Layer by Layer: combining monads

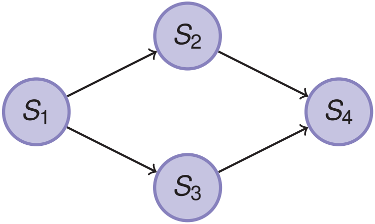
Fredrik Dahlqvist, Alexandra Silva, Louis Parlant

October 16, 2018

Motivation: ProbNetKAT

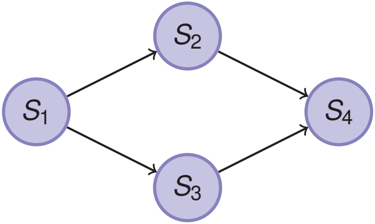
Motivation: ProbNetKAT

A simple network:



Motivation: ProbNetKAT

A simple network:

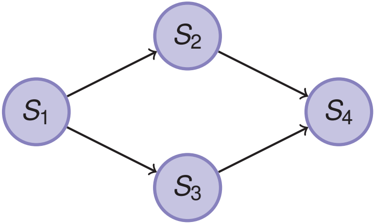


Topology:

$$
\begin{aligned}
t =& (sw = S_1; pt = 2; (sw \leftarrow S_2; pt \leftarrow 1) \oplus_{.9} \texttt{drop}) \\
&\&(sw = S_1; pt = 3; sw \leftarrow S_3; p \leftarrow 1) \\
&\&(sw = S_2; pt = 4; sw \leftarrow S_4; p \leftarrow 2) \\
&\&(sw = S_3; pt = 4; sw \leftarrow S_4; p \leftarrow 3)
\end{aligned}
$$

Motivation: ProbNetKAT

A simple network:



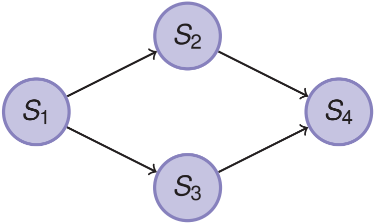
Topology:

$$\begin{aligned} t =& (sw = S_1; pt = 2; (sw \leftarrow S_2; pt \leftarrow 1) \oplus_{.9} \mathtt{drop}) \\ &\&(sw = S_1; pt = 3; sw \leftarrow S_3; p \leftarrow 1) \\ &\&(sw = S_2; pt = 4; sw \leftarrow S_4; p \leftarrow 2) \\ &\&(sw = S_3; pt = 4; sw \leftarrow S_4; p \leftarrow 3) \end{aligned}$$

Forwarding policy: $p = (sw = S_1; pt \leftarrow 2)\&(sw = S_2; pt \leftarrow 4)$

Motivation: ProbNetKAT

A simple network:



Topology:

$$\begin{aligned} t = &(sw = S_1; pt = 2; (sw \leftarrow S_2; pt \leftarrow 1) \oplus_{.9} \texttt{drop}) \\ &\&(sw = S_1; pt = 3; sw \leftarrow S_3; p \leftarrow 1) \\ &\&(sw = S_2; pt = 4; sw \leftarrow S_4; p \leftarrow 2) \\ &\&(sw = S_3; pt = 4; sw \leftarrow S_4; p \leftarrow 3) \end{aligned}$$

Forwarding policy: $p = (sw = S_1; pt \leftarrow 2)\&(sw = S_2; pt \leftarrow 4)$

A packet reaches S_4: $(t; p)^*; (sw = S_4)$

Motivation: ProbNetKAT

- Syntactically, ProbNetKAT is a kind of Kleene algebra with probabilistic choice $\oplus_\lambda$

Motivation: ProbNetKAT

- Syntactically, ProbNetKAT is a kind of Kleene algebra with probabilistic choice $\oplus_\lambda$
- BUT! The denotation of the operator & is odd...

Motivation: ProbNetKAT

- Syntactically, ProbNetKAT is a kind of Kleene algebra with probabilistic choice $\oplus_\lambda$
- BUT! The denotation of the operator & is odd...
  - & isn't idempotent.

Motivation: ProbNetKAT

- Syntactically, ProbNetKAT is a kind of Kleene algebra with probabilistic choice $\oplus_\lambda$
- BUT! The denotation of the operator & is odd...
  - & isn't idempotent.
  - ; does not distribute over &

Motivation: ProbNetKAT

- Syntactically, ProbNetKAT is a kind of Kleene algebra with probabilistic choice $\oplus_\lambda$
- BUT! The denotation of the operator & is odd...
  - & isn't idempotent.
  - ; does not distribute over &
- Why? What's going on?

