Kleene Monads: Handling Iteration in a Framework of Generic Effects

Sergey Goncharov, Lutz Schröder, Till Mossakowski

September 23, 2009

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Sichere Kognitive Systeme

Introduction

How to do real world programming with effects?

- Moggi's computational monads (no control).
- Kozen's calculus for Kleene algebras (no effects).
- Control + effects = Kleene monads!

Issues:

- Axiomatization.
- Soundness and completeness.
- Decidability/undecidability.

Introduction

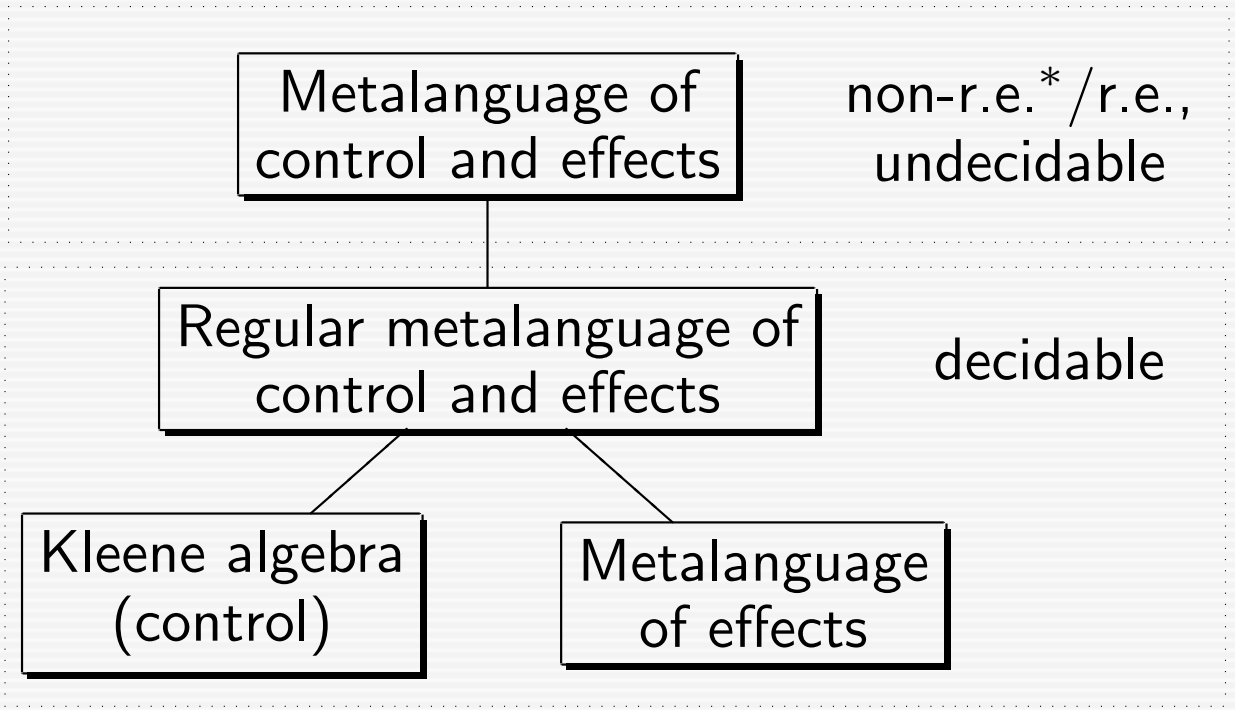
How to do real world programming with effects?

- Moggi's computational monads (no control).
- Kozen's calculus for Kleene algebras (no effects).
- Control + effects = Kleene monads!

Issues:

- Axiomatization.
- Soundness and completeness.
- Decidability/undecidability.

Overview of languages



*: Over continuous Kleene monads

Monads for computation

Strong monad $\mathbb{T}$: Underlying category $\mathcal{C}$, endofunctor $T : \mathcal{C} \to \mathcal{C}$, unit: $\eta : Id \to T$, multiplication $\mu : T^2 \to T$, plus strength: $\tau_{A,B} : A \times TB \to T(A \times B)$.

Metalanguage of effects:

- $Type_W ::= W \mid 1 \mid Type_W \times Type_W \mid T(Type_W)$
- Term construction (cartesian operators omitted):

$$\frac{x : A \in \Gamma}{\Gamma \rhd x : A} \qquad \frac{\Gamma \rhd t : A}{\Gamma \rhd f(t) : B}\ (f : A \to B \in \Sigma)$$

$$\frac{\Gamma \rhd t : A}{\Gamma \rhd \mathrm{ret}\, t : TA} \qquad \frac{\Gamma \rhd p : TA \quad \Gamma, x : A \rhd q : TB}{\Gamma \rhd \mathrm{do}\ x \leftarrow p; q : TB}$$

Monads for computation

Strong monad $\mathbb{T}$: Underlying category $\mathcal{C}$, endofunctor $T : \mathcal{C} \to \mathcal{C}$, unit: $\eta : Id \to T$, multiplication $\mu : T^2 \to T$, plus strength: $\tau_{A,B} : A \times TB \to T(A \times B)$.

