Generic Programming with Generic Effects via Effect Handling

Christoph Rauch
(joint work with Sergey Goncharov & Lutz Schröder)

FAU Erlangen-Nürnberg

TCS Seminar SS2014

June 17, 2014

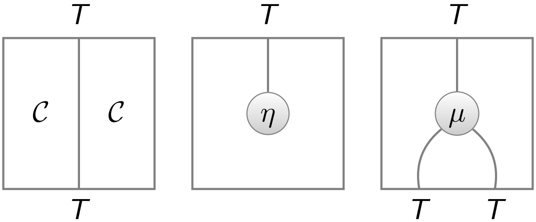
Overview

What we have so far

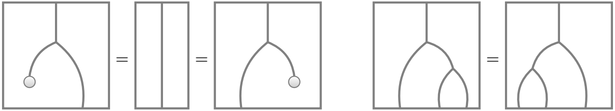
- Metalanguage with generic effects, iteration, free operations, and handlers for those operations.
- Categorical semantics parametric in an Elgot monad.
- Free operations realised using a cofree extension of the monad.
- Iteration interpreted using the fixpoint operator of the Elgot monad which is preserved by this extension.
- Interpretation of handling also employs the fixpoint operator.

(Strong) Monads

As string diagrams

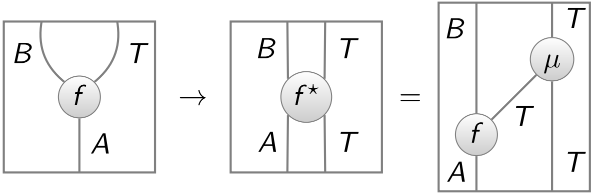


Laws



(Strong) Monads

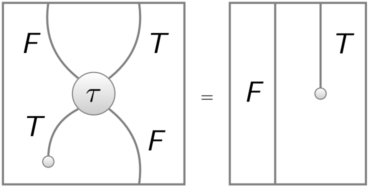
Kleisli Triple

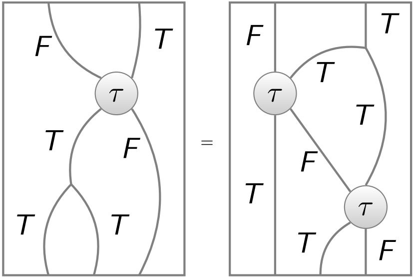


(Strong) Monads

Strength

Let $F = A \times -$. Then, strength is a morphism $\tau : FT \to TF$ with laws





plus a kind of associativity law.

Algebraic Operations

Natural transformations $\alpha_X : (TX)^v \to (TX)^w$ (+ coherence conditions)

Generic Effects

Algebraic operations correspond to generic effects $\alpha_g : w \to Tv$ (Plotkin, Power, 2003)

Examples for Algebraic Effects

$TX = \mu\gamma.\,(X + (O \times \gamma) + \gamma^I)$ (Input/Output)

- $read_X : (TX)^I \to TX$, equivalently $read^g : 1 \to TI \cong TI$.
- $write_X : TX \to (TX)^O$, equivalently $write^g : O \to T1$.

$TX = X + E$ (Exceptions)

- E nullary operations $raise_e$ for $e \in E$, equivalently $raise^g_e : T0$.
- Exception handling is not algebraic!

A Monad for Underspecified Algebraic Operations

Cofree extension

For a monad $\mathbb{T}$ and any $a, b \in |\mathcal{C}|$, let $\mathbb{T}_a^b$ be given by

$$T_a^b X = \nu\gamma\,.\,T(X + a \times \gamma^b)$$

A Monad for Underspecified Algebraic Operations (cont'd)

- $T'(X, A) = T(X + a \times A^b)$ is a *parameterized Monad* in the sense of Uustalu, 2003.
- $\Rightarrow \nu\gamma \,.\, T(X + a \times \gamma^b)$ carries a monad structure.
- Indeed, $\mathbb{T}_a^b$ is a strong monad.