Motivation: ProbNetKAT

- Syntactically, ProbNetKAT is a kind of Kleene algebra with probabilistic choice $\oplus_\lambda$
- BUT! The denotation of the operator & is odd...
  - & isn't idempotent.
  - ; does not distribute over &
- Why? What's going on?
- General question:

  How can we add features in a principled and controllable manner?

Building languages layer by layer: shopping list

Building languages layer by layer: shopping list

- First layer:
  $\mathtt{p} ::= \mathtt{skip} \mid \mathtt{p};\mathtt{p} \mid \mathtt{a} \in \mathtt{At}$
  $\mathtt{p};\mathtt{skip} = \mathtt{skip};\mathtt{p} = \mathtt{p}, \ldots$
  Monad: $(-)^*$

Building languages layer by layer: shopping list

- Second layer:
  $\mathtt{p} ::= \mathtt{abort} \mid \mathtt{p} + \mathtt{p} \mid \mathtt{a} \in \mathtt{At}$
  $\mathtt{p} + \mathtt{abort} = \mathtt{abort} + \mathtt{p} = \mathtt{p},$
  $\mathtt{p} + \mathtt{q} = \mathtt{q} + \mathtt{p}, \mathtt{p} + \mathtt{p} = \mathtt{p}, \ldots$
  Monad: $\mathcal{P}$
- First layer:
  $\mathtt{p} ::= \mathtt{skip} \mid \mathtt{p}; \mathtt{p} \mid \mathtt{a} \in \mathtt{At}$
  $\mathtt{p}; \mathtt{skip} = \mathtt{skip}; \mathtt{p} = \mathtt{p}, \ldots$
  Monad: $(-)^*$

Building languages layer by layer: shopping list

- Topping:
  $\mathtt{p} ::= \mathtt{p} \oplus_r \mathtt{p} \mid \mathtt{a} \in \mathtt{At}$
  $\mathtt{p} \oplus_r \mathtt{q} = \mathtt{q} \oplus_{1-r} \mathtt{p}, \ldots$
  Monad: $\mathcal{D}$
- Second layer:
  $\mathtt{p} ::= \mathtt{abort} \mid \mathtt{p} + \mathtt{p} \mid \mathtt{a} \in \mathtt{At}$
  $\mathtt{p} + \mathtt{abort} = \mathtt{abort} + \mathtt{p} = \mathtt{p},$
  $\mathtt{p} + \mathtt{q} = \mathtt{q} + \mathtt{p}, \mathtt{p} + \mathtt{p} = \mathtt{p}, \ldots$
  Monad: $\mathcal{P}$
- First layer:
  $\mathtt{p} ::= \mathtt{skip} \mid \mathtt{p}; \mathtt{p} \mid \mathtt{a} \in \mathtt{At}$
  $\mathtt{p}; \mathtt{skip} = \mathtt{skip}; \mathtt{p} = \mathtt{p}, \ldots$
  Monad: $(-)^*$

Combining the layers: things can go wrong!

Combining the layers: things can go wrong!

- The composition of two monads is not necessarily a monad

Combining the layers: things can go wrong!

- The composition of two monads is not necessarily a monad
- Combine monads S, T via distributive law

$$\lambda : ST \to TS$$

Combining the layers: things can go wrong!

- The composition of two monads is not necessarily a monad
- Combine monads S, T via distributive law

$$\lambda : ST \to TS$$

- No distributive law $\mathcal{P}\mathcal{D} \to \mathcal{D}\mathcal{P}$

Combining the layers: things can go wrong!

- The composition of two monads is not necessarily a monad
- Combine monads S, T via distributive law

$$\lambda : ST \to TS$$

- No distributive law $\mathcal{P}\mathcal{D} \to \mathcal{D}\mathcal{P}$
- But there exists a distributive law $(-)^*\mathcal{P} \to \mathcal{P}(-)^*$

Combining the layers: things can go wrong!

- The composition of two monads is not necessarily a monad
- Combine monads S, T via distributive law

$$\lambda : ST \to TS$$

- No distributive law $\mathcal{PD} \to \mathcal{DP}$
- But there exists a distributive law $(-)^*\mathcal{P} \to \mathcal{P}(-)^*$
- How do we deal with this systematically?