Metalanguage of effects:

- $Type_W ::= W \mid 1 \mid Type_W \times Type_W \mid T(Type_W)$
- Term construction (cartesian operators omitted):

$$\frac{x : A \in \Gamma}{\Gamma \rhd x : A} \qquad \frac{\Gamma \rhd t : A}{\Gamma \rhd f(t) : B}\ (f : A \to B \in \Sigma)$$

$$\frac{\Gamma \rhd t : A}{\Gamma \rhd \mathsf{ret}\, t : TA} \qquad \frac{\Gamma \rhd p : TA \quad \Gamma, x : A \rhd q : TB}{\Gamma \rhd \mathsf{do}\ x \leftarrow p; q : TB}$$

Examples of computational monads

- *exception monad:* $TA = A + E$,
- *state monad:* $TA = S \to (S \times A)$,
- *powerset monad:* $TA = \mathcal{P}(A)$,
  variations: *countable powerset monad*: $TA = \mathcal{P}_\omega(A)$,
  finite powerset monad: $TA = \mathcal{P}_{fin}(A)$,
- *input/output monad:* $TA = \mu X.(A + (I \to O \times X))$.
  in particular (if $I = O = 1$): *resumption monad*
  $TA = \mu X.(A + X)$,
- *state powerset monad:* $TA = S \to \mathcal{P}(S \times A)$
  (or $TA = S \to \mathcal{P}_\omega(S \times A)$ or $TA = S \to \mathcal{P}_{fin}(S \times A)$),
- *powerset exception monad:* $TA = \mathcal{P}(A + E)$
  (or $TA = \mathcal{P}_\omega(A + E)$, $TA = \mathcal{P}_{fin}(A + E)$).

Metalanguage of effects

Axioms:

- *first unit law:* $(\mathsf{do}\ x \leftarrow p; \mathsf{ret}\, x) = p$,
- *second unit law:* $(\mathsf{do}\ x \leftarrow \mathsf{ret}\, p; q) = q[p/x]$,
- *associativity law:* provided $y \notin \mathit{Vars}(r)$,
  $\mathsf{do}\ x \leftarrow (\mathsf{do}\ y \leftarrow p; q); r = (\mathsf{do}\ y \leftarrow p; x \leftarrow q; r)$.

Facts:

- Program equality is decidable.
- Conditional program equality is undecidable but r.e.

Metalanguage of effects (Example)

Swapping two variables:

```
swap = {c := a; a := b; b :=c}
```

Metalanguage of effects (Example)

Swapping two variables:

$$\mathtt{swap} = \{\mathtt{c} := \mathtt{a};\ \mathtt{a} := \mathtt{b};\ \mathtt{b} := \mathtt{c}\}$$

Axioms: $[\forall\, \mathtt{s},\mathtt{t} \in \{\mathtt{a},\mathtt{b},\mathtt{c}\}]$

$$\{\mathtt{t} := \mathtt{x};\ \mathtt{s} := \mathtt{x}\} = \{\mathtt{t} := \mathtt{x};\ \mathtt{s} := \mathtt{t}\}$$
$$\{\mathtt{s} := \mathtt{x};\ \mathtt{t} := \mathtt{y}\} = \{\mathtt{t} := \mathtt{y};\ \mathtt{s} := \mathtt{x}\}$$
$$\{\mathtt{t} := \mathtt{x};\ \mathtt{t} := \mathtt{y}\} = \{\mathtt{t} := \mathtt{y}\}$$

Metalanguage of effects (Example)

Swapping two variables:

$$\mathtt{swap} = \{\mathtt{c} := \mathtt{a};\ \mathtt{a} := \mathtt{b};\ \mathtt{b} := \mathtt{c}\}$$

Axioms: $[\forall\, \mathtt{s},\mathtt{t} \in \{\mathtt{a},\mathtt{b},\mathtt{c}\}]$

$$\{\mathtt{t} := \mathtt{x};\ \mathtt{s} := \mathtt{x}\} = \{\mathtt{t} := \mathtt{x};\ \mathtt{s} := \mathtt{t}\}$$
$$\{\mathtt{s} := \mathtt{x};\ \mathtt{t} := \mathtt{y}\} = \{\mathtt{t} := \mathtt{y};\ \mathtt{s} := \mathtt{x}\}$$
$$\{\mathtt{t} := \mathtt{x};\ \mathtt{t} := \mathtt{y}\} = \{\mathtt{t} := \mathtt{y}\}$$

Claim:

$$\{\mathtt{a} := \mathtt{x};\ \mathtt{b} := \mathtt{y};\ \mathtt{swap};\ \mathtt{swap}\} = \{\mathtt{a} := \mathtt{x};\ \mathtt{b} := \mathtt{y};\ \mathtt{c} := \mathtt{y}\}$$

Metalanguage of effects (Example)

Swapping two variables:

$\{c := a;\ a := b;\ b := c\}$ → $\mathsf{do}\ x \leftarrow \mathsf{get}_a; \mathsf{put}_c(x); y \leftarrow \mathsf{get}_b; \mathsf{put}_a(y); z \leftarrow \mathsf{get}_c; \mathsf{put}_b(z)$

Axioms: $[\forall s,t \in \{a,b,c\}]$

$$\{t := x;\ s := x\} = \{t := x;\ s := t\}$$
$$\{s := x;\ t := y\} = \{t := y;\ s := x\}$$
$$\{t := x;\ t := y\} = \{t := y\}$$

Claim:

$$\{a := x;\ b := y;\ swap;\ swap\} = \{a := x;\ b := y;\ c := y\}$$

Metalanguage of effects (Example)

Swapping two variables:

$\texttt{swap} = \{\texttt{c} := \texttt{a};\ \texttt{a} := \texttt{b};\ \texttt{b} := \texttt{c}\}$

$$\text{do } x \leftarrow \text{get}_a; \text{put}_c(x); y \leftarrow \text{get}_b; \text{put}_a(y); z \leftarrow \text{get}_c; \text{put}_b(z)$$

