Semantic Structure for Programming Languages with Effects

(a 6-lecture course)

Alex Simpson

Laboratory for Foundations of Computer Science
School of Informatics, University of Edinburgh

PhD School, CIRM, 14–18 April 2014

Course structure

Lecture 1: Computational effects and monads.

Lecture 2: Algebraic effects.

Lecture 3: A metalanguage for effects.

Lecture 4: Continuations and control.

Lecture 5: Functor category models of local store.

Lecture 6: Brief survey of selected literature

Lecture 1: Computational effects and monads.

Effect-free (pure) computation

- A program P specifies a function $[\![P]\!]$ from a set A of input values to a set B of output values

$$[\![P]\!] : A \to B$$

- Nicely implemented by typed λ-calculi (simply-typed, or dependently-typed)

- Typically, such calculi are strongly normalizing and confluent. So all computation terminates with a result that is independent of the evaluation order used.

- The standard equational rules of λ-calculus (e.g., β/η-equalities) can be used to prove equality of programs.

The effect of effects

Dependently-typed programming languages, such as Agda, illustrate the richness and power of effect-free computation.

But normal programming is full of non-functional effects.

- print `"Hello World!"` : unit
- random $\{1, \ldots, 6\}$: nat
- $l := 3$: unit
- if $mm < 1 \mid\mid mm > 12$ then raise OutOfBounds else ...
- loop
- etc.

We no longer have termination in general, and evaluation order is also important.

E.g.,:

$$(\lambda d.\, d + d)\,(\mathsf{random}\ \{1, 2\}) \ :\ \mathsf{nat}$$

$$= \left\{2\left(\frac{1}{2}\right), 4\left(\frac{1}{2}\right)\right\} \qquad \text{call by value/need}$$

$$= \left\{2\left(\frac{1}{4}\right), 3\left(\frac{1}{2}\right), 4\left(\frac{1}{4}\right)\right\} \qquad \text{call by name}$$

E.g.,:

$$(\lambda c.\, \mathsf{let}\ y = !l\ \mathit{in}\ c; c; y)\,(l := !l + 1) \ :\ \mathsf{nat}$$

which differs in behaviour in each of call-by-value, call-by-name, and call-by-need. (We shall not further consider call-by-need.)

We no longer have termination in general, and evaluation order is also important.

E.g.,:

$$(\lambda d.\, d + d)\,(\mathsf{random}\ \{1, 2\}) \ :\ \mathsf{nat}$$

$$= \left\{2\left(\frac{1}{2}\right), 4\left(\frac{1}{2}\right)\right\} \qquad \text{call by value/need}$$

$$= \left\{2\left(\frac{1}{4}\right), 3\left(\frac{1}{2}\right), 4\left(\frac{1}{4}\right)\right\} \qquad \text{call by name}$$

E.g.,:

$$(\lambda c.\, \mathsf{let}\ y = !l\ \mathit{in}\ c; c; y)\,(l := !l + 1) \ :\ \mathsf{nat}$$

which differs in behaviour in each of call-by-value, call-by-name, and call-by-need. (We shall not further consider call-by-need.)

β and η equalities fail for call-by-value.

E.g.,

$$(\lambda x.\, 0)\,(\text{raise OutOfBounds}) \;\neq\; 0$$

is a failure of β-equality.

(The terms on the previous slide are also not call-by-value equal to their β-redexes.)

E.g.,

$$l := 0; (\lambda x.\, 0) \;\neq\; \lambda y.\, (l := 0; (\lambda x.\, 0))\, y \;:\; \text{nat} \to \text{nat}$$

is a failure of η-equality.

Values versus computations

The presence of effects forces one to distinguish between the set A of values that can be returned by a computation, and the set TA of all computations that can be typed as (potentially) producing such values.

We can then model a program P that takes an input value in A and then computes a value in B (potentially causing effects, and potentially never actually terminating with a final value) as a function

$$[\![P]\!] : A \to TB$$

At this intuitive level of generality, it is already possible to address the question of what structure TA should carry in general.

Required structure on TA

- For every value $a \in A$ there should be a trivial computation $\eta(a) \in TA$ whose execution simply returns the value a (and does nothing else in the process).

- Given a program $A \xrightarrow{Q} TB$ and a computation $P \in TA$, we should be able to sequence their executions, to produce the computation in TB informally described by:

  execute P until (if ever) it returns a value $a \in A$, then feed a as input to Q and execute $Q(a)$.

  Thus Q should give rise to a derived function $TA \xrightarrow{Q^*} TB$.

Remarkably, these two simple properties (in a slightly generalized parametric form) suffice for the development of a rich theory of computational effects.

The natural level of generality is to presented the structure in the setting of an arbitrary category $\mathcal{V}$.

Kleisli triple

A Kleisli triple on a category $\mathcal{V}$ is given by:

- for every object A, an object TA and a morphism $A \xrightarrow{\eta_A} TA$,
- for every map $A \xrightarrow{f} TB$, a map $TA \xrightarrow{f^*} TB$,
- satisfying the equations below.
  1. $(\eta_A)^* = 1_{TA} \colon TA \longrightarrow TA$
  2. $f^* \circ \eta_A = f \colon A \longrightarrow TB$
  3. For any $A \xrightarrow{f} TB$ and $B \xrightarrow{g} TC$,
     $g^* \circ f^* = (g^* \circ f)^* \colon TA \to TC$

Derived monad structure

Given a Kleisli triple on $\mathcal{V}$:

- For $A \xrightarrow{f} B$ define $TA \xrightarrow{Tf} TB$ by $Tf := (\eta_B \circ f)^*$.
- Define $TTA \xrightarrow{\mu_A} TA$ by $\mu_A := (1_{TA})^*$.

Proposition 1. For any Kleisli triple on $\mathcal{V}$, the derived (T, η, μ) is a monad on $\mathcal{V}$. Moreover, this construction establishes a bijective correspondence between Kleisli triples and monads on $\mathcal{V}$.

As mentioned earlier, we actually need a parameterized version of this structure.

For this, we assume that $\mathcal{V}$ has finite products.

Derived monad structure

Given a Kleisli triple on $\mathcal{V}$:

- For $A \xrightarrow{f} B$ define $TA \xrightarrow{Tf} TB$ by $Tf := (\eta_B \circ f)^*$.
- Define $TTA \xrightarrow{\mu_A} TA$ by $\mu_A := (1_{TA})^*$.

Proposition 1. For any Kleisli triple on $\mathcal{V}$, the derived (T, η, μ) is a monad on $\mathcal{V}$. Moreover, this construction establishes a bijective correspondence between Kleisli triples and monads on $\mathcal{V}$.

As mentioned earlier, we actually need a parameterized version of this structure.

For this, we assume that $\mathcal{V}$ has finite products.

Parameterized Kleisli triple

A parameterized Kleisli triple on a category $\mathcal{V}$ with finite products is given by:

- for every object A, an object TA and a morphism $A \xrightarrow{\eta_A} TA$,
- for every map $X \times A \xrightarrow{f} TB$, a map $X \times TA \xrightarrow{f^\dagger} TB$,
- satisfying the equations below.

1. $(\eta_A \circ \pi_2)^\dagger = \pi_2 \colon X \times TA \longrightarrow TA$
2. $f^\dagger \circ (1_X \times \eta_A) = f \colon X \times A \longrightarrow TB$
3. For any $X \times A \xrightarrow{f} TB$ and $X \times B \xrightarrow{g} TC$,
   $g^\dagger \circ (\pi_1, f^\dagger) = (g^\dagger \circ (\pi_1, f))^\dagger \colon X \times TA \to TC$
4. For any $X \xrightarrow{h} Y$ and $Y \times A \xrightarrow{f} TB$,
   $(f \circ (h \times 1_A))^\dagger = f^\dagger \circ (h \times 1_{TA}) \colon X \times TA \longrightarrow TB$

Derived strong monad structure

Given a parameterized Kleisli triple on $\mathcal{V}$.

- For $A \xrightarrow{f} TB$ define $TA \xrightarrow{f^*} TB$ by $f^* := (f \circ \pi_2)^\dagger \circ (!_A, 1_A)$, where $f \circ \pi_2$ has the typing

$$\mathbf{1} \times A \xrightarrow{\pi_2} A \xrightarrow{f} TB$$

  This derives a Kleisli triple on $\mathcal{V}$. (The functorial action of T, and natural transformation μ are then derived as before.)

- Define $X \times TA \xrightarrow{t_{X,A}} T(X \times A)$ by $t_{X,A} := (\eta_{X \times A})^\dagger$.

Proposition 2. For any parameterized Kleisli triple on $\mathcal{V}$, the derived (T, η, μ, t) is a strong monad on $\mathcal{V}$. Moreover, this construction establishes a bijective correspondence between parameterized Kleisli triples and strong monads on $\mathcal{V}$.

Examples

Many examples of standard computational effects are modelled by strong monads (equivalently parameterized Kleisli triples).

We consider examples (presented as parameterized Kleisli triples) in the category **Set** of sets, though all make sense more generally than this. (We make this explicit in many cases.)

We use one main example, global store, as a running example, to illustrate the general usage of strong monads for modelling effects.

Following that, we look at many other examples more briefly.

Running example: Global state/store

Let S be a set (of states).

Parameterized Kleisli triple (for the global state monad):

$$
\begin{aligned}
TA &= S \to (A \times S) \\
\eta_A &= a \mapsto \lambda s.\,(a, s) \\
f^\dagger &= (x, P) \mapsto \lambda s.\, f(x, a')(s') \quad \text{where } P(s) \text{ is } (a', s')
\end{aligned}
$$

(This defines a parameterized Kleisli triple for any object S in a cartesian closed category $\mathcal{V}$)

In the case that S is given as a function space $\mathrm{Loc} \to \mathrm{Val}$, we call the induced monad the global store monad.

Running example: Global state/store

Let S be a set (of states).

Parameterized Kleisli triple (for the global state monad):

$$
\begin{aligned}
TA &= S \to (A \times S) \\
\eta_A &= a \mapsto \lambda s.\,(a, s) \\
f^\dagger &= (x, P) \mapsto \lambda s.\, f(x, a')(s') \quad \text{where } P(s) \text{ is } (a', s')
\end{aligned}
$$

(This defines a parameterized Kleisli triple for any object S in a cartesian closed category $\mathcal{V}$)

In the case that S is given as a function space $\mathrm{Loc} \to \mathrm{Val}$, we call the induced monad the global store monad.

Running example: Global state/store

Let S be a set (of states).

Parameterized Kleisli triple (for the global state monad):

$$
\begin{aligned}
TA &= S \to (A \times S) \\
\eta_A &= a \mapsto \lambda s.\,(a, s) \\
f^\dagger &= (x, P) \mapsto \lambda s.\, f(x, a')(s') \quad \text{where } P(s) \text{ is } (a', s')
\end{aligned}
$$

(This defines a parameterized Kleisli triple for any object S in a cartesian closed category $\mathcal{V}$)

In the case that S is given as a function space $\mathsf{Loc} \to \mathsf{Val}$, we call the induced monad the global store monad.

The Kleisli category

In the Kleisli category, morphisms model the idea that a program taking input values in A and returning output values in B should be modelled by a map $A \longrightarrow TB$.

Given a Kleisli triple on $\mathcal{V}$, the Kleisli category, $\mathcal{V}_T$, has the same objects as $\mathcal{C}$ and morphisms $\mathcal{V}_T(A, B) := \mathcal{V}(A, TB)$.

- The identity on A is $A \xrightarrow{\eta} TA$.
- The composition of $A \xrightarrow{f} TB$ and $B \xrightarrow{g} TC$ is $g^* \circ f : A \longrightarrow TC$.

Global state: Note the one-to-one correspondence

$$\frac{A \longrightarrow S \to (B \times S)}{A \times S \longrightarrow B \times S}$$

Identity and composition then behave as expected.

Modelling typed λ-calculus with effects

We use simply-typed λ-caclulus with types (α ranges over chosen collection of base types)

$$\tau \;::=\; \alpha \mid \tau_1 \to \tau_2$$

Standard rules generate judgements

$$\Gamma \;\vdash\; t \;:\; \tau$$

where Γ is a typing context.

Specialized to global store: Include a base type Val.

For every $l \in \mathrm{Loc}$ include term constructors

$$\frac{\Gamma \vdash t : \mathrm{Val} \quad \Gamma \vdash t' : \tau}{\Gamma \;\vdash\; (l := t)\,;\, t' \;:\; \tau} \qquad \frac{\Gamma,\, x{:}\mathrm{Val} \vdash t : \tau}{\Gamma \;\vdash\; \mathsf{let}\; x = !l \;\mathsf{in}\; t \;:\; \tau}$$

Modelling typed λ-calculus with effects

We use simply-typed λ-caclulus with types (α ranges over chosen collection of base types)

$$\tau \;::=\; \alpha \mid \tau_1 \to \tau_2$$

Standard rules generate judgements

$$\Gamma \;\vdash\; t \;:\; \tau$$

where Γ is a typing context.

Specialized to global store: Include a base type Val.