Monad structure

out is the final coalgebra morphism $T_a^b X \to T(X + a \times (T_a^b X)^b)$.

$$\begin{aligned} \mathsf{out} \circ \eta^\nu &= \eta \circ \mathsf{inj}_1, \\ \mathsf{out} \circ f^\S &= [\mathsf{out} \circ f, \eta \circ \mathsf{inj}_2 \circ a \times (f^\S)^b]^\star \circ \mathsf{out}, \\ \mathsf{out} \circ \tau^\nu &= T(\mathsf{id} + a \times (\tau^\nu)^b) \circ T\delta \circ \tau \circ (\mathsf{id} \times \mathsf{out}) \end{aligned}$$

where $\delta : X \times (Y + a \times (T_a^b Y)^b) \to (X \times Y) + a \times (X \times T_a^b Y)^b$.

ω-continuous Monads

Definition

An ω-continuous monad consists of a monad $\mathbb{T}$ and an enrichment of the Kleisli category $\mathcal{C}_{\mathbb{T}}$ of $\mathbb{T}$ over the category **Cppo** of complete partial orders with bottom and (nonstrict) continuous maps, satisfying the following conditions.

- Strength is continuous and respects the bottom element: $\tau\langle \mathrm{id}, \bigsqcup_i f_i\rangle = \bigsqcup_i \tau\langle \mathrm{id}, f_i\rangle$, $\tau\langle \mathrm{id}, \bot\rangle = \bot$;
- Copairing is continuous in both arguments: $\left[\bigsqcup_i f_i, \bigsqcup_i g_i\right] = \bigsqcup_i [f_i, g_i]$.

ω-continuous Monads

Definition

An ω-continuous monad consists of a monad $\mathbb{T}$ and an enrichment of the Kleisli category $\mathcal{C}_{\mathbb{T}}$ of $\mathbb{T}$ over the category **Cppo** of complete partial orders with bottom and (nonstrict) continuous maps, satisfying the following conditions.

- Strength is continuous and respects the bottom element: $\tau\langle \mathrm{id}, \bigsqcup_i f_i\rangle = \bigsqcup_i \tau\langle \mathrm{id}, f_i\rangle$, $\tau\langle \mathrm{id}, \bot\rangle = \bot$;
- Copairing is continuous in both arguments: $\left[\bigsqcup_i f_i, \bigsqcup_i g_i\right] = \bigsqcup_i [f_i, g_i]$.

Cofree extension

$\mathbb{T}_a^b$ is not ω-continuous!

Elgot Monads

Definition

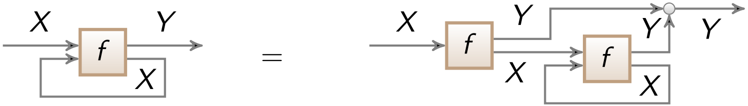
A monad $\mathbb{T}$ over $\mathcal{C}$ is an *Elgot monad* if it possesses an operator $-^{\dagger}$, sending any $f : X \to T(Y + X)$ to $f^{\dagger} : X \to TY$ satisfying the following conditions:

Axioms of Elgot Monads

Definition

A monad $\mathbb{T}$ over $\mathcal{C}$ is an *Elgot monad* if it possesses an operator $-^\dagger$, sending any $f : X \to T(Y + X)$ to $f^\dagger : X \to TY$ satisfying the following conditions:

Unfolding: $f^\dagger = [\eta, f^\dagger]^\star \circ f$

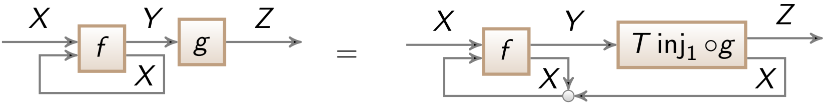


Axioms of Elgot Monads

Definition

A monad $\mathbb{T}$ over $\mathcal{C}$ is an *Elgot monad* if it possesses an operator $-^\dagger$, sending any $f : X \to T(Y + X)$ to $f^\dagger : X \to TY$ satisfying the following conditions:

Naturality: $g^\star \circ f^\dagger = ([T\,\mathsf{inj}_1 \circ g, \eta \circ \mathsf{inj}_2]^\star \circ f)^\dagger$