Axioms: $[\forall\, \texttt{s},\texttt{t} \in \{\texttt{a},\texttt{b},\texttt{c}\}]$

$$\{\texttt{t} := \texttt{x};\ \texttt{s} := \texttt{x}\} = \{\texttt{t} := \texttt{x};\ \texttt{s} := \texttt{t}\}$$
$$\{\texttt{s} := \texttt{x};\ \texttt{t} := \texttt{y}\} = \{\texttt{t} := \texttt{y};\ \texttt{s} := \texttt{x}\}$$
$$\{\texttt{t} := \texttt{x};\ \texttt{t} := \texttt{y}\} = \{\texttt{t} := \texttt{y}\}$$

$$\text{do } \text{put}_t(x); \text{put}_s(x) = \text{do } \text{put}_t(x); y \leftarrow \text{get}_t; \text{put}_s(y)$$
$$\text{do } \text{put}_s(x); \text{put}_t(y) = \text{do } \text{put}_t(y); \text{put}_s(x)$$
$$\text{do } \text{put}_t(x); \text{put}_t(y) = \text{put}_t(y)$$

Claim:

$$\{\texttt{a} := \texttt{x};\ \texttt{b} := \texttt{y};\ \texttt{swap};\ \texttt{swap}\} = \{\texttt{a} := \texttt{x};\ \texttt{b} := \texttt{y};\ \texttt{c} := \texttt{y}\}$$

Metalanguage of effects (Example)

Swapping two variables:

$\texttt{swap} = \{\texttt{c} := \texttt{a};\ \texttt{a} := \texttt{b};\ \texttt{b} := \texttt{c}\}$

$$\text{do } x \leftarrow \text{get}_a; \text{put}_c(x); y \leftarrow \text{get}_b; \text{put}_a(y); z \leftarrow \text{get}_c; \text{put}_b(z)$$

Axioms: $[\forall\, s,t \in \{a,b,c\}]$

$$\{t := x;\ s := x\} = \{t := x;\ s := t\}$$
$$\{s := x;\ t := y\} = \{t := y;\ s := x\}$$
$$\{t := x;\ t := y\} = \{t := y\}$$

$$\text{do } \text{put}_t(x); \text{put}_s(x) = \text{do } \text{put}_t(x); y \leftarrow \text{get}_t; \text{put}_s(y)$$

$$\text{do } \text{put}_s(x); \text{put}_t(y) = \text{do } \text{put}_t(y); \text{put}_s(x)$$

$$\text{do } \text{put}_t(x); \text{put}_t(y) = \text{put}_t(y)$$

Claim:

$$\{a := x;\ b := y;\ \texttt{swap};\ \texttt{swap}\} = \{a := x;\ b := y;\ c := y\}$$

$$\text{do } \text{put}_a(x); \text{put}_b(y); \texttt{swap}; \texttt{swap} = \text{do } \text{put}_a(x); \text{put}_b(y); \text{put}_c(y)$$

Metalanguage of effects (Example)

Swapping two variables:

$$\texttt{swap} = \{\texttt{c} := \texttt{a};\ \texttt{a} := \texttt{b};\ \texttt{b} := \texttt{c}\}$$

$$\text{do } x \leftarrow \text{get}_a; \text{put}_c(x); y \leftarrow \text{get}_b; \text{put}_a(y); z \leftarrow \text{get}_c; \text{put}_b(z)$$

Axioms: $[\forall\, s,t \in \{a,b,c\}]$

$$\{\texttt{t} := \texttt{x};\ \texttt{s} := \texttt{x}\} = \{\texttt{t} := \texttt{x};\ \texttt{s} := \texttt{t}\}$$
$$\{\texttt{s} := \texttt{x};\ \texttt{t} := \texttt{y}\} = \{\texttt{t} := \texttt{y};\ \texttt{s} := \texttt{x}\}$$
$$\{\texttt{t} := \texttt{x};\ \texttt{t} := \texttt{y}\} = \{\texttt{t} := \texttt{y}\}$$

$$\text{do } \text{put}_t(x); \text{put}_s(x) = \text{do } \text{put}_t(x); y \leftarrow \text{get}_t; \text{put}_s(y)$$

$$\text{do } \text{put}_s(x); \text{put}_t(y) = \text{do } \text{put}_t(y); \text{put}_s(x)$$

$$\text{do } \text{put}_t(x); \text{put}_t(y) = \text{put}_t(y)$$

Claim:

$$\{\texttt{a} := \texttt{x};\ \texttt{b} := \texttt{y};\ \texttt{swap};\ \texttt{swap}\} = \{\texttt{a} := \texttt{x};\ \texttt{b} := \texttt{y};\ \texttt{c} := \texttt{y}\}$$

$$\text{do } \text{put}_a(x); \text{put}_b(y); \texttt{swap}; \texttt{swap} = \text{do } \text{put}_a(x); \text{put}_b(y); \text{put}_c(y)$$

Remarkably $\{a := a\}$ is not provably equal to the identity program!

Additive monads

A monad $\mathbb{T}$ is an additive monad if for any $A, B \in \mathrm{Ob}(\mathcal{C})$, hom-sets $\mathrm{Hom}_{\mathcal{C}_\mathbb{T}}(A, B)$ of the Kleisli category are commutative monoids and Kleisli composition distributes over monoid operators. We refer to the monoid operations by $\varnothing$ (deadlock) and $+$ (choice).