For every $l \in \mathsf{Loc}$ include term constructors

$$\frac{\Gamma \vdash t : \mathsf{Val} \quad \Gamma \vdash t' : \tau}{\Gamma \;\vdash\; (l := t)\,;\, t' \;:\; \tau} \qquad \frac{\Gamma, x{:}\mathsf{Val} \vdash t : \tau}{\Gamma \;\vdash\; \mathsf{let}\ x = !l\ \mathsf{in}\ t \;:\; \tau}$$

Call-by-value interpretation

Given an assignment of a set $[\![\alpha]\!]$ to every base type α, define

$$\begin{aligned} [\![\alpha]\!]^{\mathsf{vl}} &= [\![\alpha]\!] \\ [\![\sigma \to \tau]\!]^{\mathsf{vl}} &= [\![\sigma]\!]^{\mathsf{vl}} \to T[\![\tau]\!]^{\mathsf{vl}} \end{aligned}$$

Given an environment ρ mapping every $x : \sigma$ in Γ to $\rho(x) \in [\![\sigma]\!]^{\mathsf{vl}}$, the interpretation of a term $\Gamma \vdash t : \tau$ is $[\![t]\!]^{\mathsf{vl}}_\rho \in T[\![\tau]\!]^{\mathsf{vl}}$ defined by:

$$\begin{aligned} [\![x]\!]^{\mathsf{vl}}_\rho &= \eta(\rho(x)) \\ [\![t\ t']\!]^{\mathsf{vl}}_\rho &= (f \mapsto (v \mapsto f(v))^* [\![t']\!]^{\mathsf{vl}}_\rho)^* [\![t]\!]^{\mathsf{vl}}_\rho \\ [\![\lambda x{:}\sigma.\ t]\!]^{\mathsf{vl}}_\rho &= \eta\left(v \mapsto [\![t]\!]^{\mathsf{vl}}_{\rho[x:=v]}\right) \end{aligned}$$

For global store: Define $[\![\mathsf{Val}]\!]^{\mathsf{vl}} = \mathsf{Val}$.

$$[\![(l := t)\,;\, t']\!]^{\mathsf{vl}}_\rho = \lambda s.\ \mathsf{let}\ (v, s') = [\![t]\!]^{\mathsf{vl}}_\rho(s)\ \mathsf{in}\ [\![t']\!]^{\mathsf{vl}}_\rho(s'[l := v])$$

$$[\![\mathsf{let}\ x = !l\ \mathsf{in}\ t]\!]^{\mathsf{vl}}_\rho = \lambda s.\ [\![t]\!]^{\mathsf{vl}}_{\rho[x:=s(l)]}(s)$$

Call-by-value interpretation

Given an assignment of a set $[\![\alpha]\!]$ to every base type α, define

$$\begin{aligned} [\![\alpha]\!]^{\mathsf{vl}} &= [\![\alpha]\!] \\ [\![\sigma \to \tau]\!]^{\mathsf{vl}} &= [\![\sigma]\!]^{\mathsf{vl}} \to T[\![\tau]\!]^{\mathsf{vl}} \end{aligned}$$

Given an environment ρ mapping every $x:\sigma$ in Γ to $\rho(x) \in [\![\sigma]\!]^{\mathsf{vl}}$, the interpretation of a term $\Gamma \vdash t : \tau$ is $[\![t]\!]^{\mathsf{vl}}_\rho \in T[\![\tau]\!]^{\mathsf{vl}}$ defined by:

$$\begin{aligned} [\![x]\!]^{\mathsf{vl}}_\rho &= \eta(\rho(x)) \\ [\![t\ t']\!]^{\mathsf{vl}}_\rho &= (f \mapsto (v \mapsto f(v))^* [\![t']\!]^{\mathsf{vl}}_\rho)^* [\![t]\!]^{\mathsf{vl}}_\rho \\ [\![\lambda x{:}\sigma.\ t]\!]^{\mathsf{vl}}_\rho &= \eta\left(v \mapsto [\![t]\!]^{\mathsf{vl}}_{\rho[x:=v]}\right) \end{aligned}$$

For global store: Define $[\![\mathsf{Val}]\!]^{\mathsf{vl}} = \mathsf{Val}$.

$$[\![(l := t)\,;\ t']\!]^{\mathsf{vl}}_\rho = \lambda s.\ \mathsf{let}\ (v, s') = [\![t]\!]^{\mathsf{vl}}_\rho(s)\ \mathsf{in}\ [\![t']\!]^{\mathsf{vl}}_\rho(s'[l := v])$$

$$[\![\mathsf{let}\ x = !l\ \mathsf{in}\ t]\!]^{\mathsf{vl}}_\rho = \lambda s.\ [\![t]\!]^{\mathsf{vl}}_{\rho[x:=s(l)]}(s)$$

"Lazy" call-by-name interpretation

This translation validates β equality, but not η.

$$\begin{aligned} [\![\alpha]\!]^{\mathsf{lz}} &= [\![\alpha]\!] \\ [\![\sigma \to \tau]\!]^{\mathsf{lz}} &= T[\![\sigma]\!]^{\mathsf{lz}} \to T[\![\tau]\!]^{\mathsf{lz}} \end{aligned}$$

Given an environment ρ mapping every $x : \sigma$ in Γ to $\rho(x) \in T[\![\sigma]\!]^{\mathsf{lz}}$, the interpretation of a term $\Gamma \vdash t : \tau$ is

$$[\![t]\!]^{\mathsf{lz}}_{\rho} \in T[\![\tau]\!]^{\mathsf{lz}}$$

Exercise: Work out the definition of $[\![t]\!]^{\mathsf{lz}}_{\rho}$.

"Lazy" call-by-name interpretation

This translation validates β equality, but not η.

$$\begin{aligned} [\![\alpha]\!]^{\mathsf{lz}} &= [\![\alpha]\!] \\ [\![\sigma \to \tau]\!]^{\mathsf{lz}} &= T[\![\sigma]\!]^{\mathsf{lz}} \to T[\![\tau]\!]^{\mathsf{lz}} \end{aligned}$$

Given an environment ρ mapping every $x : \sigma$ in Γ to $\rho(x) \in T[\![\sigma]\!]^{\mathsf{lz}}$, the interpretation of a term $\Gamma \vdash t : \tau$ is

$$[\![t]\!]^{\mathsf{lz}}_{\rho} \in T[\![\tau]\!]^{\mathsf{lz}}$$

Exercise: Work out the definition of $[\![t]\!]^{\mathsf{lz}}_{\rho}$.

Call-by-name interpretation

This translation validates both β and η equalities.

$$\begin{aligned} [\![\alpha]\!]^{\mathsf{nm}} &= T[\![\alpha]\!] \\ [\![\sigma \to \tau]\!]^{\mathsf{nm}} &= [\![\sigma]\!]^{\mathsf{nm}} \to [\![\tau]\!]^{\mathsf{nm}} \end{aligned}$$

Given an environment ρ mapping every $x : \sigma$ in Γ to $\rho(x) \in [\![\sigma]\!]^{\mathsf{nm}}$, for pure typed λ-calculus, the interpretation of a term $\Gamma \vdash t : \tau$ as $[\![t]\!]^{\mathsf{nm}}_{\rho} \in [\![\tau]\!]^{\mathsf{nm}}$ is completely standard:

$$\begin{aligned} [\![x]\!]^{\mathsf{nm}}_{\rho} &= \rho(x) \\ [\![t\ t']\!]^{\mathsf{nm}}_{\rho} &= [\![t']\!]^{\mathsf{nm}}_{\rho}\ ([\![t]\!]^{\mathsf{nm}}_{\rho}) \\ [\![\lambda x : \sigma.\ t]\!]^{\mathsf{nm}}_{\rho} &= \left(v \mapsto [\![t]\!]^{\mathsf{nm}}_{\rho[x:=v]}\right) \end{aligned}$$

The interpretation of operations dealing with effects is more interesting. We cover this in Lecture 2.

Example: Finite nondeterministic choice

Parameterized Kleisli triple:

$$
\begin{aligned}
TA &= \text{ the set, } \mathcal{P}_{\mathrm{fin}^+}(A), \text{ of finite nonempty subsets of } A \\
\eta_A &= \{a\} \quad \text{(singleton)} \\
f^\dagger &= (x, P) \mapsto \bigcup_{a' \in P} f(x, a')
\end{aligned}
$$

Example: Finite probabilistic choice

A finite probability distribution on a set A is a function $P\colon A \to [0,1]$ that is non-zero on finitely many elements of A and satisfies

$$\sum_{a \in A} P(a) \;=\; 1 \;\;.$$

Parameterized Kleisli triple:

$TA =$ the set, $\mathit{Dist}(A)$, of finite probability distributions on A

$\eta_A = a \mapsto \delta_a$ (Kronecker delta)

$f^\dagger = (x, P) \mapsto \lambda b.\ \sum_{a \in A} P(a) \,.\, f(x,a)(b)$

Example: Exceptions

Let E be a set (of exceptions).

Parameterized Kleisli triple:

$$TA = A + E$$
$$\eta_A = a \mapsto \mathsf{in}_1(a)$$
$$f^\dagger = (x, P) \mapsto \begin{cases} \mathsf{in}_1(f(x, a)) & \text{if } P = \mathsf{in}_1(a) \\ \mathsf{in}_2(e) & \text{if } P = \mathsf{in}_2(e) \end{cases}$$

This defines a parameterized Kleisli triple for any object E in a distributive category $\mathcal{V}$

Example: Exceptions

Let E be a set (of exceptions).

Parameterized Kleisli triple:

$$TA = A + E$$
$$\eta_A = a \mapsto \mathsf{in}_1(a)$$
$$f^\dagger = (x, P) \mapsto \begin{cases} \mathsf{in}_1(f(x, a)) & \text{if } P = \mathsf{in}_1(a) \\ \mathsf{in}_2(e) & \text{if } P = \mathsf{in}_2(e) \end{cases}$$

This defines a parameterized Kleisli triple for any object E in a distributive category $\mathcal{V}$

Example: Continuations

Let R be a set (of results).

Parameterized Kleisli triple:

$$TA = (A \to R) \to R$$
$$\eta_A = a \mapsto \lambda k.\, k(a)$$
$$f^\dagger = (x, P) \mapsto \lambda k.\, P(\lambda a.\, f(x, a)(k))$$

This defines a parameterized Kleisli triple for any object R in a cartesian closed category $\mathcal{V}$

Example: Continuations

Let R be a set (of results).

Parameterized Kleisli triple:

$$TA = (A \to R) \to R$$
$$\eta_A = a \mapsto \lambda k.\, k(a)$$
$$f^\dagger = (x, P) \mapsto \lambda k.\, P(\lambda a.\, f(x, a)(k))$$

This defines a parameterized Kleisli triple for any object R in a cartesian closed category $\mathcal{V}$

Example: Interactive output

Let U be a set (of output messages).

(Typically, $U = \Sigma^*$ for some output alphabet Σ, but the general definition does not assume this.)

Parameterized Kleisli triple:

$$\begin{aligned} TA &= \text{ the set, } U^* \times A \\ \eta_A &= a \mapsto (\varepsilon, a) \\ f^\dagger &= (x, (w, a)) \mapsto (ww', b) \quad \text{where } f(x, a) \text{ is } (w', b) \end{aligned}$$

This defines a parameterized Kleisli triple for any object U in a distributive category with parameterized free monoids.

Example: Interactive output

Let U be a set (of output messages).

(Typically, $U = \Sigma^*$ for some output alphabet Σ, but the general definition does not assume this.)

Parameterized Kleisli triple:

$$
\begin{aligned}
TA &= \text{ the set, } U^* \times A \\
\eta_A &= a \mapsto (\varepsilon, a) \\
f^\dagger &= (x, (w, a)) \mapsto (ww', b) \quad \text{where } f(x, a) \text{ is } (w', b)
\end{aligned}
$$

This defines a parameterized Kleisli triple for any object U in a distributive category with parameterized free monoids.

Exercises

1. Show that every Kleisli triple on *Set* extends to a unique parameterized Kleisli triple.
2. Verify the bijective correspondences between (parameterized) Kleisli triples and (strong) monads stated in Propositions 1 and 2.
3. Verify that the axioms for (parameterized) Kleisli triples are satisfied by the various examples given.
4. Verify that the Kleisli category is indeed a category. Construct an adjunction $J \dashv K : \mathcal{V}_T \to \mathcal{V}$ such that the composite KJ is the functor T.
5. Give the interpretation of terms for the lazy call-by-name semantics.

Lecture 2: Algebraic effects.

Effect-triggering operations

- Nondeterministic choice
  One binary operation, or
- Probabilistic choice
  A binary operation, $x \,{}_\lambda{+}\, y$, for every $\lambda \in [0,1]$.
- Exceptions
  A constant, raise_e, for every $e \in E$.
- Interactive output
  A unary operation, print_m, for every $m \in U$.
- Global store
  A Val-ary operation, lookup_l, for every $l \in \mathsf{Loc}$.
  A unary operation, $\mathsf{update}_{l,v}$, for every $l \in \mathsf{Loc}$ and $v \in \mathsf{Val}$.

We consider algebras for equational theories expressing the natural equalities between such operations.

Algebras for nondeterministic choice

x or y: make a binary nondeterministic choice between continuing as x or continuing as y.

Equational theory of binary semilattices:

$$\begin{aligned} x \text{ or } x &= x \\ x \text{ or } y &= y \text{ or } x \\ (x \text{ or } y) \text{ or } z &= x \text{ or } (y \text{ or } z) \end{aligned}$$

An algebra (a binary semilatice) $\underline{A}$ is a pair (A, or), where A is a set, and the function $\text{or} \colon A^2 \to A$ satisfies the equations above.

Algebras for probabilistic choice

$x \ {}_\lambda{+} \ y$: make a probabilistic choice between continuing as x with probability λ, or continuing as y with probability $1 - \lambda$.

