What Monads Can and Cannot Do with a bit of Extra Time

Rasmus Møgelberg & Maaike Zwart,
Philipp Jan Andries Stassen, Alejandro Aguirre, Lars Birkedal

IT University of Copenhagen & Aarhus University

04 April 2024

Overview

Does the Powerset monad distribute over the Delay monad?
(and what about the Distribution monad?)

Overview

Does the Powerset **monad** distribute over the Delay monad?
(and what about the Distribution monad?)

Overview

Does the **Powerset monad** distribute over the **Delay monad**?
(and what about the **Distribution monad**?)

Overview

Does the Powerset monad **distribute** over the Delay monad?
(and what about the Distribution monad?)

Overview

Does the Powerset monad distribute over the Delay monad?
(and what about the Distribution monad?)

For Powerset we will:

For Distributions we will:

Overview

Does the Powerset monad distribute over the Delay monad?
(and what about the Distribution monad?)

For Powerset we will:

- Try the obvious sequential and parallel computation

For Distributions we will:

Overview

Does the Powerset monad distribute over the Delay monad?
(and what about the Distribution monad?)

For Powerset we will:

- Try the obvious sequential and parallel computation
- See that it fails - Rasmus Møgelberg and Andrea Vezzosi

For Distributions we will:

Overview

Does the Powerset monad distribute over the Delay monad?
(and what about the Distribution monad?)

For Powerset we will:

- Try the obvious sequential and parallel computation
- See that it fails - Rasmus Møgelberg and Andrea Vezzosi
- Discover why it fails not **just** because of idempotence!

For Distributions we will:

Overview

Does the Powerset monad distribute over the Delay monad?
(and what about the Distribution monad?)

For Powerset we will:

- Try the obvious sequential and parallel computation
- See that it fails - Rasmus Møgelberg and Andrea Vezzosi
- Discover why it fails not **just** because of idempotence!
- Prove that it is impossible ...because of idempotence.

For Distributions we will:

- Prove that it is impossible ...because of idempotence.

Overview

Does the Powerset monad distribute over the Delay monad?
(and what about the Distribution monad?)

For Powerset we will:

- Try the obvious sequential and parallel computation
- See that it fails - Rasmus Møgelberg and Andrea Vezzosi
- Discover why it fails not **just** because of idempotence!
- Prove that it is impossible ...because of idempotence.

For Distributions we will:

- Prove that it is impossible ...because of idempotence.
- Look at a free combination

Monads

Why use monads?

What are monads?

Monads

Why use monads?

Models of Computation: non-determinism, probability, states, ...

What are monads?

Monads

Why use monads?

Models of Computation: non-determinism, probability, states, ...

What are monads?

Functors with structure: $\langle \mathcal{M},\ \eta : 1 \to \mathcal{M},\ \mu : \mathcal{M}\mathcal{M} \to \mathcal{M} \rangle$

Monads

Why use monads?

Models of Computation: non-determinism, probability, states, ...

What are monads?

Functors with structure: $\langle \mathcal{M},\ \eta : 1 \to \mathcal{M},\ \mu : \mathcal{M}\mathcal{M} \to \mathcal{M} \rangle$

Algebraic structures (HITs): $\langle \Sigma, E \rangle$.

Monads

Why use monads?

Models of Computation: non-determinism, probability, states, ...

What are monads?

Functors with structure: $\langle \mathcal{M},\ \eta : 1 \to \mathcal{M},\ \mu : \mathcal{M}\mathcal{M} \to \mathcal{M} \rangle$

Algebraic structures (HITs): $\langle \Sigma, E \rangle$.

Powerset monad for non-determinism:

$$\langle \mathcal{P}, \eta_{\mathcal{P}}, \mu_{\mathcal{P}} \rangle$$

$$\{1^{(0)}, *^{(2)}\}$$

$$\mathcal{P}(X) = \{Y | Y \subseteq X \text{ finite}\}$$
$$\eta_{\mathcal{P}}(x) = \{x\}$$
$$\mu_{\mathcal{P}}(Y) = \bigcup Y$$

$$1 * x = x$$
$$x * (y * z) = (x * y) * z$$
$$x * y = y * x$$
$$x * x = x$$

Monads

Why use monads?

Models of Computation: non-determinism, probability, states, ...

What are monads?

Functors with structure: $\langle \mathcal{M},\ \eta : 1 \to \mathcal{M},\ \mu : \mathcal{M}\mathcal{M} \to \mathcal{M} \rangle$

Algebraic structures (HITs): $\langle \Sigma, E \rangle$.

Distribution monad for probability:

$$\langle \mathcal{D}, \eta_{\mathcal{D}}, \mu_{\mathcal{D}} \rangle$$

$$\{+_p^{(2)} | p : [0,1]\}$$

$$\mathcal{D}(X) = \{\mu | \mu \text{ has finite support}\}$$

$$\eta_{\mathcal{D}}(x) = \delta_x$$

$$\mu_{\mathcal{D}}(Y) = \text{sum via weighted average}$$

$$x +_1 y = x$$

$$x +_p x = x$$

$$x +_p y = y +_{(1-p)} x$$

$$(x +_p y) +_q z = x +_{pq} (y +_{\frac{q-pq}{1-pq}} z)$$

Delay Monad

For recursion / computation steps

Coinductive version
(iteration)

$$\langle \mathcal{L}, \eta_{\mathcal{L}}, \mu_{\mathcal{L}} \rangle$$

$$\mathcal{L}(X) \simeq X + \mathcal{L}(X)$$

$$\eta_{\mathcal{L}}(x) = \mathsf{now}\, x = \mathsf{inl}\, x$$

$$\mathsf{step}\, x = \mathsf{inr}\, x$$

$$\mu_{\mathcal{L}}(d) = \text{‘adding steps’}$$

Delay Monad

For recursion / computation steps

Coinductive version
(iteration)

$$\langle \mathcal{L}, \eta_{\mathcal{L}}, \mu_{\mathcal{L}} \rangle$$

$$\begin{aligned} \mathcal{L}(X) &\simeq X + \mathcal{L}(X) \\ \eta_{\mathcal{L}}(x) &= \mathsf{now}\, x = \mathsf{inl}\, x \\ & \quad\ \mathsf{step}\, x = \mathsf{inr}\, x \\ \mu_{\mathcal{L}}(d) &= \text{`adding steps'} \end{aligned}$$

$$\mu_{\mathcal{L}}(\mathsf{step}\,\mathsf{now}(\mathsf{step}\,\mathsf{step}\,\mathsf{now}\, x)) = \mathsf{step}\,\mathsf{step}\,\mathsf{step}\,\mathsf{now}\, x$$