Strong additive monad $\equiv$ Strong $\sqcap$ Additive $\sqcap$ Identities:

$$\tau_{A,B}\langle \mathrm{id}_A, \varnothing \rangle = \varnothing,$$

$$\tau_{A,B}\langle \mathrm{id}_A, f + g \rangle = \tau_{A,B}\langle \mathrm{id}_A, f \rangle + \tau_{A,B}\langle \mathrm{id}_A, g \rangle.$$

Complete axiomatiation of strong additive monads:

$$p + \varnothing = p \qquad p + q = q + p$$

$$p + p = p \qquad p + (q + r) = (p + q) + r$$

$$\text{do } x \leftarrow \varnothing; r = \varnothing \qquad \text{do } x \leftarrow r; \varnothing = \varnothing$$

$$\text{do } x \leftarrow (p + q); r = (\text{do } x \leftarrow p; r) + (\text{do } x \leftarrow q; r)$$

$$\text{do } x \leftarrow r; (p + q) = (\text{do } x \leftarrow r; p) + (\text{do } x \leftarrow r; q)$$

Additive monads

A monad $\mathbb{T}$ is an additive monad if for any $A, B \in \mathrm{Ob}(\mathcal{C})$, hom-sets $\mathrm{Hom}_{\mathcal{C}_\mathbb{T}}(A, B)$ of the Kleisli category are commutative monoids and Kleisli composition distributes over monoid operators. We refer to the monoid operations by $\varnothing$ (deadlock) and $+$ (choice).

Strong additive monad $\equiv$ Strong $\sqcap$ Additive $\sqcap$ Identities:

$$\begin{aligned} \tau_{A,B}\langle \mathrm{id}_A, \varnothing \rangle &= \varnothing, \\ \tau_{A,B}\langle \mathrm{id}_A, f + g \rangle &= \tau_{A,B}\langle \mathrm{id}_A, f \rangle + \tau_{A,B}\langle \mathrm{id}_A, g \rangle. \end{aligned}$$

Complete axiomatiation of strong additive monads:

$$p + \varnothing = p \qquad p + q = q + p$$

$$p + p = p \qquad p + (q + r) = (p + q) + r$$

$$\text{do } x \leftarrow \varnothing; r = \varnothing \qquad \text{do } x \leftarrow r; \varnothing = \varnothing$$

$$\text{do } x \leftarrow (p + q); r = (\text{do } x \leftarrow p; r) + (\text{do } x \leftarrow q; r)$$

$$\text{do } x \leftarrow r; (p + q) = (\text{do } x \leftarrow r; p) + (\text{do } x \leftarrow r; q)$$

Additive monads

A monad $\mathbb{T}$ is an additive monad if for any $A, B \in \mathrm{Ob}(\mathcal{C})$, hom-sets $\mathrm{Hom}_{\mathcal{C}_\mathbb{T}}(A, B)$ of the Kleisli category are commutative monoids and Kleisli composition distributes over monoid operators. We refer to the monoid operations by $\varnothing$ (deadlock) and $+$ (choice).

Strong additive monad $\equiv$ Strong $\sqcap$ Additive $\sqcap$ Identities:

$$\begin{aligned} \tau_{A,B}\langle \mathrm{id}_A, \varnothing \rangle &= \varnothing, \\ \tau_{A,B}\langle \mathrm{id}_A, f + g \rangle &= \tau_{A,B}\langle \mathrm{id}_A, f \rangle + \tau_{A,B}\langle \mathrm{id}_A, g \rangle. \end{aligned}$$

Complete axiomatiation of strong additive monads:

$$p + \varnothing = p \qquad p + q = q + p$$

$$p + p = p \qquad p + (q + r) = (p + q) + r$$

$$\text{do } x \leftarrow \varnothing; r = \varnothing \qquad \text{do } x \leftarrow r; \varnothing = \varnothing$$

$$\text{do } x \leftarrow (p + q); r = (\text{do } x \leftarrow p; r) + (\text{do } x \leftarrow q; r)$$

$$\text{do } x \leftarrow r; (p + q) = (\text{do } x \leftarrow r; p) + (\text{do } x \leftarrow r; q)$$

Kleene monads

A strong Kleene monad is a strong additive monad equipped with the Kleene star constructor $(\mathsf{init}\ \_ \leftarrow \_\ \mathsf{in}\ \_^*)$, subject to the following axiomatization ($y \notin \mathrm{FV}(r)$):

$$\mathsf{init}\ x \leftarrow p\ \mathsf{in}\ q^* = p + \mathsf{do}\ \ x \leftarrow (\mathsf{init}\ x \leftarrow p\ \mathsf{in}\ q^*); q$$

$$\mathsf{init}\ x \leftarrow p\ \mathsf{in}\ q^* = p + \mathsf{init}\ x \leftarrow (\mathsf{do}\ \ x \leftarrow p; q;)\ \mathsf{in}\ q^*$$

$$\mathsf{init}\ x \leftarrow (\mathsf{do}\ \ y \leftarrow p; q)\ \mathsf{in}\ r^* = \mathsf{do}\ \ y \leftarrow p; (\mathsf{init}\ x \leftarrow q\ \mathsf{in}\ r^*)$$

$$\frac{\mathsf{do}\ \ x \leftarrow p; q \leqslant p}{\mathsf{init}\ x \leftarrow p\ \mathsf{in}\ q^* \leqslant p} \qquad \frac{\mathsf{do}\ \ x \leftarrow q[y/x]; r \leqslant r[y/x]}{\mathsf{do}\ \ x \leftarrow (\mathsf{init}\ x \leftarrow p\ \mathsf{in}\ q^*); r \leqslant \mathsf{do}\ \ x \leftarrow p; r}$$

where $p \leqslant q \iff p + q = q$. This extends the metalanguage of effects to the metalanguage of control and effects (MCE).

A strong Kleene monad is continuous if $(\mathsf{init}\ x \leftarrow p\ \mathsf{in}\ q^*)$ is the supremum of $\{\mathsf{do}\ \ x \leftarrow p; x \leftarrow q; \ldots; x \leftarrow q; q\}$.