Equational theory of convex sets:

$$x \ {}_\lambda{+} \ x = x$$

$$x \ {}_1{+} \ y = x$$

$$x \ {}_\lambda{+} \ y = y \ {}_{(1\text{-}\lambda)}{+} \ x$$

$$(x \ {}_\lambda{+} \ y) \ {}_{\lambda'}{+} \ z = x \ {}_{\lambda\lambda'}{+} \ (y \ {}_{(\frac{\lambda'\text{-}\lambda\lambda'}{1\text{-}\lambda\lambda'})}{+} \ z)$$

(where the last equation has the condition $\lambda\lambda' \neq 1$).

An algebra (a convex set) $\underline{A}$ is a pair $(A, {}_{(\cdot)}{+})$, where A is a set, and the function ${}_{(\cdot)}{+} \colon [0,1] \times A^2 \to A$ satisfies the equations above.

Algebras for exceptions and interactive output

Exceptions

raise_e : raise exception e (there is no continuing computation).

No equations.

An algebra $\underline{A}$ is a pair $(A, \mathsf{raise}_{(\cdot)})$, where A is a set, and $\mathsf{raise}_{(\cdot)} : E \to A$.

Interactive output

$\mathsf{print}_m(x)$: print message m and then continue computation with x.

Again, no equations.

An algebra $\underline{A}$ is a pair $(A, \mathsf{print}_{(\cdot)})$, where A is a set, $\mathsf{print}_{(\cdot)} : U \times A \to A$.

Algebras for global store

$\mathsf{lookup}_l\,(x_v)_{v\in\mathsf{Val}}$: extract the value v currently stored location l and then continue as x_v.

$\mathsf{update}_{l,v}(x)$: modify the store by performing $l := v$, then continue as x.

Equational theory of mnemoids:

$$\mathsf{lookup}_l\,(\mathsf{lookup}_l\,(x_{v,v'})_{v'})_v = \mathsf{lookup}_l\,(x_{v,v})_v$$
$$\mathsf{lookup}_l\,(\mathsf{lookup}_{l'}\,(x_{v,v'})_{v'})_v = \mathsf{lookup}_{l'}\,(\mathsf{lookup}_l\,(x_{v,v'})_v)_{v'}$$
$$\mathsf{update}_{l,v}(\mathsf{update}_{l,v'}(x)) = \mathsf{update}_{l,v'}(x)$$
$$\mathsf{update}_{l,v}(\mathsf{update}_{l',v'}(x)) = \mathsf{update}_{l',v'}(\mathsf{update}_{l,v}(x)) \qquad (l \neq l')$$
$$\mathsf{update}_{l,v}(\mathsf{lookup}_l\,(x_{v'})_{v'}) = \mathsf{update}_{l,v}(x_v)$$
$$\mathsf{update}_{l,v}(\mathsf{lookup}_{l'}\,(x_{v'})_{v'}) = \mathsf{lookup}_{l'}\,(\mathsf{update}_{l,v}(x_{v'}))_{v'} \qquad (l \neq l')$$
$$\mathsf{lookup}_l\,(\mathsf{update}_{l,v}(x))_v = x$$

An algebra (a mnemoid) $\underline{A}$ is a tuple $(A, \mathsf{lookup}_{(\cdot)}, \mathsf{update}_{(\cdot),(\cdot)})$, where A is a set, and the functions $\mathsf{lookup}_{(\cdot)} : \mathsf{Loc} \times A^{\mathsf{Val}} \to A$ and $\mathsf{update}_{(\cdot),(\cdot)} : \mathsf{Loc} \times \mathsf{Val} \times A \to A$ satisfy the equations above.

The category of algebras

For any of the examples above, let **Alg** be the category whose objects are algebras and whose morphisms are homomorphisms.

There is a forgetful functor $U\colon \mathbf{Alg} \to \mathbf{Set}$ which maps an algebra $\underline{A}$ to its carrier set $U\underline{A}$.

A general construction of the free algebra FA over a set A (quotient the set of well-founded syntax trees over constants from A by the axiomatized equalities) gives a left adjoint $F\colon \mathbf{Set} \to \mathbf{Alg}$.

Proposition (characterisations of the monad UF).

1. For binary semilattices, $UF \cong \mathcal{P}_{\mathrm{fin}^+}$.
2. For convex sets, $UF \cong \mathrm{Dist}$.
3. For exceptions, $UF \cong (-) + E$.
4. For interactive output, $UF \cong U^* \times (-)$.
5. If Loc is finite, then, for mnemoids, $UF \cong S \to ((-) \times S)$ where $S = \mathrm{Loc} \to \mathrm{Val}$.

The category of algebras

For any of the examples above, let **Alg** be the category whose objects are algebras and whose morphisms are homomorphisms.

There is a forgetful functor $U : \mathbf{Alg} \to \mathbf{Set}$ which maps an algebra $\underline{A}$ to its carrier set $U\underline{A}$.

A general construction of the free algebra FA over a set A (quotient the set of well-founded syntax trees over constants from A by the axiomatized equalities) gives a left adjoint $F : \mathbf{Set} \to \mathbf{Alg}$.

Proposition (characterisations of the monad UF).

1. For binary semilattices, $UF \cong \mathcal{P}_{\mathrm{fin}^+}$.
2. For convex sets, $UF \cong \mathrm{Dist}$.
3. For exceptions, $UF \cong (-) + E$.
4. For interactive output, $UF \cong U^* \times (-)$.
5. If Loc is finite, then, for mnemoids, $UF \cong S \to ((-) \times S)$ where $S = \mathrm{Loc} \to \mathrm{Val}$.

Structure of **Alg** (limits)

The functor $U\colon \mathbf{Alg} \to \mathbf{Set}$ creates limits.

We shall be interested in this in two special cases.

Finite products: Given algebras

$$\underline{A} = (A, \{f_o\colon A^{\mathrm{arity}(o)} \to A\}_o) \text{ and } \underline{B} = (B, \{g_o\colon B^{\mathrm{arity}(o)} \to B\}_o),$$

define the product algebra $\underline{A} \times \underline{B}$:

$$(A \times B,\ \{(x_i, y_i)_{i \in \mathrm{arity}(o)} \mapsto (f_o\,(x_i)_i,\ g_o\,(y_i)_i)\}_o)$$

Powers: Given an algebra $\underline{A} = (A, \{f_o\colon A^{\mathrm{arity}(o)} \to A\}_o)$ and a set C define the power algebra $\underline{A}^C$:

$$(A^C,\ \{((x_{i,c})_{c \in C})_{i \in \mathrm{arity}(o)} \mapsto (f_o\,(x_{i,c})_i)_c\}_o)$$

Interpreting λ-calculus with effect operations

$$\frac{\Gamma \vdash t : \tau \quad \Gamma \vdash t' : \tau}{\Gamma \vdash t \text{ or } t' : \tau}$$

Call-by-value:

$$[\![t \text{ or } t']\!]^{\mathsf{vl}}_{\rho} = [\![t]\!]^{\mathsf{vl}}_{\rho} \text{ or } [\![t']\!]^{\mathsf{vl}}_{\rho}$$

(Uses the algebra structure on $T[\![\tau]\!]^{\mathsf{vl}}$.)

Call-by-name:

$$[\![t \text{ or } t']\!]^{\mathsf{nm}}_{\rho} = [\![t]\!]^{\mathsf{nm}}_{\rho} \text{ or } [\![t']\!]^{\mathsf{nm}}_{\rho}$$

(Uses the algebra structure on $[\![\tau]\!]^{\mathsf{nm}}$, for every type τ.)

Interpreting λ-calculus with effect operations

$$\frac{\Gamma \vdash t : \mathsf{Val} \quad \Gamma \vdash t' : \tau}{\Gamma \vdash (l := t)\,;\, t' : \tau} \qquad \frac{\Gamma, x{:}\mathsf{Val} \vdash t : \tau}{\Gamma \vdash \mathsf{let}\ x = !l\ \mathsf{in}\ t : \tau}$$

Call-by-value (revisited):

$$[\![(l := t)\,;\, t']\!]^{\mathsf{vl}}_\rho = (v \mapsto \mathsf{update}_{l,v}[\![t']\!]^{\mathsf{vl}}_\rho)^* ([\![t]\!]^{\mathsf{vl}}_\rho)$$
$$[\![\mathsf{let}\ x = !l\ \mathsf{in}\ t]\!]^{\mathsf{vl}}_\rho = \mathsf{lookup}_l ([\![t]\!]^{\mathsf{vl}}_{\rho[x:=v]})_v$$

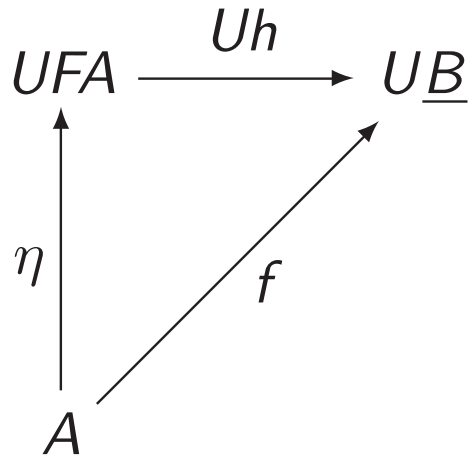
Call-by-name:

$$[\![(l := t)\,;\, t']\!]^{\mathsf{nm}}_\rho = (v \mapsto \mathsf{update}_{l,v}[\![t']\!]^{\mathsf{nm}}_\rho)^* ([\![t]\!]^{\mathsf{nm}}_\rho)$$
$$[\![\mathsf{let}\ x = !l\ \mathsf{in}\ t]\!]^{\mathsf{nm}}_\rho = \mathsf{lookup}_l ([\![t]\!]^{\mathsf{vl}}_{\rho[x:=\eta(v)]})_v$$

(Again uses the algebra structure on $[\![\tau]\!]^{\mathsf{nm}}$, for every type τ.)

Effect handlers

Free algebra property: For any algebra $\underline{B}$ and function $f : A \to U\underline{B}$, there exists a unique homomorphism $h : FA \to \underline{B}$ s.t.:



E.g., exceptions: An algebra is just a function $g : E \to B$ from the set E of exceptions to the underlying set B.
Given a function $f : A \to B$, the unique homomorphism is:

$$h[g, f] = P \mapsto \begin{cases} f(a) & \text{if } P = \text{in}_1(a) \\ g(e) & \text{if } P = \text{in}_2(e) \end{cases}$$

Exception handlers

Assume an atomic type Exn (of exceptions), thus $[\![Exn]\!] = E$.

$$\frac{\Gamma \vdash t : \sigma \quad \Gamma, x{:}\sigma \vdash t' : \tau \quad \Gamma, z{:}\mathsf{Exn} \vdash t'' : \tau}{\Gamma \ \vdash \ \mathsf{let}\ t \Rightarrow x\ \mathsf{in}\ t'\ \mathsf{handle}\ z.\, t''\ :\ \tau}$$

Call-by-value interpretation:

$$[\![\mathsf{let}\ t \Rightarrow x\ \mathsf{in}\ t'\ \mathsf{handle}\ z.\, t'']\!]^{\mathsf{vl}}_{\rho}$$
$$= h\left[e \mapsto [\![t'']\!]^{\mathsf{vl}}_{\rho[z:=e]},\ v \mapsto [\![t']\!]^{\mathsf{vl}}_{\rho[x:=v]}\right]([\![t]\!]^{\mathsf{vl}}_{\rho})$$

N.B., a call-by-name interpretation can only be given in the case that σ is a base type. (This ties in with known difficulties in incorporating exception handling into call-by-name languages such as Haskell.)

Advantages of algebraic view of effects

1. Monad derived from intuitive theory of equational properties of effect-triggering operations.
2. Properties of algebras exploited to give modular presentations of call-by-value and call-by-name semantics.
3. Universal property of monad gives rise to effect handlers.

Other advantages that there is no time to consider:

4. Operations for combining equational theories determine mechanisms for combining effects.
   (See papers by Hyland, Plotkin, Power & Levy)
5. Forms the basis of formal systems for reasoning about effects.
   (See "A logic for algebraic effects", Plotkin & Pretnar)

Exercises

1. Prove that $S \to (A \times S)$, where $S = \mathsf{Loc} \to \mathsf{Val}$, carries the structure of a mnemoid.

2. Suppose Loc is the finite set $\{l_1, \ldots, l_n\}$ and Val is finite. Prove that every term composed of operations lookup_{l_i} and $\mathsf{update}_{l_i,v}$ and variables is equal to one of the form

$$\mathsf{lookup}_{l_1}\Big(\ldots \mathsf{lookup}_{l_n}\Big(\mathsf{update}_{l_1,v_{\vec{v},1}}(\ldots \mathsf{update}_{l_n,v_{\vec{v},n}}(x_{\vec{v}})\ldots)\Big)_{v_n}\ldots\Big)_{v_1}$$

   where $\vec{v}$ is short for the vector $v_1, \ldots, v_n$ of values.

   Does the same property hold for well-founded syntax trees when *Val* is infinite but *Loc* is still finite?

   Do your answers require all 7 axioms for mnemoids?

3. Prove the isomorphisms of monads stated in the proposition on characterisations of *UF* monads. (Questions 1 and 2 above should help in the case of mnemoids.)

Lecture 3: A metalanguage for effects.

Further structure of **Alg** (colimits)

When **Alg** is the category of algebras for an equational theory over **Set** then **Alg** is cocomplete.

In general, colimits are awkward to construct and hard to visualise concretely. As with limits, we are interested in special cases.

Finite coproducts:

$$\underline{0} \qquad \underline{A} \oplus \underline{B}$$

We have $\underline{0} = F0$ and $(FA) \oplus (FB) \cong F(A + B)$.
Other instances of binary coproducts are harder to describe.