This paper

A general and modular approach for determining :

(a) if a monad combination by distributive law is possible;
(b) if it is not possible, exactly which features are broken by the extension; and
(c) suggests a way to fix the composition by modifying one of the monads.

Monads

- Monads: a categorical way to encode computational effects:

Monads

- Monads: a categorical way to encode computational effects: Non-determinism, probabilities, side-effects ...

Monads

- Monads: a categorical way to encode computational effects: Non-determinism, probabilities, side-effects . . .
- Applications of monads include programming language semantics, automata theory, etc.
- It is convenient to compositionally *combine* several effects.

Definitions

Definition

A Monad (T, η, μ) on a category C is:

- An endofunctor $T : C \to C$
- A natural transformation $\eta : 1 \to T$
- A natural transformation $\mu : TT \to T$

(Verifying some structural properties)

We will consider monads on Set.

Examples

Finite Powerset	$\mathcal{P}(A) = \{B \mid B \subseteq A, B \text{ finite}\}$
Free Monoid (List)	$A^* = \{w_1 \dots w_n \mid n \in \mathbb{N}, w_i \in A\}$
Distributions	$\mathcal{D}(A) = \{f \mid f \text{ probability distribution on } A,$ and $Supp(f)$ finite$\}$

Algebras

Definition

An *algebra* for the monad T is an object A together with a morphism $\alpha : TA \to A$.
(Verifying some structural properties involving η and μ)

Algebras

Definition

An *algebra* for the monad T is an object A together with a morphism $\alpha : TA \to A$.
(Verifying some structural properties involving η and μ)

Definition

For a signature Σ and a set of equations E we can define a monad T such that $\mathbf{EM}(T) \simeq \mathbf{Alg}(\Sigma, E)$

Examples

	Σ	E

$\mathcal{P}$	`0, +`	`x+0=0+x=x`
		`x+y=y+x`
		`(x+y)+z=x+(y+z)`
		`x+x=x`
		(join-semilattice)

$(-)^*$	`1, ;`	`x;1=1;x=x`
		`(x;y);z=x;(y;z)`
		(monoid)

S, T monads, $\mathbf{EM}(T) \simeq \mathbf{Alg}(\Sigma_T, E_T)$, $\mathbf{EM}(S) \simeq \mathbf{Alg}(\Sigma_S, E_S)$

Definition

A *distributive law* of T over S is a natural transformation $\lambda : ST \to TS$ (verifying structural properties)

S, T monads, $\mathbf{EM}(T) \simeq \mathbf{Alg}(\Sigma_T, E_T)$, $\mathbf{EM}(S) \simeq \mathbf{Alg}(\Sigma_S, E_S)$

Definition

A *distributive law* of T over S is a natural transformation $\lambda : ST \to TS$ (verifying structural properties)

If T distributes over S, then:

- TS is a monad

$$\begin{tikzcd} X \arrow[r, "\eta^T_X"] \arrow[rr, bend right, "u"'] & TX \arrow[r, "\eta^S_{TX}"] & STX \end{tikzcd} \qquad \begin{tikzcd} STSTX \arrow[r, "S\lambda_{TX}"] \arrow[rrr, bend right, "m"'] & SSTTX \arrow[r, "\mu^S_{TTX}"] & STTX \arrow[r, "S\mu^T_X"] & STX \end{tikzcd}$$

- Operations in Σ_S distribute over those of Σ_T

We call S the *inner layer*, T the *outer layer*.

Remarks and questions:

- Distributive laws are one of the go-to methods to **compose monads**
- Implements a one-way distributivity of algebraic operations
- For two given monads, how to know whether there exists a distributive law?
- How to build it?

Remarks and questions:

- Distributive laws are one of the go-to methods to **compose monads**
- Implements a one-way distributivity of algebraic operations
- For two given monads, how to know whether there exists a distributive law?
- How to build it?

Theorem

Let T be a monoidal monad, then for any finitary signature Σ, there exists a distributive law $\lambda_\Sigma : H_\Sigma T \to TH_\Sigma$ of the polynomial functor associated with Σ over T.

Remarks and questions:

- Distributive laws are one of the go-to methods to **compose monads**
- Implements a one-way distributivity of algebraic operations
- For two given monads, how to know whether there exists a distributive law?
- How to build it?

