **Cpo** (complete partial orders)
- Examples:
 - State and recursion $TX = (S \times X)_{\perp}^S$ (where $S = \omega^L$ discrete cpo of states)
 - Printing and recursion: $TX = \mu Z. (A \times Z + X)_{\perp}$ (where A is alphabet)
 - Commutative combination of print and recursion:
 $TX = (A^{\star} \times X)_{\perp}$
- In all cases TX is pointed, and any algebra for T is pointed

Semantic picture



- $U_L U = U_T$

Semantic picture



- $U_L U = U_T$
- Thm: U has a left adjoint

Pointed homsets

- Given a pair of algebras $\xi: TX \rightarrow X, \xi': TY \rightarrow Y$ the homset

$$\mathbf{Cpo}^T(\xi, \xi')$$

is a cpo.

- Is it a pointed cpo?

Pointed homsets

- Given a pair of algebras $\xi: TX \rightarrow X, \xi': TY \rightarrow Y$ the homset

$$\mathbf{Cpo}^T(\xi, \xi')$$

is a cpo.

- Is it a pointed cpo?
- Thm: For T given by operations, $\mathbf{Cpo}^T(\xi, \xi')$ contains $\lambda x: X. \perp$ for all ξ, ξ' iff all operations are strict.

Pointed homsets

- Given a pair of algebras $\xi: TX \rightarrow X, \xi': TY \rightarrow Y$ the homset

$$\mathbf{Cpo}^T(\xi, \xi')$$

is a cpo.

- Is it a pointed cpo?
- Thm: For T given by operations, $\mathbf{Cpo}^T(\xi, \xi')$ contains $\lambda x: X. \perp$ for all ξ, ξ' iff all operations are strict.
- Condition holds for state: $TX = (S \times X)_{\perp}^S$ and commutative combination of print and recursion
 $TX = (A^{\star} \times X)_{\perp}$

Pointed homsets

- Given a pair of algebras $\xi: TX \rightarrow X, \xi': TY \rightarrow Y$ the homset

$$\mathbf{Cpo}^T(\xi, \xi')$$

is a cpo.

- Is it a pointed cpo?
- Thm: For T given by operations, $\mathbf{Cpo}^T(\xi, \xi')$ contains $\lambda x: X. \perp$ for all ξ, ξ' iff all operations are strict.
- Condition holds for state: $TX = (S \times X)_{\perp}^S$ and commutative combination of print and recursion
 $TX = (A^* \times X)_{\perp}$
- Condition does not hold for usual combination of print and recursion $TX = \mu Z. (A \times Z + X)_{\perp}$

Type theory for strict operations

$$\begin{array}{c} \mathbf{Cpo}^T \\ \begin{array}{c} \uparrow \\ F \left(\dashv \right) U \\ \downarrow \end{array} \\ \mathbf{Cppo}_\perp \end{array}$$

Type theory for strict operations

$$\begin{array}{c} \mathbf{Cpo}^T \\ F \left(\begin{array}{c} \uparrow \\ + \\ \downarrow \end{array} \right) U \\ \mathbf{Cppo}_\perp \end{array}$$

- Value types in \mathbf{Cppo}_\perp , computation types in \mathbf{Cpo}^T

$$\underline{A} ::= FA \mid A \circ \rightarrow \underline{B}$$

$$A ::= FA \mid A \circ \rightarrow B \mid \underline{A} \multimap \underline{B} \mid LA$$

- Define $A \rightarrow B = LA \circ \rightarrow B$

Terms

- Type judgements are of the form:

$$\Gamma \mid \Delta \mid \Sigma \vdash t : A$$

Terms

- Type judgements are of the form:

$$\Gamma \mid \Delta \mid \Sigma \vdash t : A$$

- Fixed point combinator: $\text{fix}_X : (X \rightarrow X) \rightarrow X$

Terms

- Type judgements are of the form:

$$\Gamma \mid \Delta \mid \Sigma \vdash t : A$$

- Fixed point combinator: $\text{fix}_X : (X \rightarrow X) \rightarrow X$
- Uniformity principle: if $h : X \circ \rightarrow Y$ then

$$h \circ f = g \circ h \Rightarrow h(\text{fix}_X f) = \text{fix}_Y g$$

Terms

- Type judgements are of the form:

$$\Gamma \mid \Delta \mid \Sigma \vdash t : A$$

- Fixed point combinator: $\text{fix}_X : (X \rightarrow X) \rightarrow X$
- Uniformity principle: if $h : X \circ \rightarrow Y$ then

$$h \circ f = g \circ h \Rightarrow h(\text{fix}_X f) = \text{fix}_Y g$$

- Operations are strict, e.g.,

$$\text{lookup}_{\underline{A}} : (\text{nat} \circ \rightarrow \underline{A}) \circ \rightarrow (\text{loc} \circ \rightarrow \underline{A})$$

$$\text{update}_{\underline{A}} : \underline{A} \circ \rightarrow (\text{nat} \circ \rightarrow \text{loc} \circ \rightarrow \underline{A})$$

Recursive types

- Thm: If T is generated by strict operations then \mathbf{Cpo}^T is parametrised algebraically compact wrt. continuous functors
- Consequence: Can add nested recursive value types and nested recursive computation types to type theory
- Reasoning principles for recursive types as in Freyd's / Pitts' work