Copowers: Given an algebra $\underline{A}$ and a set C the copower algebra $C \bullet \underline{A}$ is the coproduct $\coprod_{c \in C} \underline{A}$ in **Alg**.

We have $C \bullet F1 \cong FC$.
General instances of copowers are again harder to describe.

The action of copowers

Copowers determine a functor

$$(-)\bullet(-)\colon \mathbf{Set} \times \mathbf{Alg} \to \mathbf{Alg}$$

that defines an action of **Set** (with its finite product structure) on **Alg**. (We will define what this means later.)

For every set A and algebra $\underline{B}$ the functors

$$A\bullet(-)\colon \mathbf{Alg} \to \mathbf{Alg} \qquad (-)\bullet\underline{B}\colon \mathbf{Set} \to \mathbf{Alg}$$

have right adjoints

$$(-)^A\colon \mathbf{Alg} \to \mathbf{Alg} \qquad \mathbf{Alg}(\underline{B},-)\colon \mathbf{Alg} \to \mathbf{Set}$$

demonstrated by the natural isomorphisms

$$\mathbf{Alg}(A\bullet\underline{B},\underline{C}) \;\cong\; \mathbf{Set}(A,\mathbf{Alg}(\underline{B},\underline{C})) \;\cong\; \mathbf{Alg}(\underline{B},\underline{C}^A)$$

An action on the Kleisli category

Let T be a strong monad on a category $\mathcal{V}$ with finite products.

Define a functor

$$(-)\bullet(-)\colon \mathcal{V} \times \mathcal{V}_T \to \mathcal{V}_T$$

by $A \bullet B = A \times B$ on objects, and, on morphisms $f\colon A \to A'$ and $g\colon B \to TB'$, define $f \bullet g$ to be:

$$A \times B \xrightarrow{f \times g} A' \times TB' \xrightarrow{t} T(A \times B')$$

This defines an action of $\mathcal{V}$ on $\mathcal{V}_T$.

If, for every object B, the functor $(-)\bullet B\colon \mathcal{V} \to \mathcal{V}_T$ has a right adjoint $B \to_T (-)\colon \mathcal{V}_T \to \mathcal{V}$, *viz*:

$$\mathcal{V}_T(A \times B, C) \;\cong\; \mathcal{V}(A, B \to_T C)$$

then we say that $\mathcal{V}$ has Kleisli exponentials (with respect to T).

Actions of categories

An action of a category $\mathcal{V}$ with finite products on a category $\mathcal{C}$ is a functor

$$(-)\bullet(-)\colon \mathcal{V}\times\mathcal{C}\to\mathcal{C}$$

Together with specified natural isomorphisms

$$1\bullet\underline{A}\xrightarrow{\cong}\underline{A}\qquad (A\times B)\bullet\underline{C}\xrightarrow{\cong}A\bullet(B\bullet\underline{C})$$

Making some coherence diagrams commute.

If, for every object $\underline{B}$ of $\mathcal{C}$, the functor $(-)\bullet\underline{B}\colon\mathcal{V}\to\mathcal{C}$ has a right adjoint $\underline{B}\multimap(-)$ then we say that the action is enriched.

If, for every object A of $\mathcal{V}$, the functor $A\bullet(-)\colon\mathcal{C}\to\mathcal{C}$ has a right adjoint $(-)^A$ then we say that the action has powers.

$$\mathcal{C}(A\bullet\underline{B},\underline{C})\ \cong\ \mathcal{V}(A,\underline{B}\multimap\underline{C})\ \cong\ \mathcal{C}(\underline{B},\underline{C}^A)$$

Enriched actions determine strong monads

Let $(-)\bullet(-)$ be an enriched action of $\mathcal{V}$ on $\mathcal{C}$.

Let $\underline{S}$ be an object of $\mathcal{C}$.

Since we have an adjunction

$$(-)\bullet\underline{S} \;\dashv\; \underline{S}\multimap(-) \;:\; \mathcal{C}\to\mathcal{V}$$

we have a monad $\underline{S}\multimap(-)\bullet\underline{S}$ on $\mathcal{V}$.

This monad is automatically strong.

It also has Kleisli exponentials, where $A\to_T B$ is defined by $A\bullet\underline{S}\multimap B\bullet\underline{S}$.

Any strong monad T on a category $\mathcal{V}$ with Kleisli exponentials can be obtained (isomorphically) as $\underline{S}\multimap(-)\bullet\underline{S}$, determined by an enriched action, by taking $\mathcal{C}$ to be $\mathcal{V}_T$ and defining $\underline{S}=1$.

Enriched actions determine strong monads

Let $(-)\bullet(-)$ be an enriched action of $\mathcal{V}$ on $\mathcal{C}$.

Let $\underline{S}$ be an object of $\mathcal{C}$.

Since we have an adjunction

$$(-)\bullet\underline{S} \;\dashv\; \underline{S}\multimap(-) \;:\; \mathcal{C}\to\mathcal{V}$$

we have a monad $\underline{S}\multimap(-)\bullet\underline{S}$ on $\mathcal{V}$.

This monad is automatically strong.

It also has Kleisli exponentials, where $A\to_T B$ is defined by $A\bullet\underline{S}\multimap B\bullet\underline{S}$.

Any strong monad T on a category $\mathcal{V}$ with Kleisli exponentials can be obtained (isomorphically) as $\underline{S}\multimap(-)\bullet\underline{S}$, determined by an enriched action, by taking $\mathcal{C}$ to be $\mathcal{V}_T$ and defining $\underline{S}=1$.

Free algebras via enriched actions

For a category **Alg** of algebras over **Set**, we have seen that the copower action $(-)\bullet(-)$ is enriched (with powers).

Define $\underline{S} = F1$.

Then $F \cong (-)\bullet\underline{S}$.

And $U \cong \underline{S} \multimap (-)$.

So the free algebra monad UF is isomorphic to $\underline{S} \multimap (-)\bullet\underline{S}$.

Another derivation of global state

Let $\mathcal{V}$ be a category with finite products.

Then $(-)\times(-)\colon \mathcal{V}\times\mathcal{V}\to\mathcal{V}$ is an action of $\mathcal{V}$ on itself.

The following are (trivially) equivalent.

- The action is enriched.
- The action has powers.
- The category is cartesian closed.

In which case, for any object S of $\mathcal{V}$, the adjunction

$$(-)\times S \dashv S\to(-) \;:\; \mathcal{V}\to\mathcal{V}$$

is a resolution of the state monad.

An alternative perspective

We now switch to viewing the choice of the category $\mathcal{V}$ and $\mathcal{C}$, an enriched action of $\mathcal{V}$ on $\mathcal{C}$, and an object $\underline{S}$ of $\mathcal{C}$ as the basic data.

$\mathcal{V}$ is called the category of values.

$\mathcal{C}$ is called the category of computations.

As above, this determines a strong monad $\underline{S} \multimap (-) \bullet \underline{S}$ on $\mathcal{V}$, which has Kleisli exponentials.

Moreover, any strong monad on $\mathcal{V}$ with Kleisli exponentials arises (isomorphically) in this way but not uniquely.

We shall sometimes assume further structure on $\mathcal{C}$: powers, products, coproducts.

An alternative perspective

We now switch to viewing the choice of the category $\mathcal{V}$ and $\mathcal{C}$, an enriched action of $\mathcal{V}$ on $\mathcal{C}$, and an object $\underline{S}$ of $\mathcal{C}$ as the basic data.

$\mathcal{V}$ is called the category of values.

$\mathcal{C}$ is called the category of computations.

As above, this determines a strong monad $\underline{S} \multimap (-) \bullet \underline{S}$ on $\mathcal{V}$, which has Kleisli exponentials.

Moreover, any strong monad on $\mathcal{V}$ with Kleisli exponentials arises (isomorphically) in this way but not uniquely.

We shall sometimes assume further structure on $\mathcal{C}$: powers, products, coproducts.

Enriched call-by-value calculus [Møgelberg & Staton]

We generate value types $A, B, C, \ldots$ and computation types $\underline{A}, \underline{B}, \underline{C}, \ldots$ from a set of value-type constants, ranged over by α, and computation-type constants, ranged over by $\underline{\alpha}$.

$$A ::= \alpha \mid 1 \mid A \times B \mid \underline{A} \multimap \underline{B}$$
$$\underline{A} ::= \underline{\alpha} \mid A \bullet \underline{B}$$

Terms are defined using a formal system for two judgement forms

$$\Gamma \vdash t : A \qquad\qquad \Gamma \mid z : \underline{A} \vdash t : \underline{B}$$

where Γ is a sequence of type assignments, $x : C$, mapping variables x to value types C.

$$\frac{}{\Gamma \vdash x : A}\; x : A \in \Gamma \qquad \frac{}{\Gamma \mid z : \underline{A} \vdash z : \underline{A}}$$

$$\frac{}{\Gamma \vdash * : 1}$$

$$\frac{\Gamma \vdash t_1 : A_1 \qquad \Gamma \vdash t_2 : A_2}{\Gamma \vdash (t_1, t_2) : A_1 \times A_2} \qquad \frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash \pi_i(t) : A_i}$$

$$\frac{\Gamma \mid z : \underline{A} \vdash t : \underline{B}}{\Gamma \vdash \underline{\lambda} z : \underline{A}.\, t : \underline{A} \multimap \underline{B}} \qquad \frac{\Gamma \vdash t : \underline{A} \multimap \underline{B} \qquad \Gamma \mid z : \underline{C} \vdash t' : \underline{A}}{\Gamma \mid z : \underline{C} \vdash t\,[t'] : \underline{B}}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \mid z : \underline{C} \vdash t' : \underline{B}}{\Gamma \mid z : \underline{C} \vdash t \bullet t' : A \bullet \underline{B}}$$

$$\frac{\Gamma \mid z : \underline{D} \vdash t : A \bullet \underline{B} \qquad \Gamma, x : A \mid y : \underline{B} \vdash t' : \underline{C}}{\Gamma \mid z : \underline{D} \vdash \mathsf{let}\ t \Rightarrow x \bullet y\ \mathsf{in}\ t' : \underline{C}}$$

Interpretation of the calculus

Let $(-)\bullet(-)$ be an enriched action of a category $\mathcal{V}$ with finite products on a category $\mathcal{C}$.

We interpret types as

$$[\![A]\!] \text{ an object of } \mathcal{V} \qquad [\![\underline{A}]\!] \text{ an object of } \mathcal{C}$$

defined by extending interpretations of type constants in the obvious way ($[\![A \bullet \underline{B}]\!]$ using the action, $[\![\underline{A} \multimap \underline{B}]\!]$ using enrichment).

A context $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ is interpreted as the product in $\mathcal{V}$

$$[\![\Gamma]\!] = [\![A_n]\!] \times (\cdots \times [\![A_1]\!])$$

Then terms have interpretations

$$\Gamma \vdash t : A \quad \text{interpreted as} \quad [\![\Gamma]\!] \xrightarrow{[\![t]\!]} [\![A]\!] \quad \text{in } \mathcal{V}$$

$$\Gamma \mid z : \underline{A} \vdash t : \underline{B} \quad \text{interpreted as} \quad [\![\Gamma]\!] \bullet [\![\underline{A}]\!] \xrightarrow{[\![t]\!]} [\![\underline{B}]\!] \quad \text{in } \mathcal{C}$$

Call-by-value translation of typed λ-calculus

We translate simply-typed λ-calculus to the enriched call-by-value calculus. The translation uses a computation-type constant $\underline{S}$ (intuitively, a state object).

A λ-calculus type τ is translated to a value type $\langle\!\langle \tau \rangle\!\rangle^{\mathsf{vl}}$ by

$$\begin{aligned} \langle\!\langle \alpha \rangle\!\rangle^{\mathsf{vl}} &= \alpha \\ \langle\!\langle \sigma \to \tau \rangle\!\rangle^{\mathsf{vl}} &= \langle\!\langle \sigma \rangle\!\rangle^{\mathsf{vl}} \bullet \underline{S} \multimap \langle\!\langle \tau \rangle\!\rangle^{\mathsf{vl}} \bullet \underline{S} \end{aligned}$$

A λ-calculus term $\Gamma \vdash M : \tau$ translates to an enriched cbv term

$$\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{vl}} \vdash \langle\!\langle M \rangle\!\rangle^{\mathsf{vl}} : \underline{S} \multimap \langle\!\langle \tau \rangle\!\rangle^{\mathsf{vl}} \bullet \underline{S}$$

$$\begin{aligned} \langle\!\langle x \rangle\!\rangle^{\mathsf{vl}} &= \underline{\lambda} s : \underline{S}.\ x \bullet s \\ \langle\!\langle M\, N \rangle\!\rangle^{\mathsf{vl}} &= \underline{\lambda} s : \underline{S}.\ \mathsf{let}\ \langle\!\langle M \rangle\!\rangle^{\mathsf{vl}}\ [s] \Rightarrow f \bullet s'\ \mathsf{in}\ \mathsf{let}\ \langle\!\langle N \rangle\!\rangle^{\mathsf{vl}}\ [s'] \Rightarrow x \bullet s''\ \mathsf{in}\ f\ [x \bullet s''] \\ \langle\!\langle \lambda x : \sigma.\ M \rangle\!\rangle^{\mathsf{vl}} &= \underline{\lambda} s : \underline{S}.\ (\underline{\lambda} z : \langle\!\langle \sigma \rangle\!\rangle^{\mathsf{vl}} \bullet \underline{S}.\ \mathsf{let}\ z \Rightarrow x \bullet s'\ \mathsf{in}\ \langle\!\langle M \rangle\!\rangle^{\mathsf{vl}}\ [s']) \bullet s \end{aligned}$$

Enriched effect calculus, cf. [Egger,Møgelberg & S.]