Delay Monad

For recursion / computation steps

Coinductive version
(iteration)

$$\langle \mathcal{L}, \eta_{\mathcal{L}}, \mu_{\mathcal{L}} \rangle$$

$$\begin{aligned} \mathcal{L}(X) &\simeq X + \mathcal{L}(X) \\ \eta_{\mathcal{L}}(x) &= \mathsf{now}\, x = \mathsf{inl}\, x \\ &\ \mathsf{step}\, x = \mathsf{inr}\, x \\ \mu_{\mathcal{L}}(d) &= \text{‘adding steps’} \end{aligned}$$

Delay Monad

For recursion / computation steps

Coinductive version
(iteration)

$$\langle \mathcal{L}, \eta_{\mathcal{L}}, \mu_{\mathcal{L}} \rangle$$

$$\mathcal{L}(X) \simeq X + \mathcal{L}(X)$$
$$\eta_{\mathcal{L}}(x) = \mathsf{now}\, x = \mathsf{inl}\, x$$
$$\mathsf{step}\, x = \mathsf{inr}\, x$$
$$\mu_{\mathcal{L}}(d) = \text{‘adding steps’}$$

Guarded recursive version
(recursion)

$$\langle \mathcal{L}^{\kappa}, \eta_{\mathcal{L}^{\kappa}}, \mu_{\mathcal{L}^{\kappa}} \rangle$$

$$\mathcal{L}^{\kappa}(X) \simeq X + \rhd \mathcal{L}^{\kappa} X$$
$$\eta_{\mathcal{L}^{\kappa}}(x) = \mathsf{now}^{\kappa}\, x = \mathsf{inl}\, x$$
$$\mathsf{step}^{\kappa}\, x = \mathsf{inr}\, x$$
$$\mu_{\mathcal{L}^{\kappa}}(d) = \text{‘adding steps’}$$

Delay Monad

For recursion / computation steps

Coinductive version (iteration)

$$\langle \mathcal{L}, \eta_{\mathcal{L}}, \mu_{\mathcal{L}} \rangle$$

$$\mathcal{L}(X) \simeq X + \mathcal{L}(X)$$

$$\eta_{\mathcal{L}}(x) = \mathsf{now}\, x = \mathsf{inl}\, x$$

$$\mathsf{step}\, x = \mathsf{inr}\, x$$

$$\mu_{\mathcal{L}}(d) = \text{‘adding steps’}$$

Guarded recursive version (recursion)

$$\langle \mathcal{L}^{\kappa}, \eta_{\mathcal{L}^{\kappa}}, \mu_{\mathcal{L}^{\kappa}} \rangle$$

$$\mathcal{L}^{\kappa}(X) \simeq X + \rhd \mathcal{L}^{\kappa} X$$

$$\eta_{\mathcal{L}^{\kappa}}(x) = \mathsf{now}^{\kappa}\, x = \mathsf{inl}\, x$$

$$\mathsf{step}^{\kappa}\, x = \mathsf{inr}\, x$$

$$\mu_{\mathcal{L}^{\kappa}}(d) = \text{‘adding steps’}$$

$$\mu_{\mathcal{L}^{\kappa}}(\mathsf{step}^{\kappa}\, d) = \mathsf{step}^{\kappa}(\lambda(\alpha : \kappa).\mu_{\mathcal{L}^{\kappa}}(d[\alpha]))$$

Delay Monad

For recursion / computation steps

Coinductive version
(iteration)

$$\langle \mathcal{L}, \eta_{\mathcal{L}}, \mu_{\mathcal{L}} \rangle$$

$$\mathcal{L}(X) \simeq X + \mathcal{L}(X)$$

$$\eta_{\mathcal{L}}(x) = \mathsf{now}\, x = \mathsf{inl}\, x$$

$$\mathsf{step}\, x = \mathsf{inr}\, x$$

$$\mu_{\mathcal{L}}(d) = \text{‘adding steps’}$$

Guarded recursive version
(recursion)

$$\langle \mathcal{L}^{\kappa}, \eta_{\mathcal{L}^{\kappa}}, \mu_{\mathcal{L}^{\kappa}} \rangle$$

$$\mathcal{L}^{\kappa}(X) \simeq X + \rhd \mathcal{L}^{\kappa} X$$

$$\eta_{\mathcal{L}^{\kappa}}(x) = \mathsf{now}^{\kappa}\, x = \mathsf{inl}\, x$$

$$\mathsf{step}^{\kappa}\, x = \mathsf{inr}\, x$$

$$\mu_{\mathcal{L}^{\kappa}}(d) = \text{‘adding steps’}$$

Delay Monad

For recursion / computation steps

Coinductive version
(iteration)

$$\langle \mathcal{L}, \eta_{\mathcal{L}}, \mu_{\mathcal{L}} \rangle$$

$$\mathcal{L}(X) \simeq X + \mathcal{L}(X)$$

$$\eta_{\mathcal{L}}(x) = \mathsf{now}\, x = \mathsf{inl}\, x$$

$$\mathsf{step}\, x = \mathsf{inr}\, x$$

$$\mu_{\mathcal{L}}(d) = \text{`adding steps'}$$

Guarded recursive version
(recursion)

$$\langle \mathcal{L}^{\kappa}, \eta_{\mathcal{L}^{\kappa}}, \mu_{\mathcal{L}^{\kappa}} \rangle$$

$$\mathcal{L}^{\kappa}(X) \simeq X + \triangleright \mathcal{L}^{\kappa} X$$

$$\eta_{\mathcal{L}^{\kappa}}(x) = \mathsf{now}^{\kappa}\, x = \mathsf{inl}\, x$$

$$\mathsf{step}^{\kappa}\, x = \mathsf{inr}\, x$$

$$\mu_{\mathcal{L}^{\kappa}}(d) = \text{`adding steps'}$$

$$\mathcal{L}X = \forall \kappa . \mathcal{L}^{\kappa} X$$

Combining Monads

Free combination

No interaction between monads.

Combining Monads

Free combination

No interaction between monads.

Composition $\mathcal{LP}$

Interaction between monads via distributive law $\lambda : \mathcal{PL} \to \mathcal{LP}$.

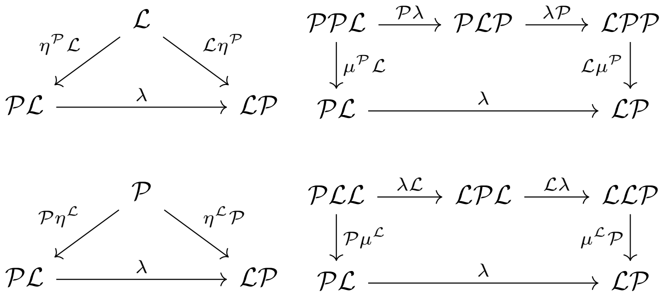
Combining Monads

Free combination

No interaction between monads.