Kleene monads

A strong Kleene monad is a strong additive monad equipped with the Kleene star constructor $(\mathsf{init}\ \_ \leftarrow \_\ \mathsf{in}\ \_^*)$, subject to the following axiomatization ($y \notin \mathrm{FV}(r)$):

$$\mathsf{init}\ x \leftarrow p\ \mathsf{in}\ q^* = p + \mathsf{do}\ \ x \leftarrow (\mathsf{init}\ x \leftarrow p\ \mathsf{in}\ q^*); q$$

$$\mathsf{init}\ x \leftarrow p\ \mathsf{in}\ q^* = p + \mathsf{init}\ x \leftarrow (\mathsf{do}\ \ x \leftarrow p; q;)\ \mathsf{in}\ q^*$$

$$\mathsf{init}\ x \leftarrow (\mathsf{do}\ \ y \leftarrow p; q)\ \mathsf{in}\ r^* = \mathsf{do}\ \ y \leftarrow p; (\mathsf{init}\ x \leftarrow q\ \mathsf{in}\ r^*)$$

$$\frac{\mathsf{do}\ \ x \leftarrow p; q \leqslant p}{\mathsf{init}\ x \leftarrow p\ \mathsf{in}\ q^* \leqslant p} \qquad \frac{\mathsf{do}\ \ x \leftarrow q[y/x]; r \leqslant r[y/x]}{\mathsf{do}\ \ x \leftarrow (\mathsf{init}\ x \leftarrow p\ \mathsf{in}\ q^*); r \leqslant \mathsf{do}\ \ x \leftarrow p; r}$$

where $p \leqslant q \iff p + q = q$. This extends the metalanguage of effects to the metalanguage of control and effects (MCE).

A strong Kleene monad is continuous if $(\mathsf{init}\ x \leftarrow p\ \mathsf{in}\ q^*)$ is the supremum of $\{\mathsf{do}\ \ x \leftarrow p; x \leftarrow q; \ldots; x \leftarrow q; q\}$.

Kleene monads (Examples)

- The powerset monad and the countable powerset monad (but not the finite powerset monad!) are strong continuous Kleene monads.
- Combining strong continuous Kleene monads with states, input/output and resumptions produces strong continuous Kleene monads.
- But, the powerset exeption monad $(\mathrm{T}A = \mathcal{P}(A + \mathrm{E}))$ is not a Kleene monad! The reason is:

$$\text{do } x \leftarrow (\text{throw } e); \varnothing = (\text{throw } e) \neq \varnothing$$

Kleene monads (Examples)

- The powerset monad and the countable powerset monad (but not the finite powerset monad!) are strong continuous Kleene monads.
- Combining strong continuous Kleene monads with states, input/output and resumptions produces strong continuous Kleene monads.
- ✖ But, the powerset exeption monad ($\mathsf{T}A = \mathcal{P}(A + \mathrm{E})$) is not a Kleene monad! The reason is:

$$\text{do } x \leftarrow (\text{throw } e); \varnothing = (\text{throw } e) \neq \varnothing$$

Metalanguage of control and effects (Example)

Let **push**, **pop** and **empty?** be the usual stack operations, where **empty?** blocks if the state is nonempty (**empty?** $= \varnothing$), and otherwise does nothing (**empty?** $=$ ret $\star$).

Then one can define the **reverse** operation as

$$\begin{aligned} &\mathsf{do}\ \ \mathrm{q} \leftarrow (\mathsf{init}\,\mathrm{p} \leftarrow \mathsf{ret}\,\textbf{empty?}\ \mathsf{in} \\ &\qquad\qquad (\mathsf{do}\ \ \mathrm{x} \leftarrow \textbf{pop}; \mathsf{ret}\ (\mathsf{do}\ \ \mathrm{p}; \textbf{push}\,\mathrm{x}))^*); \mathrm{q} \end{aligned}$$

E.g. for a two-element stack:

$$\begin{aligned} &\textbf{empty?} + \\ &\mathsf{do}\ \ \mathrm{x} \leftarrow \textbf{pop}; \textbf{empty?}; \textbf{push}\,\mathrm{x} + \\ &\mathsf{do}\ \ \mathrm{x} \leftarrow \textbf{pop}; \mathrm{y} \leftarrow \textbf{pop}; \textbf{empty?}; \textbf{push}\,\mathrm{x}; \textbf{push}\,\mathrm{y} + \\ &\mathsf{do}\ \ \mathrm{x} \leftarrow \textbf{pop}; \mathrm{y} \leftarrow \textbf{pop}; \mathrm{z} \leftarrow \textbf{pop}; \ldots \end{aligned}$$

Metalanguage of control and effects (Example)

Let **push**, **pop** and **empty?** be the usual stack operations, where **empty?** blocks if the state is nonempty (**empty?** $= \varnothing$), and otherwise does nothing (**empty?** $=$ ret $\star$).

Then one can define the **reverse** operation as

$$\begin{aligned} &\text{do } q \leftarrow (\text{init } p \leftarrow \text{ret } \textbf{empty?} \text{ in} \\ &\qquad (\text{do } x \leftarrow \textbf{pop}; \text{ret } (\text{do } p; \textbf{push } x))^*); q \end{aligned}$$

E.g. for a two-element stack:

$$\begin{aligned} &\textbf{empty?} + \\ &\text{do } x \leftarrow \textbf{pop}; \textbf{empty?}; \textbf{push } x + \\ &\text{do } x \leftarrow \textbf{pop}; y \leftarrow \textbf{pop}; \textbf{empty?}; \textbf{push } x; \textbf{push } y + \\ &\text{do } x \leftarrow \textbf{pop}; y \leftarrow \textbf{pop}; z \leftarrow \textbf{pop}; \ldots \end{aligned}$$

Metalanguage of control and effects (Example)

Let **push**, **pop** and **empty?** be the usual stack operations, where **empty?** blocks if the state is nonempty (**empty?** $= \varnothing$), and otherwise does nothing (**empty?** $=$ ret $\star$).