Add new value and computation types.

$$A ::= \alpha \mid 1 \mid A \times B \mid \underline{A} \multimap \underline{B} \mid 0 \mid A + B$$
$$\underline{A} ::= \underline{\alpha} \mid A \bullet \underline{B} \mid \underline{1} \mid \underline{A} \,\&\, \underline{B} \mid A \to \underline{B} \mid \underline{0} \mid \underline{A} \oplus \underline{B}$$

The enriched cbv calculus requires an enriched action of a category $\mathcal{V}$ with finite products on a category $\mathcal{C}$.

The red constructs require $\mathcal{C}$ to have finite products and the action to have powers.

The blue constructs require $\mathcal{V}$ to be distributive.

The purple constructs require $\mathcal{C}$ to have finite coproducts over which the action distributes (automatic when we have powers):

$$A \bullet \underline{0} \cong \underline{0}$$
$$A \bullet (\underline{B} \oplus \underline{C}) \cong (A \bullet \underline{B}) \oplus (A \bullet \underline{C})$$

Rules for the red constructs.

$$\frac{}{\Gamma \mid z : \underline{C} \vdash * : \underline{1}}$$

$$\frac{\Gamma \mid z : \underline{C} \vdash t_1 : \underline{A_1} \qquad \Gamma \mid z : \underline{C} \vdash t_2 : \underline{A_2}}{\Gamma \mid z : \underline{C} \vdash (t_1, t_2) : \underline{A_1} \,\&\, \underline{A_2}} \qquad \frac{\Gamma \mid z : \underline{C} \vdash t : \underline{A_1} \,\&\, \underline{A_2}}{\Gamma \mid z : \underline{C} \vdash \pi_i(t) : \underline{A_i}}$$

$$\frac{\Gamma, x : A \mid z : \underline{C} \vdash t : \underline{B}}{\Gamma \mid z : \underline{C} \vdash \lambda x : A.\, t : A \to \underline{B}} \qquad \frac{\Gamma \mid z : \underline{C} \vdash t : A \to \underline{B} \qquad \Gamma \vdash t' : A}{\Gamma \mid z : \underline{C} \vdash t\, t' : \underline{B}}$$

Rules for the purple constructs.

$$\frac{\Gamma \mid z : \underline{C} \vdash t : \underline{0}}{\Gamma \mid z : \underline{C} \vdash ?_{\underline{A}}[t] : \underline{A}} \qquad \frac{\Gamma \mid z : \underline{C} \vdash t_i : \underline{A}_i}{\Gamma \mid z : \underline{C} \vdash \underline{\mathsf{in}}_i[t_i] : \underline{A}_1 \oplus \underline{A}_2}$$

$$\frac{\Gamma \mid z : \underline{C} \vdash t : \underline{A}_1 \oplus \underline{A}_2 \quad \Gamma \mid z_1 : \underline{A}_1 \vdash t_1 : \underline{B} \quad \Gamma \mid z_2 : \underline{A}_2 \vdash t_2 : \underline{B}}{\Gamma \mid z : \underline{C} \vdash \underline{\mathsf{case}}\ t\ \mathsf{of}\ (\underline{\mathsf{in}}_1[z_1].\, t_1 \mid \underline{\mathsf{in}}_2[z_2].\, t_2) : \underline{B}}$$

Call-by-name translation of typed λ-calculus

A λ-calculus type τ is translated to a computation type $\underline{\langle\!\langle \tau \rangle\!\rangle}^{\mathsf{nm}}$ by

$$\begin{aligned} \underline{\langle\!\langle \alpha \rangle\!\rangle}^{\mathsf{nm}} &= \alpha \bullet \underline{S} \\ \underline{\langle\!\langle \sigma \to \tau \rangle\!\rangle}^{\mathsf{nm}} &= (\underline{S} \multimap \underline{\langle\!\langle \sigma \rangle\!\rangle}^{\mathsf{nm}}) \to \underline{\langle\!\langle \tau \rangle\!\rangle}^{\mathsf{nm}} \end{aligned}$$

A λ-calculus term $\Gamma \vdash M : \tau$ translates to an enriched-effect-calculus term

$$\underline{S} \multimap \langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{nm}} \vdash \langle\!\langle M \rangle\!\rangle^{\mathsf{nm}} : \underline{S} \multimap \underline{\langle\!\langle \tau \rangle\!\rangle}^{\mathsf{nm}}$$

$$\begin{aligned} \langle\!\langle x \rangle\!\rangle^{\mathsf{nm}} &= x \\ \langle\!\langle M\, N \rangle\!\rangle^{\mathsf{nm}} &= \underline{\lambda} s : \underline{S}.\ (\langle\!\langle M \rangle\!\rangle^{\mathsf{nm}}\, [s])\, \langle\!\langle N \rangle\!\rangle^{\mathsf{nm}} \\ \langle\!\langle \lambda x : \sigma.\, M \rangle\!\rangle^{\mathsf{nm}} &= \underline{\lambda} s : \underline{S}.\ \lambda x : \underline{S} \multimap \underline{\langle\!\langle \sigma \rangle\!\rangle}^{\mathsf{nm}}.\ \langle\!\langle M \rangle\!\rangle^{\mathsf{nm}}\, [s] \end{aligned}$$

Other points to make

1. The composition, translation into enriched effect caclulus followed by denotational interpretation of eec, factorises the cbv and cbn denotational interpretations of λ-caclulus.

2. A natural equational theory between terms allows one to prove equalities between programs with effects.

3. There is a nice treatment of effect-triggering operations in the enriched effect calculus (cf. [Møgelberg and Staton])

4. A polymorphic version allows all type constructors to be reduced to a few primitives: $\multimap$, $\to$ (defined on value types), and universal quantification over both value and computation types (cf. [Møgelberg and S.]).

Exercises

1. If T is a strong monad on a cartesian-closed category $\mathcal{V}$ then verify that $\mathcal{V}$ has Kleisli exponentials.

2. Verify that $(-)\bullet(-) : \mathcal{V} \times \mathcal{V}_T \to \mathcal{V}_T$ indeed defines an action of $\mathcal{V}$ on $\mathcal{V}_T$.

3. Prove that the monad $\underline{S} \multimap (-)\bullet\underline{S}$ determined by an enriched action of $\mathcal{V}$ on $\mathcal{C}$ is strong. (You may need to look up the definition of strong monad on-line.)

4. Write out typing rules for finite coproducts for value types.

5. Define call-by-value and call-by-name translations of typed λ-calculus with finite products into the enriched effect calculus.

   Then consider finite coproducts.

Lecture 4: Continuations and control

Kleisli category for continuations

Throughout this lecture, $\mathcal{V}$ is a cartesian-closed category.

An object R defines the continuations monad $TA = (A \to R) \to R$ on $\mathcal{V}$.

This monad is automatically strong:

$$A \times (B \to R) \to R \xrightarrow{(a,P) \mapsto \lambda p.\, P\,(\lambda b.\, p(a,b))} ((A \times B) \to R) \to R$$

Its Kleisli category has a simple reformulation

$$\frac{A \longrightarrow (B \to R) \to R}{(B \to R) \longrightarrow (A \to R)}$$

Programs are continuation transformers.

The opposite of the Kleisli category

The category ${\mathcal{V}_T}^{\mathrm{op}}$ has as morphisms from A to B, forward transformers in $\mathcal{V}$.

$$(A \to R) \longrightarrow (B \to R)$$

We henceforth assume that $\mathcal{V}$ has finite coproducts.

Proposition. ${\mathcal{V}_T}^{\mathrm{op}}$ is cartesian closed.

Proof. 0 in $\mathcal{V}$ is the terminal object in ${\mathcal{V}_T}^{\mathrm{op}}$.

$A + B$ in $\mathcal{V}$ is the product of A and B in ${\mathcal{V}_T}^{\mathrm{op}}$.

$(A \to R) \times B$ in $\mathcal{V}$ is the closed structure for ${\mathcal{V}_T}^{\mathrm{op}}(A, B)$

The opposite of the Kleisli category

The category ${\mathcal{V}_T}^{\mathrm{op}}$ has as morphisms from A to B, forward transformers in $\mathcal{V}$.

$$(A \to R) \longrightarrow (B \to R)$$

We henceforth assume that $\mathcal{V}$ has finite coproducts.

Proposition. ${\mathcal{V}_T}^{\mathrm{op}}$ is cartesian closed.

Proof. 0 in $\mathcal{V}$ is the terminal object in ${\mathcal{V}_T}^{\mathrm{op}}$.

$A + B$ in $\mathcal{V}$ is the product of A and B in ${\mathcal{V}_T}^{\mathrm{op}}$.

$(A \to R) \times B$ in $\mathcal{V}$ is the closed structure for ${\mathcal{V}_T}^{\mathrm{op}}(A, B)$

The opposite of the Kleisli category

The category $\mathcal{V}_T{}^{\mathrm{op}}$ has as morphisms from A to B, forward transformers in $\mathcal{V}$.

$$(A \to R) \longrightarrow (B \to R)$$

We henceforth assume that $\mathcal{V}$ has finite coproducts.

Proposition. $\mathcal{V}_T{}^{\mathrm{op}}$ is cartesian closed.

Proof. 0 in $\mathcal{V}$ is the terminal object in $\mathcal{V}_T{}^{\mathrm{op}}$.

$A + B$ in $\mathcal{V}$ is the product of A and B in $\mathcal{V}_T{}^{\mathrm{op}}$.

$(A \to R) \times B$ in $\mathcal{V}$ is the closed structure for $\mathcal{V}_T{}^{\mathrm{op}}(A, B)$

Verification of closed structure.

$$
\begin{aligned}
&\mathcal{V}_T{}^{\mathrm{op}}(C+A,\, B) \\
&\qquad \cong \mathcal{V}((C+A)\to R,\, B\to R) \\
&\qquad \cong \mathcal{V}((C\to R)\times(A\to R),\, B\to R) \\
&\qquad \cong \mathcal{V}((C\to R),\, (A\to R)\to (B\to R)) \\
&\qquad \cong \mathcal{V}((C\to R),\, ((A\to R)\times B)\to R) \\
&\qquad \cong \mathcal{V}_T{}^{\mathrm{op}}(C,\, (A\to R)\times B)
\end{aligned}
$$

Enriched action model of continuations

An action of $\mathcal{V}$ on $\mathcal{V}^{\mathrm{op}}$ is given by:

$$A \bullet \underline{B} \;=\; A \to \underline{B}$$

The action is enriched with $\underline{B} \multimap (-) = (-) \to \underline{B}$.

It also has powers with $(-)^A = A \times (-)$.

For an object $\underline{R}$ of $\mathcal{V}^{\mathrm{op}}$, the adjunction $(-) \bullet \underline{R} \dashv \underline{R} \multimap (-)$ is:

$$(-) \to \underline{R} \;\dashv\; (-) \to \underline{R} \;:\; \mathcal{V}^{\mathrm{op}} \to \mathcal{V}$$

which is a resolution of the continuations monad.

Furthermore, $\mathcal{V}^{\mathrm{op}}$ has finite products (coproducts in $\mathcal{V}$) and coproducts (products in $\mathcal{V}$).

Thus we obtain a model of the full enriched effect calculus.

Call-by-name and call-by-value (in general)

Consider the call-by-name and call-by-value interpretation of typed λ-calculus:

$$[\![-]\!]^{\mathsf{vl}} = [\![\langle\!\langle - \rangle\!\rangle^{\mathsf{vl}}]\!] \qquad [\![-]\!]^{\mathsf{nm}} = [\![\langle\!\langle - \rangle\!\rangle^{\mathsf{nm}}]\!]$$

In any enriched cbv model, the cbv interpretation gives $[\![\sigma \to \tau]\!]^{\mathsf{vl}}$ as a Kleisli exponential $[\![\sigma]\!]^{\mathsf{vl}} \bullet \underline{S} \multimap [\![\tau]\!]^{\mathsf{vl}} \bullet \underline{S}$, and a term $\Gamma \vdash M : \tau$ is interpreted as a map $[\![M]\!]^{\mathsf{vl}} : [\![\Gamma]\!]^{\mathsf{vl}} \to T[\![\tau]\!]^{\mathsf{vl}}$ in $\mathcal{V}$, i.e., a map from $[\![\Gamma]\!]^{\mathsf{vl}}$ to $[\![\tau]\!]^{\mathsf{vl}}$ in $\mathcal{V}_T$

In any eec model, the cbn interpretation interprets $\underline{[\![\sigma \to \tau]\!]}^{\mathsf{nm}}$ as the power $(\underline{[\![\tau]\!]}^{\mathsf{nm}})^{(\underline{S} \multimap \underline{[\![\sigma]\!]}^{\mathsf{nm}})}$. And a term $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \tau$ is interpreted as a map in $\mathcal{V}$:

$$[\![M]\!]^{\mathsf{nm}} : (\underline{S} \multimap \underline{[\![\sigma_1]\!]}^{\mathsf{nm}}) \times \cdots \times (\underline{S} \multimap \underline{[\![\sigma_n]\!]}^{\mathsf{nm}}) \longrightarrow (\underline{S} \multimap \underline{[\![\tau]\!]}^{\mathsf{nm}})$$

Duality of call-by-name and call-by-value

When we instantiate the call-by-name interpretation in the special case of the enriched-action model of continuations (in which we write $\underline{R}$ instead of $\underline{S}$), we obtain:

$$\underline{[\![\sigma \to \tau]\!]}^{\mathrm{nm}} \;=\; (\underline{[\![\sigma]\!]}^{\mathrm{nm}} \to \underline{R}) \times \underline{[\![\tau]\!]}^{\mathrm{nm}}$$

and $[\![M]\!]^{\mathrm{nm}}$ transports along isomorphisms to a map in $\mathcal{V}$:

$$(\underline{[\![\sigma_1]\!]}^{\mathrm{nm}} + \cdots + \underline{[\![\sigma_n]\!]}^{\mathrm{nm}}) \to \underline{R} \;\longrightarrow\; \underline{[\![\tau]\!]}^{\mathrm{nm}} \to \underline{R}$$

This results in the standard interpretation of M using the cartesian-closed structure of $\mathcal{V}_T{}^{\mathrm{op}}$.