Composition $\mathcal{LP}$

Interaction between monads via distributive law $\lambda : \mathcal{PL} \to \mathcal{LP}$.



Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

Delayed set of computations

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

$$\{?,?,?,?,?\}$$

Delayed set of computations

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

{?,  ,?,?,?}

Delayed set of computations

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

$$\{?,5,?,?,?\}$$

Delayed set of computations

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

$$\{?,5,?,?,?\}$$

Delayed set of computations, sequential:

 $\{?,?,?,?,?\}$

Delayed set of computations, parallel:

$\{?,?,?,?,?\}$

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

$$\{?,5,?,?,?\}$$

Delayed set of computations, sequential:

 {  ,?,?,?,?}

Delayed set of computations, parallel:

$$\{?,?,?,?,?\}$$

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

$$\{?,5,?,?,?\}$$

Delayed set of computations, sequential:

⌛ $\{2,$ ⌛ $,?,?,?\}$

Delayed set of computations, parallel:

$$\{?,?,?,?,?\}$$

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

$$\{?,5,?,?,?\}$$

Delayed set of computations, sequential:

 {2,5,  ,?,?}

Delayed set of computations, parallel:

$\{?,?,?,?,?\}$

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

$$\{?,5,?,?,?\}$$

Delayed set of computations, sequential:

⌛ $\{2,5,3,$ ⌛ $,?\}$

Delayed set of computations, parallel:

$\{?,?,?,?,?\}$

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \rightarrow \mathcal{LP}$

Set of delayed computations:

$$\{?,5,?,?,?\}$$

Delayed set of computations, sequential:

 $\{2,5,3,8,$  $\}$

Delayed set of computations, parallel:

$\{?,?,?,?,?\}$

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

$$\{?,5,?,?,?\}$$

Delayed set of computations, sequential:

$\{2,5,3,8,6\}$

Delayed set of computations, parallel:

$\{?,?,?,?,?\}$

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

{?,5,?,?,?}

Delayed set of computations, sequential:

{2,5,3,8,6}   Total time: sum of computation times

Delayed set of computations, parallel:

{?,?,?,?,?}

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

$$\{?,5,?,?,?\}$$

Delayed set of computations, sequential:

$\{2,5,3,8,6\}$ Total time: sum of computation times

Delayed set of computations, parallel:

$\{?,?,?,?,?\}$

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

$$\{?,5,?,?,?\}$$

Delayed set of computations, sequential:

$\{2,5,3,8,6\}$ Total time: sum of computation times

Delayed set of computations, parallel:

⌛ {⌛,⌛,⌛,⌛,⌛}

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

$$\{?,5,?,?,?\}$$

Delayed set of computations, sequential:

$\{2,5,3,8,6\}$ Total time: sum of computation times

Delayed set of computations, parallel:

⌛ $\{2,5,$ ⌛ $,8,$ ⌛ $\}$

Sequential and Parallel Computation

As candidates for $\lambda : \mathcal{PL} \to \mathcal{LP}$

Set of delayed computations:

{?,5,?,?,?}

Delayed set of computations, sequential:

{2,5,3,8,6} Total time: sum of computation times

Delayed set of computations, parallel:

{2,5,3,8,6} Total time: max of computation times

Sequential Computation

More precise:

$$\begin{aligned} \lambda\{\mathsf{now}\, x, \mathsf{now}\, y\} &= \mathsf{now}\{x, y\} \\ \lambda\{\mathsf{step}\, d, d'\} &= \mathsf{step}(\lambda\{d, d'\}) \end{aligned}$$

so:

$$\{\mathsf{step}\,\mathsf{step}\,\mathsf{now}\, x, \mathsf{now}\, y, \mathsf{step}\,\mathsf{now}\, z\} \mapsto \mathsf{step}\,\mathsf{step}\,\mathsf{step}\,\mathsf{now}\{x, y, z\}$$

Sequential Computation

More precise:

$$\lambda\{\text{now}\,x, \text{now}\,y\} = \text{now}\{x, y\}$$
$$\lambda\{\text{step}\,d, d'\} = \text{step}(\lambda\{d, d'\})$$

so:

$$\{\text{step}\,\text{step}\,\text{now}\,x, \text{now}\,y, \text{step}\,\text{now}\,z\} \mapsto \text{step}\,\text{step}\,\text{step}\,\text{now}\{x, y, z\}$$

Not a distributive law!

Sequential Computation

More precise:

$$\lambda\{\mathsf{now}\, x, \mathsf{now}\, y\} = \mathsf{now}\{x, y\}$$
$$\lambda\{\mathsf{step}\, d, d'\} = \mathsf{step}(\lambda\{d, d'\})$$

so:

$$\{\mathsf{step}\,\mathsf{step}\,\mathsf{now}\, x, \mathsf{now}\, y, \mathsf{step}\,\mathsf{now}\, z\} \mapsto \mathsf{step}\,\mathsf{step}\,\mathsf{step}\,\mathsf{now}\{x, y, z\}$$

Not a distributive law!
Why?

Sequential Computation

More precise:

$$\lambda\{\mathsf{now}\,x, \mathsf{now}\,y\} = \mathsf{now}\{x, y\}$$
$$\lambda\{\mathsf{step}\,d, d'\} = \mathsf{step}(\lambda\{d, d'\})$$

so:

$$\{\mathsf{step}\,\mathsf{step}\,\mathsf{now}\,x, \mathsf{now}\,y, \mathsf{step}\,\mathsf{now}\,z\} \mapsto \mathsf{step}\,\mathsf{step}\,\mathsf{step}\,\mathsf{now}\{x, y, z\}$$

Not a distributive law!
Why? Idempotence:

Sequential Computation

More precise:

$$\lambda\{\mathsf{now}\, x, \mathsf{now}\, y\} = \mathsf{now}\{x, y\}$$
$$\lambda\{\mathsf{step}\, d, d'\} = \mathsf{step}(\lambda\{d, d'\})$$

so:

$$\{\mathsf{step}\,\mathsf{step}\,\mathsf{now}\, x, \mathsf{now}\, y, \mathsf{step}\,\mathsf{now}\, z\} \mapsto \mathsf{step}\,\mathsf{step}\,\mathsf{step}\,\mathsf{now}\{x, y, z\}$$

Not a distributive law!
Why? Idempotence:

$$\{\mathsf{step}\,\mathsf{now}\, x\}$$
$$\{\mathsf{step}\,\mathsf{now}\, x, \mathsf{step}\,\mathsf{now}\, x\}$$