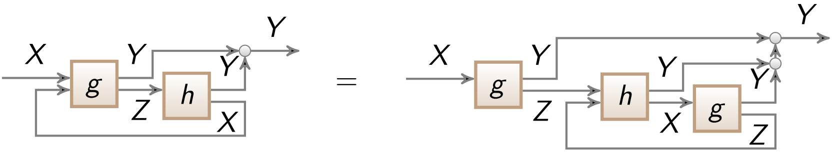


Axioms of Elgot Monads

Definition

A monad $\mathbb{T}$ over $\mathcal{C}$ is an *Elgot monad* if it possesses an operator $-^\dagger$, sending any $f : X \to T(Y + X)$ to $f^\dagger : X \to TY$ satisfying the following conditions:

Dinaturality: $([\eta \circ \mathsf{inj}_1, h]^\star \circ g)^\dagger = [\eta, ([\eta \circ \mathsf{inj}_1, g]^\star \circ h)^\dagger]^\star \circ g$

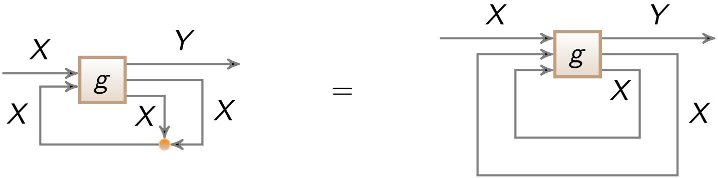


Axioms of Elgot Monads

Definition

A monad $\mathbb{T}$ over $\mathcal{C}$ is an *Elgot monad* if it possesses an operator $-^\dagger$, sending any $f : X \to T(Y + X)$ to $f^\dagger : X \to TY$ satisfying the following conditions:

Codiagonal: $(T(\mathrm{id} + \nabla) \circ g)^\dagger = (g^\dagger)^\dagger$

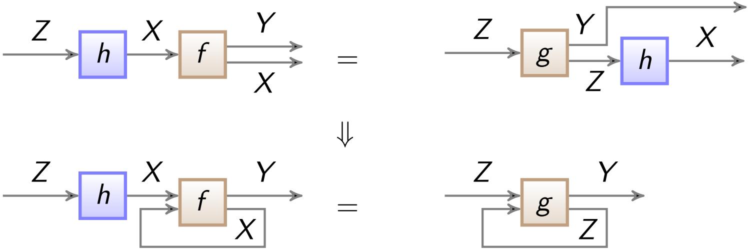


Axioms of Elgot Monads

Definition

A monad $\mathbb{T}$ over $\mathcal{C}$ is an *Elgot monad* if it possesses an operator $-^\dagger$, sending any $f : X \to T(Y + X)$ to $f^\dagger : X \to TY$ satisfying the following conditions:

Uniformity: $f \circ h = T(\mathrm{id} + h) \circ g$ implies $f^\dagger \circ h = g^\dagger$



Elgot Monads

- Any ω-continuous monad is an Elgot monad (Bloom, Ésik).
- $\mathbb{T}_a^b$ is also a (strong) Elgot monad!
- Iteration on $\mathbb{T}_a^b$ reduces to iteration on $\mathbb{T}$.
- Strength of the involved monads allows us to define

$$f^{\ddagger} = \big(T((\mathsf{pr}_2 + \mathsf{id}) \circ \delta_{Z,Y,X}) \circ \tau_{Z,Y+X} \circ \langle \mathsf{pr}_1, f\rangle\big)^{\dagger},$$

i.e. an operator $-^{\ddagger}$ sending any $f : Z \times X \to T(Y + X)$ to $f^{\ddagger} : Z \times X \to TY$.

