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

\[ A, B ::= FA \mid A \to B \mid A \twoheadrightarrow B \]

- Computation types

\[ A, B ::= FA \mid A \to B \]

- Computation types \( \subseteq \) Value types
Terms

Terms in context

\[ \Gamma \mid - \vdash t : A \quad \Gamma \mid x : B \vdash t : A \]
Terms

Terms in context

\[ \Gamma |- t : A \quad \Gamma | x : B |- t : A. \]

Terms

\[ t, s ::= x | \lambda x : A. t | s(t) | \lambda^O x : A. t \]
Typing rules

\[ \Gamma, x: A \vdash x: A \quad \Gamma \vdash_\Delta x: A \]

\[ \Gamma, x: A \vdash_\Delta t: B \quad \Gamma \vdash_\Delta \lambda x: A. t: A \to B \]

\[ \Gamma \vdash_\Delta s: A \to B \quad \Gamma \vdash t: A \]

\[ \Gamma \vdash_\Delta s(t): B \]

\[ \Gamma \vdash_\Delta s: A \to B \quad \Gamma \vdash_\Delta t: A \]

\[ \Gamma \vdash_\Delta s(t): B \]
Typing rules, monadic type

\[
\begin{align*}
\Gamma | \vdash t : B \\
\Gamma | \vdash !t : FB
\end{align*}
\]

\[
\begin{align*}
\Gamma | \Delta \vdash t : FB \\
\Gamma, x : B | \vdash u : A
\end{align*}
\]

\[
\Gamma | \Delta \vdash \text{let } !x \text{ be } t \text{ in } u : A
\]
Algebraic operations

- Specialise the theory to printing by adding

\[ \text{print}_a : A \rightarrow A \]

for each computation type \( A \) and each \( a \) in alphabet \( A \)
Algebraic operations

- Specialise the theory to printing by adding
  \[ \text{print}_a : A \rightarrow A \]
  for each computation type \( A \) and each \( a \) in alphabet \( A \)

- Or to global state by adding
  \[ \text{lookup}_A : (\text{nat} \rightarrow A) \rightarrow \text{loc} \rightarrow A \]
  \[ \text{update}_A : A \rightarrow \text{nat} \rightarrow \text{loc} \rightarrow A \]
  for each computation type \( A \)
Maps commuting with operations

Maps of homomorphism type $f : \mathbf{A} \to \mathbf{B}$ commute with operations

Examples:

- $\text{print}_a(f(x)) = f(\text{print}_a(x))$
- $\text{lookup}_{\mathbf{B}^l}(\lambda x : \text{nat}. f(t)) = f(\text{lookup}_{\mathbf{A}^l}(\lambda x : \text{nat}. t))$
Models

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

\[
\begin{align*}
\mathcal{V}^T & \xrightarrow{F} \mathcal{V} \\
\mathcal{V} & \xleftarrow{U} \downarrow
\end{align*}
\]

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

\[ \begin{array}{c}
\mathcal{C} \\
\mathcal{F} \downarrow \downarrow \\
\mathcal{V}
\end{array} \]

\[ \begin{array}{c}
\mathcal{V} \\
\mathcal{F} \langle \rightarrow \rangle \mathcal{U}
\end{array} \]

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

\[
A^* \times (-) \xrightarrow{\cdot} U \\
\text{Set}
\]

PA cat. of sets with left \( A^* \) action \((X, \cdot : A^* \times X \to X)\)
Example: Printing

\[ \begin{align*}
\text{PA} \\
A^* \times (-) \xrightarrow{-} U \\
\text{Set}
\end{align*} \]

- **PA cat. of sets with left** \( A^* \) **action** \((X, \cdot : A^* \times X \to X)\)
- → interpreted as ordinary maps in sets, → as homomorphisms
Example: Printing

\[
\begin{array}{c}
\text{PA} \\
\downarrow \quad \downarrow \\
A^* \times (-) \quad U \\
\text{Set}
\end{array}
\]