Sequential Computation

More precise:

$$\lambda\{\mathsf{now}\,x, \mathsf{now}\,y\} = \mathsf{now}\{x, y\}$$
$$\lambda\{\mathsf{step}\,d, d'\} = \mathsf{step}(\lambda\{d, d'\})$$

so:

$$\{\mathsf{step}\,\mathsf{step}\,\mathsf{now}\,x, \mathsf{now}\,y, \mathsf{step}\,\mathsf{now}\,z\} \mapsto \mathsf{step}\,\mathsf{step}\,\mathsf{step}\,\mathsf{now}\{x, y, z\}$$

Not a distributive law!
Why? Idempotence:

$$\{\mathsf{step}\,\mathsf{now}\,x\} \mapsto \mathsf{step}\,\mathsf{now}\{x\}$$
$$\{\mathsf{step}\,\mathsf{now}\,x, \mathsf{step}\,\mathsf{now}\,x\}$$

Sequential Computation

More precise:

$$\lambda\{\mathsf{now}\,x, \mathsf{now}\,y\} = \mathsf{now}\{x, y\}$$
$$\lambda\{\mathsf{step}\,d, d'\} = \mathsf{step}(\lambda\{d, d'\})$$

so:

$$\{\mathsf{step}\,\mathsf{step}\,\mathsf{now}\,x, \mathsf{now}\,y, \mathsf{step}\,\mathsf{now}\,z\} \mapsto \mathsf{step}\,\mathsf{step}\,\mathsf{step}\,\mathsf{now}\{x, y, z\}$$

Not a distributive law!
Why? Idempotence:

$$\{\mathsf{step}\,\mathsf{now}\,x\} \mapsto \mathsf{step}\,\mathsf{now}\{x\}$$
$$\{\mathsf{step}\,\mathsf{now}\,x, \mathsf{step}\,\mathsf{now}\,x\} \mapsto \mathsf{step}\,\mathsf{step}\,\mathsf{now}\{x, x\} = \mathsf{step}\,\mathsf{step}\,\mathsf{now}\{x\}$$

Sequential Computation

So what went wrong?
Imbalance of variables:

$$x * x = x$$

Sequential Computation

So what went wrong?
Imbalance of variables:

$$x * x = x \qquad x * x = x * x * x$$

Sequential Computation

So what went wrong?
Imbalance of variables:

$$x * x = x \qquad x * x = x * x * x \qquad x * (x + y) = x + y$$

Sequential Computation

So what went wrong?
Imbalance of variables:

$$x * x = x \qquad x * x = x * x * x \qquad x * (x + y) = x + y$$

Theorem

Sequential computation is a distributive law for $\mathcal{M}\mathcal{L} \to \mathcal{L}\mathcal{M}$ for $\mathcal{M}$ presented by balanced equations.

Parallel Computation

More precise:

$$\begin{aligned}
\lambda\{\mathsf{now}\,x, \mathsf{now}\,y\} &= \mathsf{now}\{x, y\} \\
\lambda\{\mathsf{step}\,d, \mathsf{now}\,y\} &= \mathsf{step}(\lambda\{d, \mathsf{now}\,y\}) \\
\lambda\{\mathsf{step}\,d, \mathsf{step}\,d'\} &= \mathsf{step}(\lambda\{d, d'\})
\end{aligned}$$

so:

$$\{\mathsf{step}\,\mathsf{step}\,\mathsf{now}\,x, \mathsf{now}\,y, \mathsf{step}\,\mathsf{now}\,z\} \mapsto \mathsf{step}\,\mathsf{step}\,\mathsf{now}\{x, y, z\}$$

Parallel Computation

More precise:

$$\begin{aligned}
\lambda\{\mathsf{now}\,x, \mathsf{now}\,y\} &= \mathsf{now}\{x, y\} \\
\lambda\{\mathsf{step}\,d, \mathsf{now}\,y\} &= \mathsf{step}(\lambda\{d, \mathsf{now}\,y\}) \\
\lambda\{\mathsf{step}\,d, \mathsf{step}\,d'\} &= \mathsf{step}(\lambda\{d, d'\})
\end{aligned}$$

so:

$$\{\mathsf{step}\,\mathsf{step}\,\mathsf{now}\,x, \mathsf{now}\,y, \mathsf{step}\,\mathsf{now}\,z\} \mapsto \mathsf{step}\,\mathsf{step}\,\mathsf{now}\{x, y, z\}$$

Not a distributive law!

Parallel Computation

More precise:

$$\lambda\{\text{now}\,x, \text{now}\,y\} = \text{now}\{x, y\}$$
$$\lambda\{\text{step}\,d, \text{now}\,y\} = \text{step}(\lambda\{d, \text{now}\,y\})$$
$$\lambda\{\text{step}\,d, \text{step}\,d'\} = \text{step}(\lambda\{d, d'\})$$

so:

$$\{\text{step step now}\,x, \text{now}\,y, \text{step now}\,z\} \mapsto \text{step step now}\{x, y, z\}$$

Not a distributive law!

Why?

Parallel Computation

More precise:

$$\begin{aligned}
\lambda\{\text{now}\, x, \text{now}\, y\} &= \text{now}\{x, y\} \\
\lambda\{\text{step}\, d, \text{now}\, y\} &= \text{step}(\lambda\{d, \text{now}\, y\}) \\
\lambda\{\text{step}\, d, \text{step}\, d'\} &= \text{step}(\lambda\{d, d'\})
\end{aligned}$$

so:

$$\{\text{step step now}\, x, \text{now}\, y, \text{step now}\, z\} \mapsto \text{step step now}\{x, y, z\}$$

Not a distributive law!

Why? NOT idempotence!