Metalanguage for side-effecting computations

Features:

- Grammar of types (separating *value*, *computational* and *predicate* types)

Metalanguage for side-effecting computations

Features:

- Grammar of types (separating *value*, *computational* and *predicate* types)

Types:

$$V ::= A \in \mathcal{V} \mid 1 \mid 0 \mid V \times V \mid V + V \qquad W ::= V \mid V \to W$$

Metalanguage for side-effecting computations

Features:

- Grammar of types (separating *value*, *computational* and *predicate* types)
- Judgements $\Gamma \vdash_v t : A$ and $\Delta \mid \Gamma \vdash_c p : B$, where A is value type, B is computation type, Γ is a context of type-variable-pairs, and Δ is an *operation context* of variables $f_i : A_i \to B_i$ (both A_i, B_i value types)

Metalanguage for side-effecting computations

Features:

- Grammar of types (separating *value*, *computational* and *predicate* types)
- Judgements $\Gamma \vdash_v t : A$ and $\Delta \mid \Gamma \vdash_c p : B$, where A is value type, B is computation type, Γ is a context of type-variable-pairs, and Δ is an *operation context* of variables $f_i : A_i \to B_i$ (both A_i, B_i value types)
- (Co)product types and operations: projections/pairing, injections/case

Metalanguage for side-effecting computations

Features:

- Grammar of types (separating *value*, *computational* and *predicate* t…
- J… e, B is … is an o… pes)
- (… in…

Coproducts:

$$\frac{\Gamma \vdash_v t : A}{\Gamma \vdash_v \mathrm{inj}_1\ t : A + B} \qquad \frac{\Gamma \vdash_v t : B}{\Gamma \vdash_v \mathrm{inj}_2\ t : A + B}$$

$$\frac{\Gamma \vdash_v s : A + B \quad \Delta \mid \Gamma, x : A \vdash_{v/c} t : C \quad \Delta \mid \Gamma, y : B \vdash_{v/c} u : C}{\Delta \mid \Gamma \vdash_v \mathrm{case}\ s\ \mathrm{of}\ \mathrm{inj}_1\, x \mapsto t;\ \mathrm{inj}_2\, y \mapsto u : C}$$

Metalanguage for side-effecting computations

Features:

- Grammar of types (separating *value*, *computational* and *predicate* types)
- Judgements $\Gamma \vdash_v t : A$ and $\Delta \mid \Gamma \vdash_c p : B$, where A is value type, B is computation type, Γ is a context of type-variable-pairs, and Δ is an *operation context* of variables $f_i : A_i \to B_i$ (both A_i, B_i value types)
- (Co)product types and operations: projections/pairing, injections/case
- Side-effecting computations

Metalanguage for side-effecting computations

Features:

- Grammar of types (separating *value*, *computational* and *predicate* types)
- Judgements $\Gamma \vdash t : A$ and $\Delta \mid \Gamma \vdash p : B$ where A is value type, B is … is an … pes)
- (… in…
- Side-effecting computations

Programs

$$\frac{\Delta \mid \Gamma \vdash_c p : A \quad \Delta \mid \Gamma, x : A \vdash_c q : B}{\Delta \mid \Gamma \vdash_c \mathsf{do}\ x \leftarrow p; q : B} \qquad \frac{\Gamma \vdash_v p : A}{\Delta \mid \Gamma \vdash_c \mathsf{ret}\ p : A}$$

Metalanguage for side-effecting computations

Features:

- Grammar of types (separating *value*, *computational* and *predicate* types)
- Judgements $\Gamma \vdash_v t : A$ and $\Delta \mid \Gamma \vdash_c p : B$, where A is value type, B is computation type, Γ is a context of type-variable-pairs, and Δ is an *operation context* of variables $f_i : A_i \to B_i$ (both A_i, B_i value types)
- (Co)product types and operations: projections/pairing, injections/case
- Side-effecting computations
- Signatures of function symbols containing operations for the language

Metalanguage for side-effecting computations

Features:

- Grammar of types (separating *value*, *computational* and *predicate* t
- J e, B is is an *o* pes)
- (Co)product types and operations: projections/pairing in
- S
- Signatures of function symbols containing operations for the language

Operations

$$\frac{f : A \to B \in \Sigma_{v/c} \qquad \Gamma \vdash_v t : A}{\Delta \mid \Gamma \vdash_{v/c} f(t) : B}$$

Signature

Σ_v contains *value operations*; Σ_c contains *generic effects*

Metalanguage for side-effecting computations

Features:

- Grammar of types (separating *value*, *computational* and *predicate* types)
- Judgements $\Gamma \vdash_v t : A$ and $\Delta \mid \Gamma \vdash_c p : B$, where A is value type, B is computation type, Γ is a context of type-variable-pairs, and Δ is an *operation context* of variables $f_i : A_i \to B_i$ (both A_i, B_i value types)
- (Co)product types and operations: projections/pairing, injections/case
- Side-effecting computations
- Signatures of function symbols containing operations for the language
- Iteration construction

Metalanguage for side-effecting computations

Features:

- Grammar of types (separating *value*, *computational* and *predicate* types)
- Judgements $\Gamma \vdash \ t : A$ and $\Delta \mid \Gamma \vdash \ p : B$ where A is value type, B is ... is an ... pes)
- (... in ...