As on the last slide, in the general case (not just for continuations), call-by-value is interpreted in $\mathcal{V}_T$.

So, for continuations, call-by-name is dual to call-by-value.

An isomorphism in the model

In the enriched-effect-calculus we have a closed term

$$\underline{\lambda} z : \underline{A}.\ \underline{\lambda} p : \underline{A} \multimap \underline{0}.\ p\,[z] \ \ :\ \ \underline{A} \multimap ((\underline{A} \multimap \underline{0}) \to \underline{0})$$

In the continuations model this is an isomorphism!

We add a constant for its inverse

$$\mathsf{dne} \ \ :\ \ ((\underline{A} \multimap \underline{0}) \to \underline{0}) \multimap \underline{A}$$

(dne for "double-negation elimination".)

Interpreting control

A famous control operator.

$$\frac{\Gamma,\ k : \sigma \to \tau \vdash\ \ t : \sigma}{\Gamma \vdash\ \ \text{callcc}\ k : \sigma \to \tau\ \text{in}\ t : \sigma}$$

Using dne, we can extend the call-by-value translation of typed λ-calculus into eec to include callcc.

$$\begin{aligned}
&\langle\!\langle \text{callcc}\ k : \sigma \to \tau\ \text{in}\ t \rangle\!\rangle^{\mathsf{vl}} = \\
&\qquad \underline{\lambda} r : \underline{R}. \\
&\qquad\qquad \text{dne}\big(\, \lambda p : \langle\!\langle \sigma \rangle\!\rangle^{\mathsf{vl}} \bullet \underline{R} \multimap \underline{0}. \\
&\qquad\qquad\qquad (\lambda k : \langle\!\langle \sigma \rangle\!\rangle^{\mathsf{vl}} \bullet \underline{R} \multimap \langle\!\langle \tau \rangle\!\rangle^{\mathsf{vl}} \bullet \underline{R}.\ p\,[\langle\!\langle t \rangle\!\rangle^{\mathsf{vl}}\,[r]]) \\
&\qquad\qquad\qquad\qquad (\underline{\lambda} y : \langle\!\langle \sigma \rangle\!\rangle^{\mathsf{vl}} \bullet \underline{R}.\ ?_{\langle\!\langle \tau \rangle\!\rangle^{\mathsf{vl}} \bullet \underline{R}}\,[p\,[y]])\ \big)
\end{aligned}$$

(callcc for "call with current continuation".)

Exercises

1. Verify the correctness of the definition of the strength of the continuations monad. (You may need to look up the definition of strong monad on-line.)

2. Show in detail that the canonical term of type $\underline{A} \multimap ((\underline{A} \multimap \underline{0}) \to \underline{0})$ is an isomorphism in the continuation model.

3. Is it possible for the canonical term of type $\underline{A} \multimap ((\underline{A} \multimap \underline{0}) \to \underline{0})$ to be an isomorphism, for every type $\underline{A}$, in intuitionistic linear logic (where $\underline{0}$ is the unit for $\oplus$)?

4. Verify that the eec term $\langle\!\langle \mathsf{callcc}\ x : \sigma \to \tau\ \mathsf{in}\ t \rangle\!\rangle^{\mathsf{vl}}$, translating callcc into eec with dne, has the correct type.

5. Use dne to give a call-by-name translation of callcc.

Lecture 5: Functor category models of local store.

Local resources

A common feature (indeed effect) in programming languages is the creation of a fresh resource (local identifier, "name", private key for encryption, etc.).

At the time it is created, such a resource is private to the process that generates it.

However, it may be shared by explicitly passing it.

If it is not explicitly passed then it remains hidden.

In this lecture, we look at this in the context of local store.

Local resources

A common feature (indeed effect) in programming languages is the creation of a fresh resource (local identifier, "name", private key for encryption, etc.).

At the time it is created, such a resource is private to the process that generates it.

However, it may be shared by explicitly passing it.

If it is not explicitly passed then it remains hidden.

In this lecture, we look at this in the context of local store.

Local store

Add a type Loc of local locations.

$$\frac{\Gamma \vdash L : \mathsf{Loc} \qquad \Gamma \vdash M : \mathsf{Val} \qquad \Gamma \vdash N : \tau}{\Gamma \;\vdash\; (L := M)\,;\, N \;:\; \tau}$$

$$\frac{\Gamma \vdash L : \mathsf{Loc} \qquad \Gamma,\, x : \mathsf{Val} \vdash M : \tau}{\Gamma \;\vdash\; \mathsf{let}\; x = !L\; \mathsf{in}\; M \;:\; \tau}$$

$$\frac{\Gamma \vdash M : \mathsf{Val} \qquad \Gamma,\, l : \mathsf{Loc} \vdash N : \tau}{\Gamma \;\vdash\; \mathsf{new}\; l := M\; \mathsf{in}\; N \;:\; \tau}$$

$$\frac{\Gamma \vdash L_1 : \mathsf{Loc} \qquad \Gamma \vdash L_2 : \mathsf{Loc}}{\Gamma \;\vdash\; L_1 = L_2 \;:\; \mathsf{bool}}$$

Example programs

For convenience, we store natural numbers, i.e., $\mathit{Val} = \mathbb{N}$.

Consider

$$\text{new } l := 0 \text{ in } l \;:\; \text{Loc}$$

If we run this in current state

$$s : \{l_1, \ldots, l_n\} \to \text{Val}$$

then the program creates a new location l_{n+1} and returns this location in the state:

$$s[l_{n+1} := 0] : \{l_1, \ldots, l_n, l_{n+1}\} \to \text{Val}$$

Consider

$$\text{new } l := 0 \text{ in new } l' := 1 \text{ in } l \;:\; \text{Loc}$$

When run in current state $s\colon \{l_1, \ldots, l_n\} \to \text{Val}$, the program creates new locations l_{n+1}, l_{n+2} and returns l_{n+1} in the state:

$$s[l_{n+1} := 0, l_{n+2} := 1]\colon \{l_1, \ldots, l_n, l_{n+1}, l_{n+2}\} \to \text{Val}$$

However, since the program does not disclose the identity of l', this location can never be used by any program context containing the above program. Thus the program above is behaviourally equivalent to the one on the previous slide:

$$\text{new } l := 0 \text{ in } l \;:\; \text{Loc}$$

Garbage collection!

Consider

$$\text{new } l := 0 \text{ in new } l' := 1 \text{ in } l \;:\; \text{Loc}$$

When run in current state $s\colon \{l_1, \ldots, l_n\} \to \text{Val}$, the program creates new locations l_{n+1}, l_{n+2} and returns l_{n+1} in the state:

$$s[l_{n+1} := 0, l_{n+2} := 1]\colon \{l_1, \ldots, l_n, l_{n+1}, l_{n+2}\} \to \text{Val}$$

However, since the program does not disclose the identity of l', this location can never be used by any program context containing the above program. Thus the program above is behaviourally equivalent to the one on the previous slide:

$$\text{new } l := 0 \text{ in } l \;:\; \text{Loc}$$

Garbage collection!

Consider

$$\text{new } l := 0 \text{ in new } l' := 1 \text{ in } (l, l') \ : \ \text{Loc} \times \text{Loc}$$

If run in current state $s\colon \{l_1, \ldots, l_n\} \to \text{Val}$, the program creates new locations l_{n+1}, l_{n+2} and returns (l_{n+1}, l_{n+2}) in the state:

$$s[l_{n+1} := 0, l_{n+2} := 1]\colon \{l_1, \ldots, l_n, l_{n+1}, l_{n+2}\} \to \text{Val}$$

Although this program divulges both l and l', it is impossible for any containing program context to determine the order in which they were created. Thus the above program is equivalent to:

$$\text{new } l := 1 \text{ in new } l' := 0 \text{ in } (l', l) \ : \ \text{Loc} \times \text{Loc}$$

But it is (of course) distinguishable from:

$$\text{new } l := 1 \text{ in new } l' := 0 \text{ in } (l, l') \ : \ \text{Loc} \times \text{Loc}$$

Consider

$$\text{new } l := 0 \text{ in new } l' := 1 \text{ in } (l, l') \ : \ \text{Loc} \times \text{Loc}$$

If run in current state $s\colon \{l_1, \ldots, l_n\} \to \text{Val}$, the program creates new locations l_{n+1}, l_{n+2} and returns (l_{n+1}, l_{n+2}) in the state:

$$s[l_{n+1} := 0, l_{n+2} := 1]\colon \{l_1, \ldots, l_n, l_{n+1}, l_{n+2}\} \to \text{Val}$$

Although this program divulges both l and l', it is impossible for any containing program context to determine the order in which they were created. Thus the above program is equivalent to:

$$\text{new } l := 1 \text{ in new } l' := 0 \text{ in } (l', l) \ : \ \text{Loc} \times \text{Loc}$$

But it is (of course) distinguishable from:

$$\text{new } l := 1 \text{ in new } l' := 0 \text{ in } (l, l') \ : \ \text{Loc} \times \text{Loc}$$

The category $\mathbf{I}$

At any stage in computation the world of locations in current usage will be finite. But it can grow arbitrarily.

Let $\mathbf{I}$ be the category with:

- objects: finite subsets U of a fixed infinite set (of potential locations);
- morphisms from U to V: injective (i.e., one-to-one) functions.

The idea is that morphisms in $\mathbf{I}$ represent permissible means of extending the current world to another world.

The possibility of renaming locations reflects the fact that we do not care which name (or address) the implemention assigns to the location.

The requirement of injectivity reflects that existing distinct locations remain distinct.

Types as functors

At a world $U = \{l_1, \ldots, l_n\}$ the possible values of type *Loc* are just the local locations, i.e., $l_1, \ldots, l_n$.

If we transform world by $i : U \rightarrowtail V$, then a location l gets transformed to $i(l)$.

We thus interpret the type Loc as the inclusion functor

$$\mathsf{Loc} \ : \ \mathbf{I} \rightarrow \mathbf{Set}$$

In general, the interpretation $[\![\sigma]\!]$ of a type σ needs to assign a set of values $[\![\sigma]\!](U)$ dependent on the world U.

Moreover, a transformation $i\colon U \rightarrowtail V$ of worlds needs to transport values

$$[\![\sigma]\!](i) \;:\; [\![\sigma]\!](U) \to [\![\sigma]\!](V)$$

functorially!

Thus types are interpreted as objects of the functor category

$$[\mathbf{I}, \mathbf{Set}]$$

Examples: Loc as on previous slide
Val $:\; U \mapsto \mathbb{N}$

A major aim of this lecture is to present the local store monad on the functor category $[\mathbf{I}, \mathbf{Set}]$.

In general, the interpretation $[\![\sigma]\!]$ of a type σ needs to assign a set of values $[\![\sigma]\!](U)$ dependent on the world U.

Moreover, a transformation $i\colon U \rightarrowtail V$ of worlds needs to transport values

$$[\![\sigma]\!](i) \;:\; [\![\sigma]\!](U) \to [\![\sigma]\!](V)$$

functorially!

Thus types are interpreted as objects of the functor category

$$[\mathbf{I}, \mathbf{Set}]$$

Examples: Loc as on previous slide
Val $:\; U \mapsto \mathbb{N}$

A major aim of this lecture is to present the local store monad on the functor category $[\mathbf{I}, \mathbf{Set}]$.

In general, the interpretation $[\![\sigma]\!]$ of a type σ needs to assign a set of values $[\![\sigma]\!](U)$ dependent on the world U.

Moreover, a transformation $i\colon U \rightarrowtail V$ of worlds needs to transport values

$$[\![\sigma]\!](i) \;:\; [\![\sigma]\!](U) \to [\![\sigma]\!](V)$$

functorially!

Thus types are interpreted as objects of the functor category

$$[\mathbf{I}, \mathbf{Set}]$$

Examples: Loc as on previous slide

Val : $U \mapsto \mathbb{N}$

A major aim of this lecture is to present the local store monad on the functor category $[\mathbf{I}, \mathbf{Set}]$.

In general, the interpretation $[\![\sigma]\!]$ of a type σ needs to assign a set of values $[\![\sigma]\!](U)$ dependent on the world U.

Moreover, a transformation $i\colon U \rightarrowtail V$ of worlds needs to transport values

$$[\![\sigma]\!](i) \;:\; [\![\sigma]\!](U) \to [\![\sigma]\!](V)$$

functorially!

Thus types are interpreted as objects of the functor category

$$[\mathbf{I}, \mathbf{Set}]$$

Examples: Loc as on previous slide

Val : $U \mapsto \mathbb{N}$

A major aim of this lecture is to present the local store monad on the functor category $[\mathbf{I}, \mathbf{Set}]$.