Parallel Computation

$$
\begin{array}{ccccc}
\mathcal{PLL} & \xrightarrow{\lambda\mathcal{L}} & \mathcal{LPL} & \xrightarrow{\mathcal{L}\lambda} & \mathcal{LLP} \\
\downarrow{\scriptstyle \mathcal{P}\mu^{\mathcal{L}}} & & & & \downarrow{\scriptstyle \mu^{\mathcal{L}}\mathcal{P}} \\
\mathcal{PL} & & \xrightarrow{\qquad\quad\lambda\qquad\quad} & & \mathcal{LP}
\end{array}
$$

$\{\mathsf{step}\,\mathsf{now}(\mathsf{now}\,x), \mathsf{now}(\mathsf{step}\,\mathsf{now}\,y)\}$

Parallel Computation

$$\begin{array}{ccccc}
\mathcal{P}\mathcal{L}\mathcal{L} & \xrightarrow{\lambda\mathcal{L}} & \mathcal{L}\mathcal{P}\mathcal{L} & \xrightarrow{\mathcal{L}\lambda} & \mathcal{L}\mathcal{L}\mathcal{P} \\
\downarrow{\scriptstyle \mathcal{P}\mu^{\mathcal{L}}} & & & & \downarrow{\scriptstyle \mu^{\mathcal{L}}\mathcal{P}} \\
\mathcal{P}\mathcal{L} & & \xrightarrow{\qquad\lambda\qquad} & & \mathcal{L}\mathcal{P}
\end{array}$$

$$\{\text{step now}(\text{now } x), \text{now}(\text{step now } y)\} \downarrow_{\mathcal{P}\mu^{\mathcal{L}}} \{\text{step now } x, \text{step now } y\}$$

Parallel Computation

$$
\begin{array}{ccccc}
\mathcal{PLL} & \xrightarrow{\lambda\mathcal{L}} & \mathcal{LPL} & \xrightarrow{\mathcal{L}\lambda} & \mathcal{LLP} \\
\downarrow{\scriptstyle \mathcal{P}\mu^{\mathcal{L}}} & & & & \downarrow{\scriptstyle \mu^{\mathcal{L}}\mathcal{P}} \\
\mathcal{PL} & & \xrightarrow{\qquad\lambda\qquad} & & \mathcal{LP}
\end{array}
$$

$$
\{\text{step now}(\text{now } x), \text{now}(\text{step now } y)\} \downarrow_{\mathcal{P}\mu^{\mathcal{L}}} \{\text{step now } x, \text{step now } y\}
$$

$$
\xrightarrow{\lambda} \text{step now}\{x, y\}
$$

Parallel Computation

$$\begin{array}{ccccc}
\mathcal{PLL} & \xrightarrow{\lambda\mathcal{L}} & \mathcal{LPL} & \xrightarrow{\mathcal{L}\lambda} & \mathcal{LLP} \\
\downarrow{\scriptstyle \mathcal{P}\mu^{\mathcal{L}}} & & & & \downarrow{\scriptstyle \mu^{\mathcal{L}}\mathcal{P}} \\
\mathcal{PL} & & \xrightarrow{\qquad\lambda\qquad} & & \mathcal{LP}
\end{array}$$

$$\{\text{step now}(\text{now } x), \text{now}(\text{step now } y)\} \downarrow_{\mathcal{P}\mu^{\mathcal{L}}} \{\text{step now } x, \text{step now } y\}$$

$$\xrightarrow{\lambda} \text{step now}\{x, y\}$$

$$\{\text{step now}(\text{now } x), \text{now}(\text{step now } y)\}$$

Parallel Computation

$$\begin{array}{ccccc}
\mathcal{PLL} & \xrightarrow{\lambda\mathcal{L}} & \mathcal{LPL} & \xrightarrow{\mathcal{L}\lambda} & \mathcal{LLP} \\
\downarrow{\scriptstyle \mathcal{P}\mu^{\mathcal{L}}} & & & & {\scriptstyle \mu^{\mathcal{L}}\mathcal{P}}\downarrow \\
\mathcal{PL} & & \xrightarrow{\qquad\lambda\qquad} & & \mathcal{LP}
\end{array}$$

$$\{\text{step now}(\text{now } x), \text{now}(\text{step now } y)\} \downarrow_{\mathcal{P}\mu^{\mathcal{L}}} \{\text{step now } x, \text{step now } y\}$$

$$\xrightarrow{\lambda} \text{step now}\{x, y\}$$

$$\{\text{step now}(\text{now } x), \text{now}(\text{step now } y)\} \xrightarrow{\lambda\mathcal{L}} \text{step now}(\{\text{now } x, \text{step now } y\})$$

Parallel Computation

$$
\begin{array}{ccccc}
\mathcal{PLL} & \xrightarrow{\lambda\mathcal{L}} & \mathcal{LPL} & \xrightarrow{\mathcal{L}\lambda} & \mathcal{LLP} \\
\downarrow{\scriptstyle \mathcal{P}\mu^{\mathcal{L}}} & & & & \downarrow{\scriptstyle \mu^{\mathcal{L}}\mathcal{P}} \\
\mathcal{PL} & & \xrightarrow{\qquad\lambda\qquad} & & \mathcal{LP}
\end{array}
$$

$$\{\mathsf{step}\ \mathsf{now}(\mathsf{now}\ x), \mathsf{now}(\mathsf{step}\ \mathsf{now}\ y)\} \downarrow_{\mathcal{P}\mu^{\mathcal{L}}} \{\mathsf{step}\ \mathsf{now}\ x, \mathsf{step}\ \mathsf{now}\ y\}$$
$$\xrightarrow{\lambda} \mathsf{step}\ \mathsf{now}\{x, y\}$$

$$\{\mathsf{step}\ \mathsf{now}(\mathsf{now}\ x), \mathsf{now}(\mathsf{step}\ \mathsf{now}\ y)\} \xrightarrow{\lambda\mathcal{L}} \mathsf{step}\ \mathsf{now}(\{\mathsf{now}\ x, \mathsf{step}\ \mathsf{now}\ y\})$$
$$\xrightarrow{\mathcal{L}\lambda} \mathsf{step}\ \mathsf{now}(\mathsf{step}\ \mathsf{now}\{x, y\})$$

Parallel Computation

$$
\begin{array}{ccccc}
\mathcal{P}\mathcal{L}\mathcal{L} & \xrightarrow{\lambda\mathcal{L}} & \mathcal{L}\mathcal{P}\mathcal{L} & \xrightarrow{\mathcal{L}\lambda} & \mathcal{L}\mathcal{L}\mathcal{P} \\
\downarrow{\scriptstyle \mathcal{P}\mu^{\mathcal{L}}} & & & & \downarrow{\scriptstyle \mu^{\mathcal{L}}\mathcal{P}} \\
\mathcal{P}\mathcal{L} & & \xrightarrow{\qquad\lambda\qquad} & & \mathcal{L}\mathcal{P}
\end{array}
$$

$$
\begin{aligned}
\{\text{step now}(\text{now } x), \text{now}(\text{step now } y)\} &\downarrow_{\mathcal{P}\mu^{\mathcal{L}}} \{\text{step now } x, \text{step now } y\} \\
&\xrightarrow{\lambda} \text{step now}\{x, y\}
\end{aligned}
$$

$$
\begin{aligned}
\{\text{step now}(\text{now } x), \text{now}(\text{step now } y)\} &\xrightarrow{\lambda\mathcal{L}} \text{step now}(\{\text{now } x, \text{step now } y\}) \\
&\xrightarrow{\mathcal{L}\lambda} \text{step now}(\text{step now}\{x, y\}) \\
&\downarrow_{\mu^{\mathcal{L}}\mathcal{P}} \text{step step now}\{x, y\}
\end{aligned}
$$

Parallel Computation

So what went wrong?
Nothing specific to Powerset!
Only ingredient: "structure with two elements".