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

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

\[ A, B ::= \ X | A \to B | \forall X. A | X | A \to B | \forall X. A \]

- Computation types

\[ A, B ::= \ A \to B | \forall X. A | X | \forall X. A \]

- \( F \) can be encoded using parametricity:

\[ FA = \forall X. (A \to X) \to X. \]
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

\[ \mu^\circ X. \mathsf{A} \overset{\text{def}}{=} \forall X. (\mathsf{A} \rightarrow X) \rightarrow X \quad (X \text{ +ve in } \mathsf{A}) \]
Inductive and coinductive types in PE

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

\[
\mu^\circ X. A \overset{\text{def}}{=} \forall X. (A \rightarrow X) \rightarrow X \quad (X \text{ +ve in } A)
\]

- 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

\[ \mu X. A \overset{\text{def}}{=} \forall X. (A \rightarrow X) \rightarrow X \quad (X \text{ +ve in } A) \]

- 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
  \[ \mu^\circ X. ~ A \overset{\text{def}}{=} \forall X. (A \rightarrow X) \rightarrow X \]  \[(X \text{ +ve in } A)\]

- 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 $\text{fix}_X : (X \to X) \to X$
- 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 $\text{fix}_X: (X \to X) \to X$
- 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 $h(\bot) = \bot$)

- This suggests three notions of maps: $\to$, $\rightsquigarrow$ as above and $\multimap$ for strict maps
Examples of monads

Models are monads on $\mathbb{CPO}$ (complete partial orders)
Examples of monads

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

- Examples:
  - State and recursion: $TX = (S \times X)^S_\perp$ (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: $TX = (A^* \times X)_\perp$
Examples of monads

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

Thm: $U$ has a left adjoint
Pointed homsets

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

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

is a cpo.

Is it a pointed cpo?
Pointed homsets

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

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

is a cpo.

Is it a pointed cpo?

Thm: For $T$ given by operations, $\text{Cpo}^T(\xi, \xi')$ contains $\lambda x: X. \bot$ for all $\xi, \xi'$ iff all operations are strict.
Pointed homsets

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

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

is a cpo.

- Is it a pointed cpo?

- Thm: For $T$ given by operations, $\text{Cpo}^T(\xi, \xi')$ contains $\lambda x: X. \bot$ for all $\xi, \xi'$ iff all operations are strict.

- Condition holds for state: $TX = (S \times X)^S$ and commutative combination of print and recursion $TX = (A^* \times X)^\bot$
Pointed homsets

- Given a pair of algebras $\xi: TX \to X, \xi': TY \to Y$ the homset
  \[ \text{Cpo}^T(\xi, \xi') \]
  is a cpo.

- Is it a pointed cpo?

- Thm: For $T$ given by operations, $\text{Cpo}^T(\xi, \xi')$ contains $\lambda x: X. \bot$ for all $\xi, \xi'$ iff all operations are strict.

  - Condition holds for state: $TX = (S \times X)^S \bot$ and commutative combination of print and recursion $TX = (A^* \times X) \bot$

  - Condition does not hold for usual combination of print and recursion $TX = \mu Z. (A \times Z + X) \bot$
Type theory for strict operations

\[
\begin{align*}
\text{Cpo}^T \\
F \downarrow \uparrow U \\
\text{Cppo}_\perp
\end{align*}
\]
Type theory for strict operations

\[
\begin{align*}
&\text{Cpo}^T \\
&F \downarrow \quad U \\
&\text{Cppo}_\bot
\end{align*}
\]

- Value types in \(\text{Cppo}_\bot\), computation types in \(\text{Cpo}^T\)

- Define \(A \to B = \text{LA} \circ \to B\)
Terms

Type judgements are of the form:

\[ \Gamma \mid \Delta \mid \Sigma \vdash t : A \]
Terms

- Type judgements are of the form:
  \[ \Gamma | \Delta \mid \Sigma \vdash t : A \]