Iteration

$$\frac{\Delta \mid \Gamma \ \vdash_c p : B + A \quad \Delta \mid \Gamma, x : A \vdash_c q : B + A}{\Delta \mid \Gamma \ \vdash_c \ \text{iter inj}_2\, x \leftarrow p; q : B}$$

- Side-effecting computations
- Signatures of function symbols containing operations for the language
- Iteration construction

Metalanguage for side-effecting computations

Features:

- Grammar of types (separating *value*, *computational* and *predicate* types)
- Judgements $\Gamma \vdash_v t : A$ and $\Delta \mid \Gamma \vdash_c p : B$, where A is value type, B is computation type, Γ is a context of type-variable-pairs, and Δ is an *operation context* of variables $f_i : A_i \to B_i$ (both A_i, B_i value types)
- (Co)product types and operations: projections/pairing, injections/case
- Side-effecting computations
- Signatures of function symbols containing operations for the language
- Iteration construction
- Effect handling

Metalanguage for side-effecting computations

Featu

- G ... te ty
- Judgements ... $\Delta \mid \Gamma \vdash_c p : B$, where ... is value type, B is ... is an o ... pes)
- (... i
- S
- S ... guage
- It
- Effect handling

Free operations

$$\frac{f : A \to B \in \Delta \quad \Gamma \vdash_v t : A}{\Delta \mid \Gamma \vdash_c f(t) : B}$$

Handling

$$\frac{\Delta, f \mid \Gamma \vdash_c p : C \quad \Delta, f, v : B \to C \mid \Gamma \vdash_c h : A \to C}{\Delta, f : A \to B \mid \Gamma \vdash_c \text{handle } f \text{ in } p \text{ with } v.\, h : C}$$

$$\frac{\Delta, f \mid \Gamma \vdash_c p : C \quad \Delta, f, v : B \to C \mid \Gamma \vdash_c h : A \to C}{\Delta \mid \Gamma \vdash_c \text{handlerec } f \text{ in } p \text{ with } v.\, h : C}$$

Handling

Handlers

$$[\![\Delta \mid \Gamma, f : A \to B, v : B \to C \vdash_c h : A \to C]\!] = \underline{h} : \underline{\Gamma} \to \underline{A} \to T_{\Delta,f,v}\underline{C}$$

$$[\![\Delta \mid \Gamma, f : A \to B]\!] \vdash_c p : C = \underline{p} : \underline{\Gamma} \to T_{\Delta,f}\underline{C}$$

Interpretation

The morphism $\underline{h}$ can be converted to a morphism

$$w : \underline{\Gamma} \to (T_{\Delta,f}\underline{C})^{\underline{B}} \to (T_{\Delta,f}\underline{C})^{\underline{A}}.$$

Using this, we define

$$\Psi_{\underline{A}}^{\underline{B}}(w, \underline{p}) = (\psi_{\underline{A}}^{\underline{B}})^{\ddagger} \circ \langle w, \underline{p} \rangle : \underline{\Gamma} \to T_{\Delta}\underline{C}.$$

Recursive Handling

Interpretation

The morphism $\underline{h}$ can be converted to a morphism

$$w : \underline{\Gamma} \to (T_{\Delta,f}\underline{C})^{\underline{B}} \to (T_{\Delta,f}\underline{C})^{\underline{A}}.$$