Theorem

Let T be a monoidal monad, then for any finitary signature Σ, there exists a distributive law $\lambda_\Sigma : H_\Sigma T \to TH_\Sigma$ of the polynomial functor associated with Σ over T.

Monoidal helps with lifting operations but not equations.

The procedure: 1. Build ‘candidate’ $\lambda : ST \to TS$

The procedure: 1. Build ‘candidate’ $\lambda : ST \to TS$

- S always given by signature Σ and equations E

The procedure: 1. Build 'candidate' $\lambda : ST \to TS$

- S always given by signature Σ and equations E
- Use monoidal 'tensor'

$$\otimes_{-,-} : T(-) \times T(-) \to T(- \times -)$$

The procedure: 1. Build 'candidate' $\lambda : ST \to TS$

- S always given by signature Σ and equations E
- Use monoidal 'tensor'

$$\otimes_{-,-} : T(-) \times T(-) \to T(- \times -)$$

- Define a lifting $\widehat{T}$ of T on Σ-algebras

$$(A, \sigma : A^{\mathrm{ar}(\sigma)} \to A)_{\sigma \in \Sigma} \to (TA, T\sigma \circ \otimes^{\mathrm{ar}(\sigma)} : (TA)^{\mathrm{ar}(\sigma)} \to TA)_{\sigma \in \Sigma}$$

The procedure: 1. Build ‘candidate’ $\lambda : ST \to TS$

- S always given by signature Σ and equations E
- Use monoidal ‘tensor’

$$\otimes_{-,-} : T(-) \times T(-) \to T(- \times -)$$

- Define a lifting $\widehat{T}$ of T on Σ-algebras

$$(A, \sigma : A^{\mathrm{ar}(\sigma)} \to A)_{\sigma \in \Sigma} \to (TA, T\sigma \circ \otimes^{\mathrm{ar}(\sigma)} : (TA)^{\mathrm{ar}(\sigma)} \to TA)_{\sigma \in \Sigma}$$

- $\widehat{;} : (\mathcal{P}(\mathtt{At})^*)^2 \to \mathcal{P}(\mathtt{At}^*), (U, V) \mapsto \{u; v \mid u \in U, v \in V\}, \widehat{\mathtt{skip}} = \{\epsilon\}$

The procedure: 1. Build 'candidate' $\lambda : ST \to TS$

- S always given by signature Σ and equations E
- Use monoidal 'tensor'

$$\otimes_{-,-} : T(-) \times T(-) \to T(- \times -)$$

- Define a lifting $\widehat{T}$ of T on Σ-algebras

$$(A, \sigma : A^{\mathrm{ar}(\sigma)} \to A)_{\sigma\in\Sigma} \to (TA, T\sigma \circ \otimes^{\mathrm{ar}(\sigma)} : (TA)^{\mathrm{ar}(\sigma)} \to TA)_{\sigma\in\Sigma}$$

- $\hat{;} : (\mathcal{P}(\mathtt{At})^*)^2 \to \mathcal{P}(\mathtt{At}^*), (U, V) \mapsto \{u; v \mid u \in U, v \in V\}, \widehat{\mathtt{skip}} = \{\epsilon\}$

Theorem

If $\widehat{T}$ sends (Σ, E)-algebras to (Σ, E)-algebras, then it defines a distributive law $\lambda : ST \to TS$.

The procedure: 2. Check if $\widehat{T} : \mathbf{Alg}(\Sigma, E) \to \mathbf{Alg}(\Sigma, E)$

The procedure: 2. Check if $\widehat{T} : \mathbf{Alg}(\Sigma, E) \to \mathbf{Alg}(\Sigma, E)$

Illustration with idempotency, $(A, \bullet : A^2 \to A) \models x \bullet x = x$

The procedure: 2. Check if $\widehat{T} : \mathbf{Alg}(\Sigma, E) \to \mathbf{Alg}(\Sigma, E)$

Illustration with idempotency, $(A, \bullet : A^2 \to A) \models x \bullet x = x$

$$\begin{tikzcd} A \arrow[r, "\Delta"] \arrow[dr, "\mathrm{Id}_A"'] & A^2 \arrow[d, "\bullet"] \\ & A \end{tikzcd}$$

The procedure: 2. Check if $\widehat{T} : \mathbf{Alg}(\Sigma, E) \to \mathbf{Alg}(\Sigma, E)$