Theorem

Parallel computation is **never** *a distributive law for $\mathcal{ML} \to \mathcal{LM}$ if $\mathcal{M}$ is presented by a theory with a binary term.*

No Hope for Powerset

What can we do?

$$\lambda\{\mathsf{now}\, x, \mathsf{now}\, y\} = \mathsf{now}\{x, y\}$$
$$\lambda\{\mathsf{step}\, d, \mathsf{now}\, y\} = \mathsf{step}(\lambda\{d, \mathsf{now}\, y\})$$
$$\lambda\{\mathsf{step}\, d, \mathsf{step}\, d'\} = \mathsf{step}(\lambda\{d, d'\})$$

No Hope for Powerset

What can we do?

$$\lambda\{\mathsf{now}\, x, \mathsf{now}\, y\} = \mathsf{now}\{x, y\}$$
$$\lambda\{\mathsf{step}\, d, \mathsf{now}\, y\} = \mathsf{step}(\lambda\{d, \mathsf{now}\, y\})$$
$$\underline{\lambda\{\mathsf{step}\, d, \mathsf{step}\, d'\}} = \mathsf{step}(\lambda\{d, d'\})$$

No Hope for Powerset

What can we do?

$$
\begin{aligned}
\lambda\{\mathrm{now}\, x, \mathrm{now}\, y\} &= \mathrm{now}\{x, y\} \\
\lambda\{\mathrm{step}\, d, \mathrm{now}\, y\} &= \mathrm{step}(\lambda\{d, \mathrm{now}\, y\}) \\
\underline{\lambda\{\mathrm{step}\, d, \mathrm{step}\, d'\}} &= \mathrm{step}(\lambda\{d, d'\})
\end{aligned}
$$

$$
\begin{array}{ccc}
\mathcal{PL}(2) & \xrightarrow{\lambda} & \mathcal{LP}(2) \\
\downarrow{\scriptstyle \mathcal{PL}f} & & \downarrow{\scriptstyle \mathcal{LP}f} \\
\mathcal{PL}(1) & \xrightarrow{\lambda} & \mathcal{LP}(1)
\end{array}
\qquad
\begin{array}{ccc}
\{\mathrm{step}\,\mathrm{now}\, x, \mathrm{step}\,\mathrm{now}\, y\} & \xmapsto{\lambda} & ? \\
\downarrow{\scriptstyle \mathcal{PL}f} & & \downarrow{\scriptstyle \mathcal{LP}f} \\
\{\mathrm{step}\,\mathrm{now}\, x\} & \xmapsto{\lambda} & \mathrm{step}\,\mathrm{now}\{x\}
\end{array}
$$

No Hope for Powerset

What can we do?

$$\begin{aligned}
\lambda\{\text{now}\,x, \text{now}\,y\} &= \text{now}\{x, y\} \\
\lambda\{\text{step}\,d, \text{now}\,y\} &= \text{step}(\lambda\{d, \text{now}\,y\}) \\
\underline{\lambda\{\text{step}\,d, \text{step}\,d'\}} &= \text{step}(\lambda\{d, d'\})
\end{aligned}$$

$$\begin{array}{ccc}
\mathcal{PL}(2) & \xrightarrow{\lambda} & \mathcal{LP}(2) \\
\downarrow{\scriptstyle \mathcal{PL}f} & & \downarrow{\scriptstyle \mathcal{LP}f} \\
\mathcal{PL}(1) & \xrightarrow{\lambda} & \mathcal{LP}(1)
\end{array}
\qquad
\begin{array}{ccc}
\{\text{step now}\,x, \text{step now}\,y\} & \xmapsto{\lambda} & \text{step now}\{?\} \\
\downarrow{\scriptstyle \mathcal{PL}f} & & \downarrow{\scriptstyle \mathcal{LP}f} \\
\{\text{step now}\,x\} & \xmapsto{\lambda} & \text{step now}\{x\}
\end{array}$$

No Hope for Powerset

What can we do?

$$\begin{aligned}\lambda\{\mathsf{now}\,x, \mathsf{now}\,y\} &= \mathsf{now}\{x, y\}\\ \lambda\{\mathsf{step}\,d, \mathsf{now}\,y\} &= \mathsf{step}(\lambda\{d, \mathsf{now}\,y\})\\ \underline{\lambda\{\mathsf{step}\,d, \mathsf{step}\,d'\}} &= \mathsf{step}(\lambda\{d, d'\})\end{aligned}$$

$$\begin{array}{ccc} \mathcal{PL}(2) & \xrightarrow{\lambda} & \mathcal{LP}(2) \\ \downarrow{\scriptstyle \mathcal{PL}f} & & \downarrow{\scriptstyle \mathcal{LP}f} \\ \mathcal{PL}(1) & \xrightarrow{\lambda} & \mathcal{LP}(1) \end{array} \qquad \begin{array}{ccc} \{\mathsf{step\,now}\,x, \mathsf{step\,now}\,y\} & \stackrel{\lambda}{\longmapsto} & \mathsf{step\,now}\{?\} \\ \downarrow{\scriptstyle \mathcal{PL}f} & & \downarrow{\scriptstyle \mathcal{LP}f} \\ \{\mathsf{step\,now}\,x\} & \stackrel{\lambda}{\longmapsto} & \mathsf{step\,now}\{x\} \end{array}$$

$$\{?\} = \varnothing \quad \{?\} = \{x\} \quad \{?\} = \{y\} \quad \{?\} = \{x, y\}$$

No Hope for Powerset

What can we do?

$$\lambda\{\mathsf{now}\,x, \mathsf{now}\,y\} = \mathsf{now}\{x, y\}$$
$$\lambda\{\mathsf{step}\,d, \mathsf{now}\,y\} = \mathsf{step}(\lambda\{d, \mathsf{now}\,y\})$$
$$\underline{\lambda\{\mathsf{step}\,d, \mathsf{step}\,d'\}} = \mathsf{step}(\lambda\{d, d'\})$$

$$\begin{array}{ccc} \mathcal{PL}(2) & \xrightarrow{\lambda} & \mathcal{LP}(2) \\ \downarrow{\scriptstyle \mathcal{PL}f} & & \downarrow{\scriptstyle \mathcal{LP}f} \\ \mathcal{PL}(1) & \xrightarrow{\lambda} & \mathcal{LP}(1) \end{array} \qquad \begin{array}{ccc} \{\mathsf{step}\,\mathsf{now}\,x, \mathsf{step}\,\mathsf{now}\,y\} & \longmapsto^{\lambda} & \mathsf{step}\,\mathsf{now}\{?\} \\ \downarrow{\scriptstyle \mathcal{PL}f} & & \downarrow{\scriptstyle \mathcal{LP}f} \\ \{\mathsf{step}\,\mathsf{now}\,x\} & \longmapsto^{\lambda} & \mathsf{step}\,\mathsf{now}\{x\} \end{array}$$

$$\{?\} = \varnothing \quad \{?\} = \{x\} \quad \{?\} = \{y\} \quad \{?\} = \{x, y\}$$