Using this, we define

$$\Psi^{\underline{B}}_{\underline{A}}(w, \underline{p}) = (\psi^{\underline{B}}_{\underline{A}})^{\ddagger} \circ \langle w, \underline{p} \rangle : \underline{\Gamma} \to T_{\Delta}\underline{C},$$

where

$$\begin{aligned} \psi^{\underline{B}}_{\underline{A}}(s, t) = {} & T\big(\mathrm{id} + \mathrm{ev}(\mathrm{id} \times s)\big)(\mathrm{out}(t)) \\ & : \big((T_{\Delta,f}\underline{C})^{\underline{B}} \to (T_{\Delta,f}\underline{C})^{\underline{A}}\big) \times T_{\Delta,f}\underline{C} \to T(\underline{C} + T_{\Delta,f}\underline{C}). \end{aligned}$$

This is used to interpret recursive handling.

Shallow Handling

Interpretation

Again, given

$$w : \underline{\Gamma} \to (T_{\Delta,f}\,\underline{C})^{\underline{B}} \to (T_{\Delta,f}\,\underline{C})^{\underline{A}}.$$

we interpret the shallow handling of f in p using

$$[\eta_{\underline{C}}, \mathrm{id}_{T_{\Delta,f}\,\underline{C}}]^\star \circ \mathrm{ext} \circ \psi^{\underline{B}}_{\underline{A}} \circ \langle w, \underline{p} \rangle : \underline{\Gamma} \to T_{\Delta,f}\,\underline{C},$$

where ψ is as before and

$$\mathrm{ext} = \mathrm{out}^{-1} \circ T_{\Delta}\,\mathrm{inj}_1 : \mathbb{T}_{\Delta} \to \mathbb{T}_{\Delta,f:A\to B}.$$

Work in progress

- Program normalisation
- Completeness results
- Handler elimination
- Operational semantics
- Adequacy results
- Hoare-style verification calculus

Program normalisation

A Folklore Conjecture

All programs in the simple programming language not containing handling normalise to a program with only one loop, which is the outermost part of the term, i.e. to a program of the form

$$\mathsf{iter}\ \mathsf{inj}_2\, x \leftarrow p; q,$$

where p and q are loop-free programs. We call this *loop normal form*.

Axioms

- The axioms of Elgot monads can be translated into the metalanguage, e.g. naturality:

$$
\begin{aligned}
& \mathsf{do}\ y \leftarrow (\mathsf{iter}\ \mathsf{inj}_2\, x \leftarrow p; q); r \\
= \ & \mathsf{iter}\ \mathsf{inj}_2\, x \leftarrow (\mathsf{do}\ k \leftarrow p; \mathsf{case}\ k\ \mathsf{of} \\
& \qquad\qquad \mathsf{inj}_1\, y \mapsto (\mathsf{do}\ l \leftarrow r; \mathsf{ret}\, \mathsf{inj}_1\, l); \\
& \qquad\qquad \mathsf{inj}_2\, x \mapsto \mathsf{ret}\, \mathsf{inj}_2\, x); \\
& (\mathsf{do}\ k \leftarrow q; \mathsf{case}\ k\ \mathsf{of} \\
& \qquad\qquad \mathsf{inj}_1\, y \mapsto (\mathsf{do}\ l \leftarrow r; \mathsf{ret}\, \mathsf{inj}_1\, l); \\
& \qquad\qquad \mathsf{inj}_2\, x \mapsto \mathsf{ret}\, \mathsf{inj}_2\, x)
\end{aligned}
$$

- Using these axioms (and perhaps others), we prove rules to distribute loops over other terms.
- This might be indicative for a future completeness result.

Rules

Distributing loops over case

$$
\begin{aligned}
&\mathsf{case}\ c\ \mathsf{of} \\
&\quad \mathsf{inj}_1\ k \mapsto \mathsf{iter}\ \mathsf{inj}_2\ x \leftarrow p; q; \\
&\quad \mathsf{inj}_2\ l \mapsto r \\
=\ &\mathsf{iter}\ \mathsf{inj}_2\ x \leftarrow (\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inj}_1\ k \mapsto p;\ \mathsf{inj}_2\ l \mapsto \mathsf{do}\,\mathsf{inl}\, r); \\
&(\mathsf{case}\ c\ \mathsf{of}\ \mathsf{inj}_1\ k \mapsto q;\ \mathsf{inj}_2\ l \mapsto \mathsf{do}\,\mathsf{inl}\, r),
\end{aligned}
$$