Then one can define the **reverse** operation as

$$\begin{aligned} &\text{do } q \leftarrow (\text{init } p \leftarrow \text{ret } \textbf{empty?} \text{ in} \\ &\qquad (\text{do } x \leftarrow \textbf{pop}; \text{ret } (\text{do } p; \textbf{push } x))^*); q \end{aligned}$$

E.g. for a two-element stack:

$$\begin{aligned} &\textbf{empty?} + \\ &\text{do } x \leftarrow \textbf{pop}; \textbf{empty?}; \textbf{push } x + \\ &\text{do } x \leftarrow \textbf{pop}; y \leftarrow \textbf{pop}; \textbf{empty?}; \textbf{push } x; \textbf{push } y + \\ &\text{do } x \leftarrow \textbf{pop}; y \leftarrow \textbf{pop}; z \leftarrow \textbf{pop}; \ldots \end{aligned}$$

Metalanguage of control and effects (Example)

Let **push**, **pop** and **empty?** be the usual stack operations, where **empty?** blocks if the state is nonempty (**empty?** $= \varnothing$), and otherwise does nothing (**empty?** $=$ ret $\star$).

Then one can define the **reverse** operation as

$$\begin{aligned} &\text{do } q \leftarrow (\text{init } p \leftarrow \text{ret } \textbf{empty?} \text{ in} \\ &\qquad (\text{do } x \leftarrow \textbf{pop}; \text{ret } (\text{do } p; \textbf{push } x))^*); q \end{aligned}$$

E.g. for a two-element stack:

$$\begin{aligned} &\textbf{empty?} + \\ &\text{do } x \leftarrow \textbf{pop}; \textbf{empty?}; \textbf{push } x + \\ &\text{do } x \leftarrow \textbf{pop}; y \leftarrow \textbf{pop}; \textbf{empty?}; \textbf{push } x; \textbf{push } y + \\ &\text{do } x \leftarrow \textbf{pop}; y \leftarrow \textbf{pop}; z \leftarrow \textbf{pop}; \ldots \end{aligned}$$

Metalanguage of control and effects (Example)

Let **push**, **pop** and **empty?** be the usual stack operations, where **empty?** blocks if the state is nonempty (**empty?** $= \varnothing$), and otherwise does nothing (**empty?** $=$ ret $\star$).

Then one can define the **reverse** operation as

$$\begin{aligned} &\text{do } q \leftarrow (\text{init } p \leftarrow \text{ret } \textbf{empty?} \text{ in} \\ &\qquad (\text{do } x \leftarrow \textbf{pop}; \text{ret } (\text{do } p; \textbf{push } x))^*); q \end{aligned}$$

E.g. for a two-element stack:

$$\begin{aligned} &\textbf{empty?} + \\ &\text{do } x \leftarrow \textbf{pop}; \textbf{empty?}; \textbf{push } x + \\ &\text{do } x \leftarrow \textbf{pop}; y \leftarrow \textbf{pop}; \textbf{empty?}; \textbf{push } x; \textbf{push } y + \\ &\text{do } x \leftarrow \textbf{pop}; y \leftarrow \textbf{pop}; z \leftarrow \textbf{pop}; \ldots \end{aligned}$$

Metalanguage of control and effects (Example)

Let **push**, **pop** and **empty?** be the usual stack operations, where **empty?** blocks if the state is nonempty (**empty?** $= \varnothing$), and otherwise does nothing (**empty?** $=$ ret $\star$).

Then one can define the **reverse** operation as

$$\begin{aligned} &\text{do } q \leftarrow (\text{init } p \leftarrow \text{ret } \textbf{empty?} \text{ in} \\ &\qquad (\text{do } x \leftarrow \textbf{pop}; \text{ret } (\text{do } p; \textbf{push } x))^*); q \end{aligned}$$

E.g. for a two-element stack:

$$\begin{aligned} &\textbf{empty?} + \\ &\text{do } x \leftarrow \textbf{pop}; \textbf{empty?}; \textbf{push } x + \\ &\text{do } x \leftarrow \textbf{pop}; y \leftarrow \textbf{pop}; \textbf{empty?}; \textbf{push } x; \textbf{push } y + \\ &\text{do } x \leftarrow \textbf{pop}; y \leftarrow \textbf{pop}; z \leftarrow \textbf{pop}; \ldots \end{aligned}$$

Negative results

Restriction. In the remainder we require that for any $f : A \to B \in \Sigma$, A is T-free.

Theorem (Negative result #1). Let p and q be two MCE programs and $\mathfrak{C}$ be the class of all strong Kleene monads. The problem $\mathfrak{C} \models p = q$ is undecidable.

Theorem (Negative result #2). Let p and q be two MCE programs and $\mathfrak{C}_\omega$ be the class of all strong continuous Kleene monads. The problem $\mathfrak{C}_\omega \models p = q$ is non-r.e.

Corollary. There is no complete recursive axiomatization of the equational theory of strong continuous Kleene monads.

Idea of the proof: reduction from Post correspondence problem.

Negative results

Restriction. In the remainder we require that for any $f : A \to B \in \Sigma$, A is T-free.

Theorem (Negative result #1). Let p and q be two MCE programs and $\mathfrak{C}$ be the class of all strong Kleene monads. The problem $\mathfrak{C} \models p = q$ is undecidable.