The state presheaf

The set of possible local states at world U is

$$\underline{S}(U) \;=\; U \to \mathbb{N}$$

This is a contravariant functor (presheaf)

$$\underline{S} \;:\; \mathbf{I}^{\mathrm{op}} \to \mathbf{Set}$$
$$\underline{S}(i)(s) \;=\; s \circ i \qquad\qquad \text{for } i : U \rightarrowtail V$$

So we need to work with two categories $[\mathbf{I}, \mathbf{Set}]$ and $[\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$.

The state presheaf

The set of possible local states at world U is

$$\underline{S}(U) \;=\; U \to \mathbb{N}$$

This is a contravariant functor (presheaf)

$$\begin{aligned} \underline{S} \;&:\; \mathbf{I}^{\mathrm{op}} \to \mathbf{Set} \\ \underline{S}(i)(s) \;&=\; s \circ i \qquad\qquad \text{for } i \colon U \rightarrowtail V \end{aligned}$$

So we need to work with two categories $[\mathbf{I}, \mathbf{Set}]$ and $[\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$.

The state presheaf

The set of possible local states at world U is

$$\underline{S}(U) \;=\; U \to \mathbb{N}$$

This is a contravariant functor (presheaf)

$$\begin{aligned} \underline{S} \;&:\; \mathbf{I}^{\mathrm{op}} \to \mathbf{Set} \\ \underline{S}(i)(s) \;&=\; s \circ i \qquad\qquad \text{for } i \colon U \rightarrowtail V \end{aligned}$$

So we need to work with two categories $[\mathbf{I}, \mathbf{Set}]$ and $[\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$.

Motivating the monad

We look at how to model possible behaviours in a world U.

$$\text{new } l := 0 \text{ in } l \;:\; \text{Loc}$$

A possible behaviour: map the start state $s \in \underline{S}(U)$ to:

$$\begin{array}{rcll} i & : & U \rightarrowtail U \uplus \{l\} & \text{create a new location } l \\ l & \in & \text{Loc}(U \uplus \{l\}) & \text{result returned by computation} \\ s[l := 0] & \in & \underline{S}(U \uplus \{l\}) & \text{result state} \end{array}$$

Consider

$$\text{new } l := 0 \text{ in new } l' := 1 \text{ in } l \;:\; \mathsf{Loc}$$

A possible behaviour: map the start state $s \in \underline{S}(U)$ to:

$$\begin{array}{rcll} i' & : & U \rightarrowtail U \uplus \{l, l'\} & \text{create two new locations } l, l' \\ l & \in & \mathsf{Loc}(U \uplus \{l, l'\}) & \text{result of computation} \\ s[l := 0, l' := 1] & \in & \underline{S}(U \uplus \{l, l'\}) & \text{result state} \end{array}$$

Clearly differs from the previous slide

This is taken account of by exploiting bifunctoriality in V of

$$\mathbf{I}(U, V) \times \mathsf{Loc}(V) \times \underline{S}(V)$$

i.e.,

$$\mathbf{I}(U, V) \times \mathsf{Loc}(V) \times \underline{S}(\underline{V}) \;:\; \mathbf{I}^{\mathrm{op}} \times \mathbf{I} \to \mathbf{Set}$$

Consider

$$\text{new } l := 0 \text{ in new } l' := 1 \text{ in } l \;:\; \mathrm{Loc}$$

A possible behaviour: map the start state $s \in \underline{S}(U)$ to:

$$\begin{array}{rcll} i' & : & U \rightarrowtail U \uplus \{l, l'\} & \text{create two new locations } l, l' \\ l & \in & \mathrm{Loc}(U \uplus \{l, l'\}) & \text{result of computation} \\ s[l := 0, l' := 1] & \in & \underline{S}(U \uplus \{l, l'\}) & \text{result state} \end{array}$$

Clearly differs from the previous slide

This is taken account of by exploiting bifunctoriality in V of

$$\mathbf{I}(U, V) \times \mathrm{Loc}(V) \times \underline{S}(V)$$

i.e.,

$$\mathbf{I}(U, V) \times \mathrm{Loc}(V) \times \underline{S}(\underline{V}) \;:\; \mathbf{I}^{\mathrm{op}} \times \mathbf{I} \to \mathbf{Set}$$

Consider

$$\text{new } l := 0 \text{ in new } l' := 1 \text{ in } l \;:\; \mathrm{Loc}$$

A possible behaviour: map the start state $s \in \underline{S}(U)$ to:

$$\begin{array}{rcll} i' & : & U \rightarrowtail U \uplus \{l, l'\} & \text{create two new locations } l, l' \\ l & \in & \mathrm{Loc}(U \uplus \{l, l'\}) & \text{result of computation} \\ s[l := 0, l' := 1] & \in & \underline{S}(U \uplus \{l, l'\}) & \text{result state} \end{array}$$

Clearly differs from the previous slide

This is taken account of by exploiting bifunctoriality in V of

$$\mathbf{I}(U, V) \times \mathrm{Loc}(V) \times \underline{S}(V)$$

i.e.,

$$\mathbf{I}(U, V) \times \mathrm{Loc}(V) \times \underline{S}(\underline{V}) \;:\; \mathbf{I}^{\mathrm{op}} \times \mathbf{I} \to \mathbf{Set}$$

Consider the inclusion $j\colon V \rightarrowtail V'$ where

$$V = U \uplus \{l\} \qquad V' = U \uplus \{l, l'\}$$

Then

$$\mathbf{I}(U,V) \times \mathrm{Loc}(V) \times \underline{S}(j)\colon \mathbf{I}(U,V) \times \mathrm{Loc}(V) \times \underline{S}(V') \to \mathbf{I}(U,V) \times \mathrm{Loc}(V) \times \underline{S}(V)$$
$$\mathbf{I}(U,j) \times \mathrm{Loc}(j) \times \underline{S}(V')\colon \mathbf{I}(U,V) \times \mathrm{Loc}(V) \times \underline{S}(V') \to \mathbf{I}(U,V') \times \mathrm{Loc}(V') \times \underline{S}(V')$$

map $(i, l, s[l := 0, l' := 1])$ to respectively:

$$(i, l, s[l := 0]) \qquad (i', l, s[l := 0, l' := 1])$$

This says that both possible behaviours are manifestations of the same behaviour.

Consider the inclusion $j\colon V \rightarrowtail V'$ where

$$V = U \uplus \{l\} \qquad V' = U \uplus \{l, l'\}$$

Then

$$\mathbf{I}(U,V) \times \mathrm{Loc}(V) \times \underline{S}(j)\colon \mathbf{I}(U,V) \times \mathrm{Loc}(V) \times \underline{S}(V') \to \mathbf{I}(U,V) \times \mathrm{Loc}(V) \times \underline{S}(V)$$
$$\mathbf{I}(U,j) \times \mathrm{Loc}(j) \times \underline{S}(V')\colon \mathbf{I}(U,V) \times \mathrm{Loc}(V) \times \underline{S}(V') \to \mathbf{I}(U,V') \times \mathrm{Loc}(V') \times \underline{S}(V')$$

map $(i, l, s[l := 0, l' := 1])$ to respectively:

$$(i, l, s[l := 0]) \qquad (i', l, s[l := 0, l' := 1])$$

This says that both possible behaviours are manifestations of the same behaviour.

Given a bifunctor $F : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$ and an object D of $\mathcal{D}$, an extranatural transformation from F to the object (not bifunctor) D is a family of maps in $\mathcal{D}$

$$\alpha_C : F(C, C) \longrightarrow D$$

s.t., for every $f : C \longrightarrow C'$ in $\mathcal{C}$, the diagram below commutes.

$$\begin{array}{ccc} F(C', C) & \xrightarrow{F(j, C)} & F(C, C) \\ {\scriptstyle F(C', j)}\downarrow & & \downarrow{\scriptstyle \alpha_c} \\ F(C', C') & \xrightarrow[\alpha_{C'}]{} & D \end{array}$$

General extranatural transformations are more general. They, in turn, are special cases of dinatural transformations.

Given a bifunctor $F : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$ and an object D of $\mathcal{D}$, an extranatural transformation from F to the object (not bifunctor) D is a family of maps in $\mathcal{D}$

$$\alpha_C : F(C, C) \longrightarrow D$$

s.t., for every $f : C \longrightarrow C'$ in $\mathcal{C}$, the diagram below commutes.

$$\begin{array}{ccc} F(C', C) & \xrightarrow{F(j, C)} & F(C, C) \\ {\scriptstyle F(C', j)}\downarrow & & \downarrow{\scriptstyle \alpha_c} \\ F(C', C') & \xrightarrow[\alpha_{C'}]{} & D \end{array}$$

General extranatural transformations are more general. They, in turn, are special cases of dinatural transformations.

Coends

A coend for a bifunctor $F\colon \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$ is a universal extranatural transformation

$$\alpha\colon F \Longrightarrow \int^{C} F(C, C)$$

to an object $\int^{C} F(C, C)$ of $\mathcal{D}$.

I.e., for any extranatural transformation $\beta\colon F \Longrightarrow D$ there exists a unique map

$$\int^{C} F(C, C) \xrightarrow{g} D$$

in $\mathcal{D}$ such that $\beta = g \circ \alpha$.

The local store monad

Given an object F of $[\mathbf{I}, \mathbf{Set}]$ we define

$$(TF)(U) \;=\; \underline{S}(U) \to \int^{V} \mathbf{I}(U,V) \times F(V) \times \underline{S}(V)$$

Concretely the coend is a quotient of the set

$$\coprod_{V} \mathbf{I}(U,V) \times F(V) \times \underline{S}(V)$$

by the equivalence relation induced by the extranaturality condition.

Functorial action of TF

Given $\mathsf{i}\colon U \rightarrowtail U'$ and $P \in (TF)(U)$, we need to define

$$(TF)(i)(P) \;:\; \underline{S}(U') \to \int^{V'} \mathbf{I}(U', V') \times F(V') \times \underline{S}(V')$$

Given $s \in \underline{S}(U')$, let

$$(j, x, s') \;\in\; \mathbf{I}(U, V) \times F(V) \times \underline{S}(V)$$

be a representative of the equivalence class of $P(s_U)$, where $(s_U, s_{U'\backslash U})$ is the pair determined from s by $\underline{S}(U') \cong \underline{S}(U) \times \underline{S}(U'\backslash U)$, arising from $U' \cong U + U'\backslash U$.

Then define $(TF)(i)(P)(s)$ to be the equivalence class of

$$(\mathrm{in}_{U'},\, F(\mathrm{in}_V)(x),\, (s', s_{U'\backslash U})) \;\in\; \mathbf{I}(U', V') \times F(V') \times \underline{S}(V')$$

where $V' = (V\backslash U) + U + (U'\backslash U)$

It would of course be possible to describe the strong monad structure of T directly, and to verify its properties by direct calculation.

Instead, we shall derive it as an instance of a linear state monad $\underline{S} \multimap (-) \bullet \underline{S}$.

For this, we need an enriched action.

Action of $[\mathbf{I}, \mathbf{Set}]$ on $[\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$

Given $F\colon \mathbf{I} \to \mathbf{Set}$ and $\underline{G}\colon \mathbf{I}^{\mathrm{op}} \to \mathbf{Set}$ define $F \bullet \underline{G}\colon \mathbf{I}^{\mathrm{op}} \to \mathbf{Set}$ by:

$$(F \bullet \underline{G})(V) \;=\; \int^{V'} \mathbf{I}(V, V') \times F(V') \times \underline{G}(V')$$

On morphisms, $i\colon U \rightarrowtail V$, given a representative

$$(j, x, y) \;\in\; \mathbf{I}(V, V') \times F(V') \times \underline{G}(V')$$

of an equivalence class in $(F \bullet \underline{G})(V)$, define $(F \bullet \underline{G})(i)$ to be the equivalence class of

$$(j \circ i, x, y) \;\in\; \mathbf{I}(U, V') \times F(V') \times \underline{G}(V')$$

$[\mathbf{I}, \mathbf{Set}]$-enrichment of $[\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$

Given $\underline{F} \colon \mathbf{I}^{\mathrm{op}} \to \mathbf{Set}$ and $\underline{G} \colon \mathbf{I}^{\mathrm{op}} \to \mathbf{Set}$ define $\underline{F} \multimap \underline{G} \colon \mathbf{I} \to \mathbf{Set}$ by:

$$(\underline{F} \multimap \underline{G})(U) \;=\; \int_{V'} \mathbf{I}(U, V') \times \underline{F}(V') \to \underline{G}(V')$$

This formula uses an end, the dual notion to coend. In this case, the end defines the hom-set $[(U/\mathbf{I})^{\mathrm{op}}, \mathbf{Set}](U/\underline{F},\, U/\underline{G})$ in presheaves over the co-slice $U/\mathbf{I}$.

On morphisms, $i \colon U \rightarrowtail V$, given $\pi \in (\underline{F} \multimap \underline{G})(U)$, define $(\underline{F} \multimap \underline{G})(i)(\pi)_{V'}$ to be:

$$(j, x) \mapsto \pi(j \circ i,\, x) \;\; : \;\; \mathbf{I}(V, V') \times \underline{F}(V') \to \underline{G}(V')$$

which is indeed natural.

Powers in $[\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$

Given $F\colon \mathbf{I} \to \mathbf{Set}$ and $\underline{G}\colon \mathbf{I}^{\mathrm{op}} \to \mathbf{Set}$ define $\underline{G}^F : \mathbf{I}^{\mathrm{op}} \to \mathbf{Set}$ by:

$$(\underline{G}^F)(V) \;=\; F(V) \to G(V)$$

On morphisms, $i\colon U \rightarrowtail V$, given $f \in (\underline{G}^F)(V)$, define

$$(\underline{G}^F)(i)(f) \;=\; \underline{G}(i) \circ f \circ F(i) \;:\; F(U) \to \underline{G}(U)$$

We have given an enriched action with powers of $[\mathbf{I}, \mathbf{Set}]$ on $[\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$.