Rules

Distributing loops over do

- Recall that

$$f^{\ddagger} = \big(T((\mathsf{pr}_2 + \mathsf{id}) \circ \delta) \circ \tau \circ \langle \mathsf{pr}_1, f \rangle\big)^{\dagger}.$$

- For a morphism $f : \Gamma \times A \times B \to T(C + B)$ we can define $g : \Gamma \times A \times B \to T(C + A \times B)$ as

$$g = T((\mathsf{pr}_2 + \mathsf{id}) \circ \delta) \circ \tau \circ \langle \mathsf{pr}_2, f \rangle.$$

- Then, $f^{\ddagger} = g^{\ddagger}$ and we get a rule for distributing iter over do.

Rules

Distributing loops over do

$$
\begin{aligned}
&\mathsf{do}\ x \leftarrow p; (\mathsf{iter}\ \mathsf{inj}_2\, y \leftarrow q; r) \\
= &\,\mathsf{iter}\ \mathsf{inj}_2\, y' \leftarrow \big(\mathsf{do}\ x \leftarrow p; z \leftarrow q; (\mathsf{case}\ z\ \mathsf{of} \\
&\quad \mathsf{inj}_1\, k \mapsto \mathsf{ret}\, \mathsf{inj}_1\, k; \\
&\quad \mathsf{inj}_2\, l \mapsto \mathsf{ret}\, \mathsf{inj}_2(l, x))\big); \\
&\big(\mathsf{do}\ z' \leftarrow r[\mathsf{pr}_1\, y'/y, \mathsf{pr}_2\, y'/x]; (\mathsf{case}\ z'\ \mathsf{of} \\
&\quad \mathsf{inj}_1\, k \mapsto \mathsf{ret}\, \mathsf{inj}_1\, k; \\
&\quad \mathsf{inj}_2\, l \mapsto \mathsf{ret}\, \mathsf{inj}_2(l, \mathsf{pr}_2\, y'))\big)
\end{aligned}
$$

Rules

Collapsing nested loops

- The main tool for this task is the codiagonal axiom

$$
\begin{aligned}
& \mathrm{iter}\ \mathrm{inj}_2\, x \leftarrow p; (\mathrm{iter}\ \mathrm{inj}_2\, y \leftarrow \mathrm{ret}\, \mathrm{inj}_2\, x; q) \\
= {} & \mathrm{iter}\ \mathrm{inj}_2\, x \leftarrow p; (\mathrm{do}\ k \leftarrow q[x/y]; (\mathrm{case}\ k\ \mathrm{of} \\
& \qquad \mathrm{inj}_1\, l \mapsto \mathrm{ret}\, l; \\
& \qquad \mathrm{inj}_2\, y \mapsto \mathrm{ret}\, \mathrm{inj}_2\, y)).
\end{aligned}
$$

Rules

Collapsing nested loops

- The main tool for this task is the codiagonal axiom

$$\begin{aligned} & \mathsf{iter}\ \mathsf{inj}_2\, x \leftarrow p; (\mathsf{iter}\ \mathsf{inj}_2\, y \leftarrow \mathsf{ret}\,\mathsf{inj}_2\, x; q) \\ = {} & \mathsf{iter}\ \mathsf{inj}_2\, x \leftarrow p; (\mathsf{do}\ k \leftarrow q[x/y]; (\mathsf{case}\ k\ \mathsf{of} \\ & \qquad \mathsf{inj}_1\ l \mapsto \mathsf{ret}\ l; \\ & \qquad \mathsf{inj}_2\, y \mapsto \mathsf{ret}\,\mathsf{inj}_2\, y)). \end{aligned}$$

- To apply it, we need $\Delta \mid \Gamma, x : A, y : A \vdash_c q : C + A + A$.