Theorem (Negative result #2). Let p and q be two MCE programs and $\mathfrak{C}_\omega$ be the class of all strong continuous Kleene monads. The problem $\mathfrak{C}_\omega \models p = q$ is non-r.e.

Corollary. There is no complete recursive axiomatization of the equational theory of strong continuous Kleene monads.

Idea of the proof: reduction from Post correspondence problem.

Post correspondence problem (PCP)

Let Σ be a finite alphabet ($|\Sigma| \geqslant 2$) and Σ^* be the set of words over Σ.

Theorem (Post). There exists $B = \{\langle p_1, q_1\rangle, \ldots, \langle p_n, q_n\rangle\}$ with $p_i, q_i \in \Sigma^*$ such that it is undecidable if there are indices $i_1, \ldots, i_m$ for which $p_{i_1} p_{i_2} \ldots p_{i_m} \equiv q_{i_1} q_{i_2} \ldots q_{i_m}$.

Every such B is called a PCP instance. Every sequence of indices $i_1, \ldots, i_m$ for which $p_{i_1} p_{i_2} \ldots p_{i_m} \equiv q_{i_1} q_{i_2} \ldots q_{i_m}$ is called a solution of the PCP instance.

Example. PCP instance: $\left\langle \begin{matrix} 0 \\ 100 \end{matrix}, \begin{matrix} 01 \\ 00 \end{matrix}, \begin{matrix} 110 \\ 11 \end{matrix} \right\rangle$.

Solution:

110	01	110	0
11	00	11	100

Post correspondence problem (PCP)

Let Σ be a finite alphabet ($|\Sigma| \geqslant 2$) and Σ^* be the set of words over Σ.

Theorem (Post). There exists $B = \{\langle p_1, q_1\rangle, \ldots, \langle p_n, q_n\rangle\}$ with $p_i, q_i \in \Sigma^*$ such that it is undecidable if there are indices $i_1, \ldots, i_m$ for which $p_{i_1} p_{i_2} \ldots p_{i_m} \equiv q_{i_1} q_{i_2} \ldots q_{i_m}$.

Every such B is called a PCP instance. Every sequence of indices $i_1, \ldots, i_m$ for which $p_{i_1} p_{i_2} \ldots p_{i_m} \equiv q_{i_1} q_{i_2} \ldots q_{i_m}$ is called a solution of the PCP instance.

Example. PCP instance: $\left\langle \begin{array}{|c|}\hline 0 \\ \hline 100 \\ \hline \end{array}, \begin{array}{|c|}\hline 01 \\ \hline 00 \\ \hline \end{array}, \begin{array}{|c|}\hline 110 \\ \hline 11 \\ \hline \end{array} \right\rangle$.

Solution:

110	01	110	0
11	00	11	100

.

Outline of the proof of negative result #1

Let $\Sigma = \{a_1, a_2\}$, and let $\{\langle p_1, q_1\rangle, \ldots, \langle p_n, q_n\rangle\}$ be a PCP instance in the alphabet Σ. Every word $s \in (\Sigma \cup \{c\})^*$ can be considered as a program: $\epsilon \mapsto \mathrm{ret}\,\star$, $a_{i_1} \ldots a_{i_n} \mapsto (\mathrm{do}\ \ a_{i_1}; \ldots; a_{i_n})$. Then

$$s = \mathrm{do}\ \ x \leftarrow \Big(\mathrm{init}\,x \leftarrow \sum_{i=1}^{n} \mathrm{ret}(\mathrm{do}\ \ p_i; c; q_i^{-1})\ \mathrm{in} \sum_{i=1}^{n} \mathrm{ret}(\mathrm{do}\ \ p_i; x; q_i^{-1})^*\Big); x$$

generates all possible strings of the form lcr^{-1} with $l = p_{i_1} \ldots p_{i_k}$ and $r = q_{i_1} \ldots q_{i_k}$.

The PCP instance does have a solution iff

$$\mathrm{ret}\,s \leqslant \mathrm{do}\ \ x \leftarrow t; \mathrm{ret}(s + x)$$

where t presents the collection $\{\mathrm{ret}(\mathrm{do}\ \ l; c; l^{-1}) \mid l \in \Sigma^*\}$:

$$t = \mathrm{init}\,x \leftarrow \mathrm{ret}\,c\ \mathrm{in} \sum_{i=1}^{n} \mathrm{ret}(\mathrm{do}\ \ a_1; x; a_1 + \mathrm{do}\ \ a_2; x; a_2)^*$$

Outline of the proof of negative result #1

Let $\Sigma = \{a_1, a_2\}$, and let $\{\langle p_1, q_1\rangle, \ldots, \langle p_n, q_n\rangle\}$ be a PCP instance in the alphabet Σ. Every word $s \in (\Sigma \cup \{c\})^*$ can be considered as a program: $\epsilon \mapsto \mathsf{ret}\star$, $a_{i_1} \ldots a_{i_n} \mapsto (\mathsf{do}\ \ a_{i_1}; \ldots; a_{i_n})$. Then

$$s = \mathsf{do}\ \ x \leftarrow \Big(\mathsf{init}\, x \leftarrow \sum_{i=1}^{n} \mathsf{ret}(\mathsf{do}\ \ p_i; c; q_i^{-1})\ \mathsf{in} \sum_{i=1}^{n} \mathsf{ret}(\mathsf{do}\ \ p_i; x; q_i^{-1})^*\Big); x$$

generates all possible strings of the form lcr^{-1} with $l = p_{i_1} \ldots p_{i_k}$ and $r = q_{i_1} \ldots q_{i_k}$. The PCP instance does have a solution iff

$$\mathsf{ret}\, s \leqslant \mathsf{do}\ \ x \leftarrow t; \mathsf{ret}(s + x)$$

where t presents the collection $\{\mathsf{ret}(\mathsf{do}\ \ l; c; l^{-1}) \mid l \in \Sigma^*\}$:

$$t = \mathsf{init}\, x \leftarrow \mathsf{ret}\, c\ \mathsf{in} \sum_{i=1}^{n} \mathsf{ret}(\mathsf{do}\ \ a_1; x; a_1 + \mathsf{do}\ \ a_2; x; a_2)^*$$