We thus obtain a model of the enriched effect calculus with value category $[\mathbf{I}, \mathbf{Set}]$ and computation category $[\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$.

The computation category has a natural state object $\underline{S}$.

Sadly, $\underline{S} \multimap (-) \bullet \underline{S}$ is not the local store monad.

$$(\underline{S} \multimap F \bullet \underline{S})(U) \;=\; \int_V \left(\mathbf{I}(U, V) \times \underline{S}(V) \to \int^{V'} \mathbf{I}(V, V') \times F(V') \times \underline{S}(V') \right)$$

We have given an enriched action with powers of $[\mathbf{I}, \mathbf{Set}]$ on $[\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$.

We thus obtain a model of the enriched effect calculus with value category $[\mathbf{I}, \mathbf{Set}]$ and computation category $[\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$.

The computation category has a natural state object $\underline{S}$.

Sadly, $\underline{S} \multimap (-) \bullet \underline{S}$ is not the local store monad.

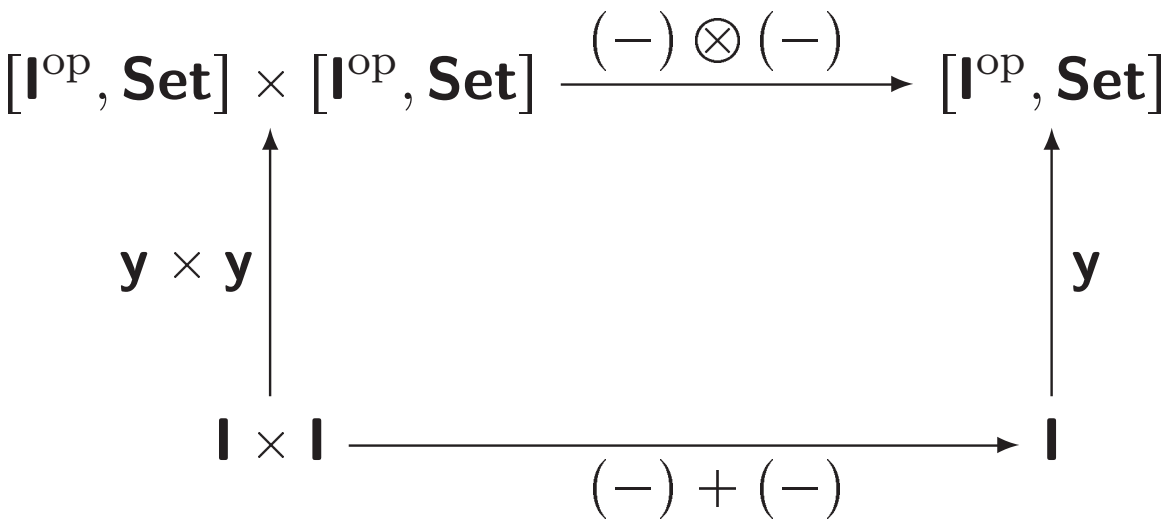
$$(\underline{S} \multimap F \bullet \underline{S})(U) \;=\; \int_V \left(\mathbf{I}(U, V) \times \underline{S}(V) \to \int^{V'} \mathbf{I}(V, V') \times F(V') \times \underline{S}(V') \right)$$

Day convolution

Coproduct in **Set** gives a symmetric monoidal product

$$(U, V) \mapsto U + V : \mathbf{I} \times \mathbf{I} \to \mathbf{I}$$

Day convolution lifts this to a symmetric monoidal product on presheaves $[\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$.



It is the unique (up to natural isomorphism) functor that is cocontinuous in each argument separately and makes the diagram commute (up to natural isomorphism).

The general definition of Day convolution gives the formula

$$\underline{F} \otimes \underline{G} \;=\; \int^{V_1, V_2} \mathbf{I}(-, V_1 + V_2) \times \underline{F}(V_1) \times \underline{G}(V_2)$$

In the case at hand, this simplifies to:

$$(\underline{F} \otimes \underline{G})(V) \;=\; \coprod_{U \in \mathcal{P}(V)} \underline{F}(U) \times \underline{G}(V \backslash U)$$

(Note that $\underline{G}(V \backslash U)$ is covariant in U.)

The unit of the monoidal structure is $\underline{I}$:

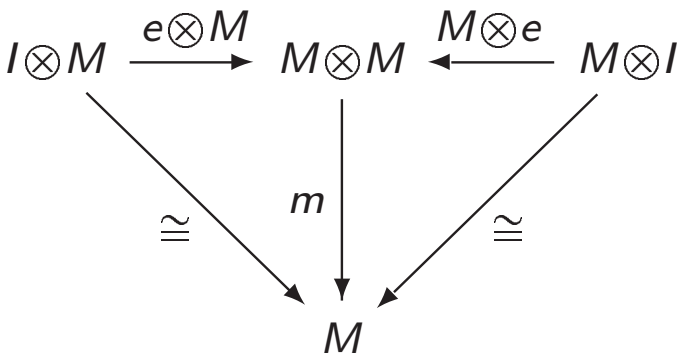
$$\underline{I}(U) \;=\; \begin{cases} 1 & \text{if } U = 0 \\ 0 & \text{otherwise} \end{cases}$$

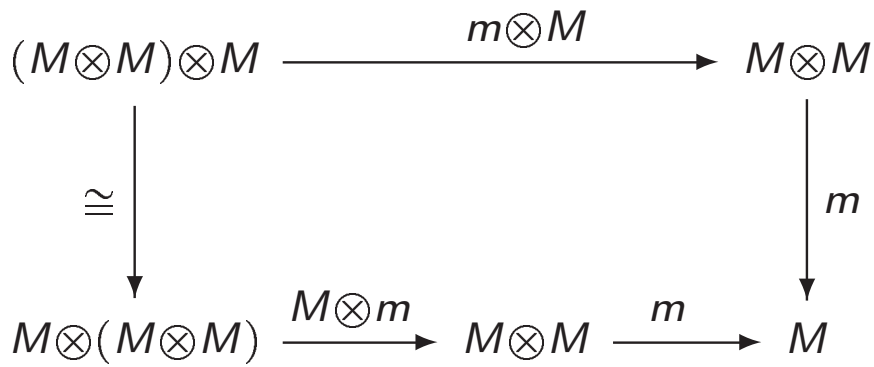
Monoids in a monoidal category

A monoid in a monoidal category $(\mathcal{C}, \otimes, I)$ is given by a structure

$$I \xrightarrow{e} M \xleftarrow{m} M \otimes M$$

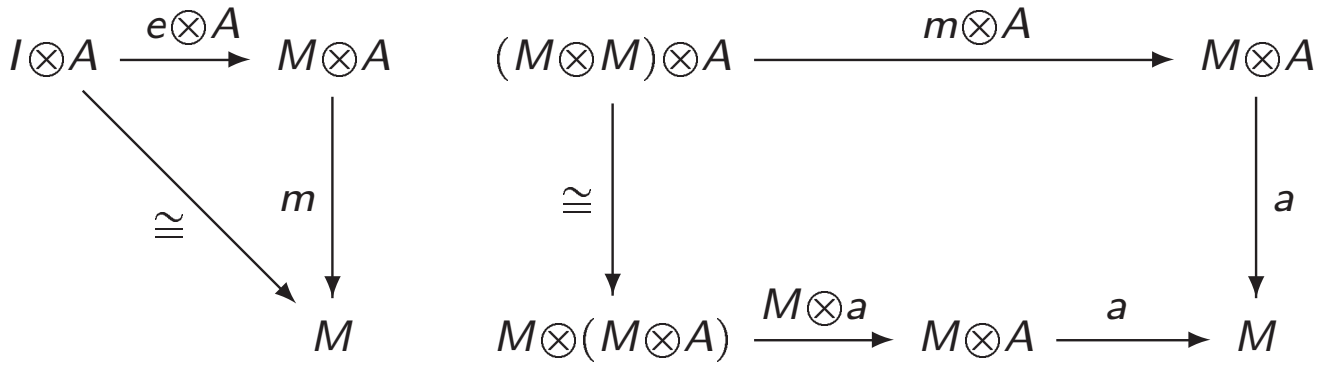
such that the diagrams below commute.



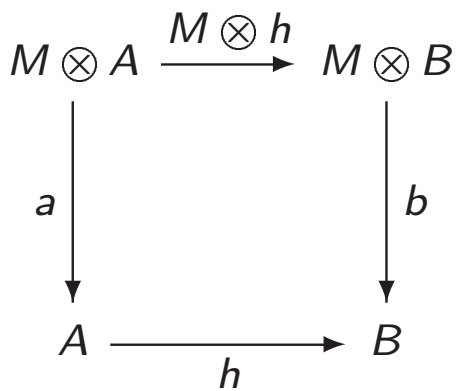


Monoid actions in a monoidal category

A monoid action is a structure (A, a) where $a\colon M \otimes A \longrightarrow A$ is such that the diagrams below commute.



A homomorphism of monoid actions from (A, a) to (B, b) is a map $h\colon A \longrightarrow B$ such that the diagram below commutes.



$\underline{S}$ is a monoid in $[\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$

We define a monoid structure on $\underline{S}$ w.r.t. Day convolution:

$$\underline{I} \xrightarrow{e} \underline{S} \xleftarrow{m} \underline{S} \otimes \underline{S}$$

The unit e is the unique map.

A component m_V of the multiplication maps an element

$$(U, s_U, s_{V \setminus U}) \in \mathcal{P}(V) \times \underline{S}(U) \times \underline{S}(V \setminus U)$$

of $(\underline{S} \otimes \underline{S})(V)$ to $(s_U, s_{V \setminus U})$.

Our computation category will be the category $\mathbf{Act}_{\underline{S}}$ of $\underline{S}$-actions.

$\underline{S}$ is a monoid in $[\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$

We define a monoid structure on $\underline{S}$ w.r.t. Day convolution:

$$\underline{I} \xrightarrow{e} \underline{S} \xleftarrow{m} \underline{S} \otimes \underline{S}$$

The unit e is the unique map.

A component m_V of the multiplication maps an element

$$(U, s_U, s_{V \setminus U}) \in \mathcal{P}(V) \times \underline{S}(U) \times \underline{S}(V \setminus U)$$

of $(\underline{S} \otimes \underline{S})(V)$ to $(s_U, s_{V \setminus U})$.

Our computation category will be the category $\mathbf{Act}_{\underline{S}}$ of $\underline{S}$-actions.

Action of $[\mathbf{I}, \mathbf{Set}]$ on $\mathbf{Act}_{\underline{S}}$

The action is created by the forgetful functor $\mathbf{Act}_{\underline{S}} \to [\mathbf{I}^{\mathrm{op}}, \mathbf{Set}]$.

Given an $\underline{S}$-action $(\underline{G}, g)$, the action $a\colon \underline{S} \otimes (F \bullet \underline{G}) \longrightarrow F \bullet \underline{G}$ on

$$F \bullet \underline{G} \;=\; \int^{V} \mathbf{I}(-, V) \times F(V) \times \underline{G}(V)$$

is defined as follows. Consider a representative

$$(U \backslash U', s', (i, x, y)) \;\in\; \mathcal{P}(U) \times \underline{S}(U \backslash U') \times (\mathbf{I}(U', V) \times F(V) \times \underline{G}(V))$$

of an element of $(\underline{S} \otimes (F \bullet \underline{G}))(U)$. Then a_U maps this to the equivalence class represented by

$$(\mathrm{in}_U, F(\mathrm{in}_V)(x), g(s', y)) \;\in\; \mathbf{I}(U, V') \times F(U \backslash V') \times \underline{G}(V'))$$

where $V' = V + U \backslash U'$, which is an element of $F \bullet \underline{G}$.

Definition of $\underline{S} \multimap_{\mathbf{Act}_{\underline{S}}} (-)$

The action of $[\mathbf{I}, \mathbf{Set}]$ on $\mathbf{Act}_{\underline{S}}$ is enriched and has powers. But the general definition of this structure is intricate.

We just define the case of enrichment we we need, $\underline{S} \multimap_{\mathbf{Act}_{\underline{S}}} (\underline{G}, g)$ for an $\underline{S}$-action $(\underline{G}, g)$, which is comparatively simple:

$$(\underline{S} \multimap_{\mathbf{Act}_{\underline{S}}} (\underline{G}, g))(U) \;=\; \underline{S}(U) \to \underline{G}(U)$$

On morphisms, $i \colon U \rightarrowtail V$, given $f : \underline{S}(U) \to \underline{G}(U)$, define $(\underline{S} \multimap_{\mathbf{Act}_{\underline{S}}} (\underline{G}, g))(i)(f)$ to be:

$$s \mapsto g_V(V \backslash U,\, s_{V \backslash U},\, f(s_U)) \;:\; \underline{S}(V) \to \underline{G}(V)$$

The local store monad

We have derived

$$(\underline{S} \multimap_{\mathbf{Act}_{\underline{S}}} F \bullet \underline{S})(U) \;=\; \underline{S}(U) \to \int^{V} \mathbf{I}(U,V) \times F(V) \times \underline{S}(V)$$

Hence $\underline{S} \multimap_{\mathbf{Act}_{\underline{S}}} F \bullet \underline{S} = TF$, and so the definition of T indeed defines a strong monad.

[[These slides will be completed and updated with references]]