Conclusions

- Combinations of recursion and effects fall in two categories: Strict and non-strict operations
- Have sketched type theory for case of strict operations
 - Uniformity principle for fixed points
 - Special case: Uniformity of fixed points with respect to operations generating effects
 - Recursive computation types
- In non-strict case we can not model recursive computation types

Typing rules

$$\overline{\Gamma, x: \mathbf{B} \mid - \mid - \vdash x: \mathbf{B}}$$

$$\overline{\Gamma \mid x: \mathbf{B} \mid - \vdash x: \mathbf{B}}$$

$$\overline{\Gamma \mid - \mid x: \underline{\mathbf{A}} \vdash x: \underline{\mathbf{A}}}$$

$$\frac{\Gamma \mid \Delta, x: \mathbf{B} \mid \Sigma \vdash t: \mathbf{C}}{\Gamma \mid \Delta \mid \Sigma \vdash \lambda x: \mathbf{B}. t: \mathbf{B} \circ \rightarrow \mathbf{C}}$$

$$\frac{\Gamma \mid \Delta \mid \Sigma \vdash s: \mathbf{B} \circ \rightarrow \mathbf{C} \quad \Gamma \mid \Delta' \mid - \vdash t: \mathbf{B}}{\Gamma \mid \Delta, \Delta' \mid \Sigma \vdash s(t): \mathbf{C}} \quad \text{dom}(\Delta) \cap \text{dom}(\Delta') = \emptyset$$

Typing rules

$$\frac{\Gamma \mid \Delta \mid x : \underline{A} \vdash t : \underline{B}}{\Gamma \mid \Delta \mid - \vdash \lambda^\circ x : \underline{B}. t : \underline{A} \multimap \underline{B}}$$

$$\frac{\Gamma \mid \Delta \mid - \vdash s : \underline{A} \multimap \underline{B} \quad \Gamma \mid \Delta' \mid \Sigma \vdash t : \underline{A}}{\Gamma \mid \Delta, \Delta' \mid \Sigma \vdash s(t) : \underline{B}} \quad \text{dom}(\Delta) \cap \text{dom}(\Delta') = \emptyset$$

$$\frac{\Gamma \mid - \mid - \vdash t : \underline{B}}{\Gamma \mid - \mid - \vdash !t : \underline{LB}}$$

$$\frac{\Gamma \mid \Delta \mid - \vdash t : \underline{LB} \quad \Gamma, x : \underline{B} \mid \Delta' \mid \Sigma \vdash u : \underline{A}}{\Gamma \mid \Delta, \Delta' \mid \Sigma \vdash \text{let } !x \text{ be } t \text{ in } u : \underline{A}}$$

Typing rules

$$\frac{\Gamma \mid \Delta \mid - \vdash t : \mathbf{B}}{\Gamma \mid \Delta \mid - \vdash [t] : \mathbf{FB}}$$

$$\frac{\Gamma \mid \Delta \mid \Sigma \vdash t : \mathbf{FB} \quad \Gamma \mid \Delta', x : \mathbf{B} \mid - \vdash u : \underline{\underline{\mathbf{A}}}}{\Gamma \mid \Delta, \Delta' \mid \Sigma \vdash \text{let } [x] \text{ be } t \text{ in } u : \underline{\underline{\mathbf{A}}}} \quad \text{dom}(\Delta) \cap \text{dom}(\Delta') = \emptyset$$

Equations

$$(\lambda x:A. t)(u) = t[u/x]$$

$$\lambda x:A. t(x) = t$$

if $t: A \circ \rightarrow B$ and $x \notin \text{FV}(t)$

$$(\lambda^\circ x:\underline{A}. t)(u) = t[u/x]$$

$$\lambda^\circ x:\underline{A}. t(x) = t$$

if $t: \underline{A} \multimap \underline{B}$ and $x \notin \text{FV}(t)$

Equations

$$\frac{\Gamma \mid - \mid - \vdash t : B \quad \Gamma, x : B \mid \Delta \mid \Sigma \vdash u : A}{\Gamma \mid \Delta \mid \Sigma \vdash \text{let } !x \text{ be } !t \text{ in } u = u[t/x]}$$

$$\frac{\Gamma \mid \Delta \mid - \vdash t : LA \quad \Gamma \mid y : LA \mid \Sigma \vdash u : B}{\Gamma \mid \Delta \mid \Sigma \vdash \text{let } !x \text{ be } t \text{ in } u[!x/y] = u[t / y]}$$

Equations

$$\frac{\Gamma \mid \Delta \mid - \vdash t : \mathbf{B} \quad \Gamma \mid \Delta', x : \mathbf{B} \mid - \vdash u : \underline{\mathbf{A}}}{\Gamma \mid \Delta, \Delta' \mid - \vdash \text{let } [x] \text{ be } [t] \text{ in } u = u[t/x]}$$

$$\frac{\Gamma \mid \Delta \mid \Sigma \vdash t : \mathbf{FA} \quad \Gamma \mid \Delta' \mid y : \mathbf{FA} \vdash u : \underline{\mathbf{B}}}{\Gamma \mid \Delta, \Delta' \mid \Sigma \vdash \text{let } [x] \text{ be } t \text{ in } u[[x] / y] = u[t / y]}$$