Decidable and complete fragment of MCE

A program is regular if for any of its subterms $(\mathsf{init}\, x \leftarrow q\, \mathsf{in}\, r^*)$ and for any subterm $(\mathsf{ret}\, t)$ of r, t does not contain control structures (i.e. ret, binding, deadlock, choice and Kleene star).

Example: $\mathsf{init}\, x \leftarrow p\, \mathsf{in}(\mathsf{do}\ \ y \leftarrow q; \mathsf{ret}\langle \mathsf{fst}(x), y\rangle)^*$.

Theorem. Given two regular programs p and q, the equality $p = q$ holds over all strong Kleene monads iff it holds over all strong continuous Kleene monads. Moreover, the equational theory of strong Kleene monads w.r.t. regular programs is decidable.

Idea of the proof: ~~torturous~~ successive reduction to more and more narrow program classes with further call of Kozen's completeness theorem for Kleene algebra calculus over the algebra of regular events.

A simple example: "loop optimization"

Let $p = \mathsf{init}\, x \leftarrow s\, \mathsf{in}(\mathsf{do}\ y \leftarrow f(x_1, x_2); \mathsf{ret}\langle x_1, y\rangle)^*$

and $q = s + \mathsf{do}\ x \leftarrow s; y \leftarrow (\mathsf{init}\, y \leftarrow f(x_1, x_2)\, \mathsf{in}\, f(x_1, y)^*); \mathsf{ret}\langle x_1, y\rangle$

where $x_1 = \mathsf{fst}(x), x_2 = \mathsf{snd}(x)$.

One can see that $p = q$ in the continuous case:

$$
\begin{aligned}
p = {} & s + \mathsf{do}\ x \leftarrow s; y \leftarrow f(x_1, x_2); \mathsf{ret}\langle x_1, y\rangle \\
& + \mathsf{do}\ x \leftarrow s; y \leftarrow f(x_1, x_2); x \leftarrow \mathsf{ret}\langle x_1, y\rangle; y \leftarrow f(x_1, x_2); \mathsf{ret}\langle x_1, y\rangle \\
& + \ldots \\
q = {} & s + \mathsf{do}\ x \leftarrow s; y \leftarrow f(x_1, x_2); \mathsf{ret}\langle x_1, y\rangle \\
& + \mathsf{do}\ x \leftarrow s; y \leftarrow f(x_1, x_2); y \leftarrow f(x_1, y); \mathsf{ret}\langle x_1, y\rangle \\
& + \ldots
\end{aligned}
$$

Therefore p and q are provably equal.

A simple example: "loop optimization"

Let $p = \mathsf{init}\,\mathbf{x} \leftarrow s\,\mathsf{in}(\mathsf{do}\ y \leftarrow f(\mathbf{x_1}, \mathbf{x_2}); \mathsf{ret}\langle \mathbf{x_1}, y\rangle)^*$

and $q = s + \mathsf{do}\ x \leftarrow s; y \leftarrow (\mathsf{init}\,\mathbf{y} \leftarrow f(x_1, x_2)\,\mathsf{in}\, f(x_1, \mathbf{y})^*); \mathsf{ret}\langle x_1, y\rangle$

where $x_1 = \mathsf{fst}(x), x_2 = \mathsf{snd}(x)$.

One can see that $p = q$ in the continuous case:

$$
\begin{aligned}
p = {} & s + \mathsf{do}\ x \leftarrow s; y \leftarrow f(x_1, x_2); \mathsf{ret}\langle x_1, y\rangle \\
& + \mathsf{do}\ x \leftarrow s; y \leftarrow f(x_1, x_2); x \leftarrow \mathsf{ret}\langle x_1, y\rangle; y \leftarrow f(x_1, x_2); \mathsf{ret}\langle x_1, y\rangle \\
& + \ldots \\
q = {} & s + \mathsf{do}\ x \leftarrow s; y \leftarrow f(x_1, x_2); \mathsf{ret}\langle x_1, y\rangle \\
& + \mathsf{do}\ x \leftarrow s; y \leftarrow f(x_1, x_2); y \leftarrow f(x_1, y); \mathsf{ret}\langle x_1, y\rangle \\
& + \ldots
\end{aligned}
$$

Therefore p and q are provably equal.

Future work

- Justify and study Kleene monads with tests, an extension of Kleene monads allowing to interpret imperative control structure according to Fischer-Ladner encoding:

$$\begin{aligned} \mathrm{if}(\mathrm{b},\mathrm{p},\mathrm{q}) &= \text{do } \mathrm{b}?;\mathrm{p} + \text{do } \bar{\mathrm{b}}?;\mathrm{q} \\ \mathit{while}(\mathrm{b},\mathrm{p}) &= \text{do } (\text{init } \mathrm{b}? \text{ in } \mathrm{p}^*);\bar{\mathrm{b}}? \end{aligned}$$

- Integrate Kleene monad calculus with monad-based dynamic logics.
- Extend the decidability result for the case of an underlying data theory.

The End

Thanks for your attention!

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH
Sichere Kognitive Systeme