- 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:

  \[ \Gamma | \Delta | \Sigma \vdash t : A \]

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

- Uniformity principle: if \( h : X \circ \to Y \) then

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

- Operations are strict, e.g.,

  \[
  \text{lookup}_A : (\text{nat} \circ \to A) \circ \to (\text{loc} \circ \to A) \\
  \text{update}_A : A \circ \to (\text{nat} \circ \to \text{loc} \circ \to A)
  \]
Recursive types

Thm: If $T$ is generated by strict operations then $\mathsf{Cpo}^T$ 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 cannot model recursive computation types.
Typing rules

\[
\Gamma, \ x: B \vdash x: B
\]

\[
\Gamma \vdash x: B \vdash x: B \quad \Gamma \vdash x: A \vdash x: A
\]

\[
\Gamma \vdash \Delta, x: B \vdash \Sigma \vdash t: C
\]

\[
\Gamma \vdash \Delta \vdash \Sigma \vdash \lambda x: B. t: B \rightarrowto C
\]

\[
\Gamma \vdash \Delta \vdash \Sigma \vdash s: B \rightarrowto C \quad \Gamma \vdash \Delta' \vdash t: B
\]

\[
\Gamma \vdash \Delta, \Delta' \vdash \Sigma \vdash s(t): C \quad \text{dom}(\Delta) \cap \text{dom}(\Delta') = \emptyset
\]
Typing rules

\[
\frac{\Gamma \mid \Delta \mid x : A \vdash t : B}{\Gamma \mid \Delta \mid - \vdash \lambda^x : B. \, t : A \to B}
\]

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

\[
\frac{\Gamma \mid - \mid - \vdash t : B}{\Gamma \mid - \mid - \vdash !t : L\, B}
\]

\[
\frac{\Gamma \mid \Delta \mid - \vdash t : L\, B \quad \Gamma, x : B \mid \Delta' \mid \Sigma \vdash u : A}{\Gamma \mid \Delta, \Delta' \mid \Sigma \vdash \text{let } !x \text{ be } t \text{ in } u : A}
\]
Typing rules

$$\Gamma | \Delta | - \vdash t : B$$

$$\Gamma | \Delta | - \vdash [t] : FB$$

$$\Gamma | \Delta | \Sigma \vdash t : FB \quad \Gamma | \Delta', x : B | - \vdash u : A \quad \text{dom}(\Delta) \cap \text{dom}(\Delta') = \emptyset$$

$$\Gamma | \Delta, \Delta' | \Sigma \vdash \text{let } [x] \text{ be } t \text{ in } u : A$$
Equations

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

\[\lambda x: A. \ t(x) = t \quad \text{if } t: A \rightarrow B \text{ and } x \notin \text{FV}(t) \]

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

\[\lambda^\circ x: A. \ t(x) = t \quad \text{if } t: A \rightarrow^\circ B \text{ and } x \notin \text{FV}(t) \]
Equations

\[\begin{align*}
\Gamma &\vdash t : B \quad \Gamma, x : B &\vdash \Delta &\vdash u : A \\
\Gamma &\vdash \Delta &\vdash \Sigma &\vdash \text{let } !x \text{ be } !t \text{ in } u = u[t/x]
\end{align*}\]

\[\begin{align*}
\Gamma &\vdash \Delta &\vdash t : LA \quad \Gamma &\vdash y : LA &\vdash \Sigma &\vdash u : B \\
\Gamma &\vdash \Delta &\vdash \Sigma &\vdash \text{let } !x \text{ be } t \text{ in } u[!x/y] = u[t / y]
\end{align*}\]
$$\frac{\Gamma | \Delta \vdash t : B \quad \Gamma | \Delta', x : B \vdash u : A}{\Gamma | \Delta, \Delta' \vdash \text{let } [x] \text{ be } [t] \text{ in } u = u[t/x]}$$

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