No Hope for Powerset

What can we do?

$$\lambda\{\text{now } x, \text{now } y\} = \text{now}\{x, y\}$$
$$\lambda\{\text{step } d, \text{now } y\} = \text{step}(\lambda\{d, \text{now } y\})$$
$$\underline{\lambda\{\text{step } d, \text{step } d'\}} = \text{step}(\lambda\{d, d'\})$$

$$\begin{array}{ccc} \mathcal{P}\mathcal{L}(2) & \xrightarrow{\lambda} & \mathcal{L}\mathcal{P}(2) \\ \downarrow{\scriptstyle \mathcal{P}\mathcal{L}f} & & \downarrow{\scriptstyle \mathcal{L}\mathcal{P}f} \\ \mathcal{P}\mathcal{L}(1) & \xrightarrow{\lambda} & \mathcal{L}\mathcal{P}(1) \end{array} \qquad \begin{array}{ccc} \{\text{step now } x, \text{step now } y\} & \overset{\lambda}{\longmapsto} & \text{step now}\{?\} \\ \downarrow{\scriptstyle \mathcal{P}\mathcal{L}f} & & \downarrow{\scriptstyle \mathcal{L}\mathcal{P}f} \\ \{\text{step now } x\} & \overset{\lambda}{\longmapsto} & \text{step now}\{x\} \end{array}$$

$$\{?\} = \varnothing \qquad \{?\} = \{x\} \qquad \{?\} = \{y\} \qquad \{?\} = \{x, y\}$$

No Hope for Powerset

What can we do?

$$
\begin{aligned}
\lambda\{\mathsf{now}\ x, \mathsf{now}\ y\} &= \mathsf{now}\{x, y\} \\
\lambda\{\mathsf{step}\ d, \mathsf{now}\ y\} &= \mathsf{step}(\lambda\{d, \mathsf{now}\ y\}) \\
\lambda\{\mathsf{step}\ d, \mathsf{step}\ d'\} &= \mathsf{step}(\lambda\{d, d'\})
\end{aligned}
$$

$$
\begin{array}{ccc}
\mathcal{PL}(2) & \xrightarrow{\lambda} & \mathcal{LP}(2) \\
\downarrow{\scriptstyle \mathcal{PL}f} & & \downarrow{\scriptstyle \mathcal{LP}f} \\
\mathcal{PL}(1) & \xrightarrow{\lambda} & \mathcal{LP}(1)
\end{array}
\qquad
\begin{array}{ccc}
\{\mathsf{step\ now}\ x, \mathsf{step\ now}\ y\} & \overset{\lambda}{\longmapsto} & \mathsf{step\ now}\{?\} \\
\downarrow{\scriptstyle \mathcal{PL}f} & & \downarrow{\scriptstyle \mathcal{LP}f} \\
\{\mathsf{step\ now}\ x\} & \overset{\lambda}{\longmapsto} & \mathsf{step\ now}\{x\}
\end{array}
$$

$\{?\} = \varnothing \quad \{?\} = \{x\} \quad \{?\} = \{y\} \quad \{?\} = \{x, y\}$

No Hope for Distribution

What can we do?

$$
\begin{aligned}
\lambda(\mathsf{now}\, x +_p \mathsf{now}\, y) &= \mathsf{now}(x +_p y) \\
\lambda(\mathsf{step}\, d +_p \mathsf{now}\, y) &= \mathsf{step}(\lambda(d +_p \mathsf{now}\, y)) \\
\lambda(\mathsf{step}\, d +_p \mathsf{step}\, d') &= \mathsf{step}(\lambda(d +_p d'))
\end{aligned}
$$

$$
\begin{array}{ccc}
\mathcal{DL}(2) & \xrightarrow{\lambda} & \mathcal{LD}(2) \\
\downarrow{\scriptstyle \mathcal{DL}f} & & \downarrow{\scriptstyle \mathcal{LD}f} \\
\mathcal{DL}(1) & \xrightarrow{\lambda} & \mathcal{LD}(1)
\end{array}
\qquad
\begin{array}{ccc}
(\mathsf{step}\,\mathsf{now}\, x) +_p (\mathsf{step}\,\mathsf{now}\, y) & \overset{\lambda}{\longmapsto} & \mathsf{step}\,\mathsf{now}\, ? \\
\downarrow{\scriptstyle \mathcal{DL}f} & & \downarrow{\scriptstyle \mathcal{LD}f} \\
\delta(\mathsf{step}\,\mathsf{now}\, x) & \overset{\lambda}{\longmapsto} & \mathsf{step}\,\mathsf{now}\, \delta_x
\end{array}
$$

$$
? = \delta_x \qquad ? = \delta_y \qquad ? = x +_q y
$$

No Hope in General?

Theorem

There is no **causal** *distributive law* $\mathcal{ML} \to \mathcal{LM}$*, if* $\mathcal{M}$ *is presented by a theory with an idempotent and commutative term.*

[1] Rasmus Møgelberg and Maaike Zwart - What Monads Can and Cannot Do with a Bit of Extra Time. Doi: 10.4230/LIPIcs.CSL.2024.39

No Hope in General?

Theorem

There is no **causal** *distributive law $\mathcal{ML} \to \mathcal{LM}$, if $\mathcal{M}$ is presented by a theory with an idempotent and commutative term.*

$$\mathcal{ML}^{\kappa} \to \mathcal{L}^{\kappa}\mathcal{M} \Rightarrow \mathcal{ML} \to \mathcal{LM}$$

[1]Rasmus Møgelberg and Maaike Zwart - What Monads Can and Cannot Do with a Bit of Extra Time. Doi: 10.4230/LIPIcs.CSL.2024.39

No Hope in General?