Rules

Collapsing nested loops

- The main tool for this task is the codiagonal axiom

$$\begin{aligned} & \mathsf{iter}\ \mathsf{inj}_2\, x \leftarrow p; (\mathsf{iter}\ \mathsf{inj}_2\, y \leftarrow \mathsf{ret}\ \mathsf{inj}_2\, x; q) \\ = \ & \mathsf{iter}\ \mathsf{inj}_2\, x \leftarrow p; (\mathsf{do}\ k \leftarrow q[x/y]; (\mathsf{case}\ k\ \mathsf{of} \\ & \qquad \mathsf{inj}_1\, l \mapsto \mathsf{ret}\ l; \\ & \qquad \mathsf{inj}_2\, y \mapsto \mathsf{ret}\ \mathsf{inj}_2\, y)). \end{aligned}$$

- To apply it, we need $\Delta \mid \Gamma, x : A, y : A \vdash_c q : C + A + A$.
- General terms $\Delta \mid \Gamma, x : B, y : A \vdash_c r : C + B + A$ therefore need to be converted to terms $\Delta \mid \Gamma, x : B + A, y : B + A \vdash_c r' : C + (B + A) + (B + A)$.

Rules

Collapsing nested loops

- The main tool for this task is the codiagonal axiom

$$
\begin{aligned}
& \mathsf{iter}\ \mathsf{inj}_2\, x \leftarrow p; (\mathsf{iter}\ \mathsf{inj}_2\, y \leftarrow \mathsf{ret}\, \mathsf{inj}_2\, x; q) \\
= {} & \mathsf{iter}\ \mathsf{inj}_2\, x \leftarrow p; (\mathsf{do}\ k \leftarrow q[x/y]; (\mathsf{case}\ k\ \mathsf{of} \\
& \qquad\qquad\qquad \mathsf{inj}_1\, l \mapsto \mathsf{ret}\, l; \\
& \qquad\qquad\qquad \mathsf{inj}_2\, y \mapsto \mathsf{ret}\, \mathsf{inj}_2\, y)).
\end{aligned}
$$

- To apply it, we need $\Delta \mid \Gamma, x : A, y : A \vdash_c q : C + A + A$.
- General terms $\Delta \mid \Gamma, x : B, y : A \vdash_c r : C + B + A$ therefore need to be converted to terms $\Delta \mid \Gamma, x : B + A, y : B + A \vdash_c r' : C + (B + A) + (B + A)$.
- The initialisation part of the inner loop needs to be $\mathsf{ret}\, \mathsf{inj}_2\, x$. Any program residing there needs to be shifted into r' as well.

Ongoing work

- Find a set of necessary and sufficient axioms in the metalanguage to prove derivability of the rules.
- Show that loop-free programs also have a normal form (nested case constructions).
- Show completeness w.r.t. the categorical semantics.

Mid-Step Semantics

- Small-step semantics cannot work in our setting due to atomicity of the operations in e.g. Σ_c:

$$\text{case } \mathit{toss} \text{ of } \mathrm{inj}_1 \star \mapsto p;\ \mathrm{inj}_2 \star \mapsto q.$$

- Big-step semantics too coarse: handle f in p with $v.\,h$ does not need full evaluation of p.
- Therefore, we design a *mid*-step semantics with rules e.g.

$$\frac{p \Rightarrow \mathrm{ret}\ t}{\mathrm{handle}\ f\ \mathrm{in}\ p\ \mathrm{with}\ v.\,h \Rightarrow \mathrm{ret}\ t}$$

$$\frac{p \Rightarrow \mathrm{do}\ x \xleftarrow{i} g(s);\, p'}{\mathrm{handle}\ f\ \mathrm{in}\ p\ \mathrm{with}\ v.\,h \Rightarrow \mathrm{do}\ x \xleftarrow{i} g(s);\, \mathrm{handle}\ f\ \mathrm{in}\ p'\ \mathrm{with}\ v.\,h}$$

References

S. Goncharov, L. Schröder, C. Rauch
(Co-)Algebraic Foundations for Effect Handling and Iteration (submitted)

S. Goncharov, L. Schröder
A Relatively Complete Generic Hoare Logic for Order-Enriched Effects (2013)

G. Plotkin, M. Pretnar
Handling Algebraic Effects (2013)

A. Simpson, G. Plotkin
Complete Axioms for Categorical Fixed-Point Operators (2000)

T. Uustalu
Generalizing Substitution (2003)