Theorem

There is no **causal** *distributive law* $\mathcal{ML} \to \mathcal{LM}$*, if* $\mathcal{M}$ *is presented by a theory with an idempotent and commutative term.*

$$\mathcal{ML}^{\kappa} \to \mathcal{L}^{\kappa}\mathcal{M} \Rightarrow \mathcal{ML} \to \mathcal{LM}$$

$$\mathcal{ML} \to \mathcal{LM} \not\Rightarrow \mathcal{ML}^{\kappa} \to \mathcal{L}^{\kappa}\mathcal{M}$$

Example in the paper.[1]

[1] Rasmus Møgelberg and Maaike Zwart - What Monads Can and Cannot Do with a Bit of Extra Time. Doi: 10.4230/LIPIcs.CSL.2024.39

A bit of Hope!

Main problem so far:

$$\text{step}\,\text{step} \neq \text{step}$$

A bit of Hope!

Main problem so far:

$$\mathsf{step}\,\mathsf{step} \neq \mathsf{step}$$

But what if:

$$\mathsf{step}\,\mathsf{step} \sim \mathsf{step}$$

A bit of Hope!

Main problem so far:

$$\text{step}\,\text{step} \neq \text{step}$$

But what if:

$$\text{step}\,\text{step} \sim \text{step}$$

Theorem

Parallel computation defines a distributive law $\mathcal{ML} \to \mathcal{LM}$ **up to weak bisimilarity**, *if $\mathcal{M}$ is presented by a theory with no drop equations.*

✓ $x * x = x$ × $x * y = x$

Free Powerset + Delay

May convergence[2]

$$\mathcal{P}^{\kappa}_{\diamond}(A) \simeq \mathcal{P}(A + \rhd^{\kappa} \mathcal{P}^{\kappa}_{\diamond}(A))$$

Free monad of join semilattices + delay algebra

[2] Rasmus Møgelberg and Andrea Vezzosi - Two guarded recursive powerdomains for applicative simulation. Doi: 10.4204/EPTCS.351.13

Free Powerset + Delay

May convergence[2]

$$\mathcal{P}^{\kappa}_{\diamond}(A) \simeq \mathcal{P}(A + \rhd^{\kappa} \mathcal{P}^{\kappa}_{\diamond}(A))$$

Free monad of **join semilattices** + delay algebra

$$\begin{aligned} 1 * x &= x \\ x * (y * z) &= (x * y) * z \\ x * y &= y * x \\ x * x &= x \end{aligned}$$

[2] Rasmus Møgelberg and Andrea Vezzosi - Two guarded recursive powerdomains for applicative simulation. Doi: 10.4204/EPTCS.351.13

Free Powerset + Delay

May convergence[2]

$$\mathcal{P}^{\kappa}_{\diamond}(A) \simeq \mathcal{P}(A + \triangleright^{\kappa} \mathcal{P}^{\kappa}_{\diamond}(A))$$

Free monad of join semilattices + **delay algebra**

$$1 * x = x$$
$$x * (y * z) = (x * y) * z$$
$$x * y = y * x$$
$$x * x = x$$

$$\mathrm{now}(x) = \{\mathrm{inl}\, x\}$$
$$\mathrm{step}(x) = \{\mathrm{inr}\, x\}$$

[2] Rasmus Møgelberg and Andrea Vezzosi - Two guarded recursive powerdomains for applicative simulation. Doi: 10.4204/EPTCS.351.13

Free Powerset + Delay

May convergence[2]

$$\mathcal{P}^{\kappa}_{\diamond}(A) \simeq \mathcal{P}(A + \rhd^{\kappa} \mathcal{P}^{\kappa}_{\diamond}(A))$$

Free monad of join semilattices + delay algebra

$$\begin{aligned} 1 * x &= x \\ x * (y * z) &= (x * y) * z \\ x * y &= y * x \\ x * x &= x \end{aligned}$$

$$\begin{aligned} \mathrm{now}(x) &= \{\mathrm{inl}\, x\} \\ \mathrm{step}(x) &= \{\mathrm{inr}\, x\} \end{aligned}$$

[2] Rasmus Møgelberg and Andrea Vezzosi - Two guarded recursive powerdomains for applicative simulation. Doi: 10.4204/EPTCS.351.13

Free Distribution + Delay[3]

$$\mathcal{D}^{\kappa}_{\diamond}(A) \simeq \mathcal{D}(A + \rhd^{\kappa} \mathcal{D}^{\kappa}_{\diamond}(A))$$

Free monad of convex algebras + delay algebra

$$\begin{aligned} x +_1 y &= x \\ x +_p x &= x \\ x +_p y &= y +_{(1-p)} x \\ (x +_p y) +_q z &= x +_{pq} (y +_{\frac{q-pq}{1-pq}} z) \end{aligned}$$

$$\begin{aligned} \mathsf{now}(x) &= \delta_{\mathsf{inl}\, x} \\ \mathsf{step}(x) &= \delta_{\mathsf{inr}\, x} \end{aligned}$$

[3] Philipp Stassen, Rasmus Møgelberg, Maaike Zwart, Alejandro Aguirre and Lars Birkedal - Modelling Probabilistic FPC in Guarded Type Theory. Under review.

Free $\mathcal{M}$ + Delay

$$\mathcal{M}^{\kappa}_{\diamond}(A) \simeq \mathcal{M}(A + \rhd^{\kappa} \mathcal{M}^{\kappa}_{\diamond}(A))$$

Free monad of $\mathcal{M}$-algebras + delay algebra

Conclusion

We saw:

- Sequential computation gives $\lambda : \mathcal{ML} \to \mathcal{LM}$ for balanced $\mathcal{M}$.
- Parallel computation gives $\lambda : \mathcal{ML} \to \mathcal{LM}$ for non-drop $\mathcal{M}$, but only up to weak bisimilarity.
- Distributive law $\mathcal{PL} \to \mathcal{LP}$ impossible.
- Distributive law $\mathcal{DL} \to \mathcal{LD}$ impossible.
- Causal distributive law $\mathcal{ML} \to \mathcal{LM}$ impossible for idempotent and commutative $\mathcal{M}$.

Conclusion

We saw:

- Sequential computation gives $\lambda : \mathcal{ML} \to \mathcal{LM}$ for balanced $\mathcal{M}$.
- Parallel computation gives $\lambda : \mathcal{ML} \to \mathcal{LM}$ for non-drop $\mathcal{M}$, but only up to weak bisimilarity.
- Distributive law $\mathcal{PL} \to \mathcal{LP}$ impossible.
- Distributive law $\mathcal{DL} \to \mathcal{LD}$ impossible.
- Causal distributive law $\mathcal{ML} \to \mathcal{LM}$ impossible for idempotent and commutative $\mathcal{M}$.

More in the papers!

- Combinations of Delay with Exceptions, Reader, State, Selection ...
- How to reason about probabilistic programs with $\mathcal{D}^{\kappa}_{\diamond}$.