Illustration with idempotency, $(A, \bullet : A^2 \to A) \models x \bullet x = x$

$$\begin{tikzcd} TA \arrow[r, "T\Delta"] \arrow[dr, "\mathrm{Id}_A"'] & T(A^2) \arrow[d, "T\bullet"] \\ & TA \end{tikzcd}$$

The procedure: 2. Check if $\widehat{T} : \mathbf{Alg}(\Sigma, E) \to \mathbf{Alg}(\Sigma, E)$

Illustration with idempotency, $(A, \bullet : A^2 \to A) \models x \bullet x = x$

$$\begin{tikzcd}
 & (TA)^2 \arrow[d, "\otimes_{A,A}"] \\
TA \arrow[ur, "\Delta"] \arrow[r, "T\Delta"] \arrow[dr, "\mathrm{Id}_A"'] & T(A^2) \arrow[d, "T\bullet"] \\
 & TA
\end{tikzcd}$$

The procedure: 2. Check if $\widehat{T} : \mathbf{Alg}(\Sigma, E) \to \mathbf{Alg}(\Sigma, E)$

Illustration with idempotency, $(A, \bullet : A^2 \to A) \models x \bullet x = x$

$$\begin{tikzcd}
 & (TA)^2 \arrow[d, "\otimes_{A,A}"] \\
TA \arrow[ur, "\Delta"] \arrow[r, "T\Delta"] \arrow[dr, "\mathrm{Id}_A"'] & T(A^2) \arrow[d, "T\bullet"] \\
 & TA
\end{tikzcd}$$

- We call the upper triangle the *residual diagram* of $x \bullet x = x$.

The procedure: 2. Check if $\widehat{T} : \mathbf{Alg}(\Sigma, E) \to \mathbf{Alg}(\Sigma, E)$

Illustration with idempotency, $(A, \bullet : A^2 \to A) \models x \bullet x = x$

$$
\begin{tikzcd}
 & (TA)^2 \arrow[d, "\otimes_{A,A}"] \\
TA \arrow[ur, "\Delta"] \arrow[r, "T\Delta"] \arrow[dr, "\mathrm{Id}_A"'] & T(A^2) \arrow[d, "T\bullet"] \\
 & TA
\end{tikzcd}
$$

- We call the upper triangle the *residual diagram* of $x \bullet x = x$.
- If it commutes then $(TA, \hat{\bullet} : (TA)^2 \to TA) \models x \hat{\bullet} x = x$.

The procedure: 2. Check if $\widehat{T} : \mathbf{Alg}(\Sigma, E) \to \mathbf{Alg}(\Sigma, E)$

Illustration with idempotency, $(A, \bullet : A^2 \to A) \models x \bullet x = x$

$$\begin{tikzcd}
 & (TA)^2 \arrow[d, "\otimes_{A,A}"] \\
TA \arrow[ur, "\Delta"] \arrow[r, "T\Delta"] \arrow[dr, "\mathrm{Id}_A"'] & T(A^2) \arrow[d, "T\bullet"] \\
 & TA
\end{tikzcd}$$

- We call the upper triangle the *residual diagram* of $x \bullet x = x$.
- If it commutes then $(TA, \hat{\bullet} : (TA)^2 \to TA) \models x \hat{\bullet} x = x$.
- If it doesn't, we know exactly where the obstacle is and can troubleshoot accordingly.

Application: combining $(-)^*$, $\mathcal{P}$ and $\mathcal{D}$

- $\mathcal{P}$ sends monoids to monoids $\Rightarrow \mathcal{P}(-)^*$ is a monad: the free idempotent semiring monad.

Application: combining $(-)^*$, $\mathcal{P}$ and $\mathcal{D}$

- $\mathcal{P}$ sends monoids to monoids $\Rightarrow \mathcal{P}(-)^*$ is a monad: the free idempotent semiring monad.
- Distributivity and absorption law are enforced by design!

Application: combining $(-)^*$, $\mathcal{P}$ and $\mathcal{D}$

- $\mathcal{P}$ sends monoids to monoids $\Rightarrow \mathcal{P}(-)^*$ is a monad: the free idempotent semiring monad.
- Distributivity and absorption law are enforced by design!
- $\mathcal{D}$ with $\otimes$ given by the product measure makes the residual diagrams for associativity, units and commutativity commute...

Application: combining $(-)^*$, $\mathcal{P}$ and $\mathcal{D}$

- $\mathcal{P}$ sends monoids to monoids $\Rightarrow \mathcal{P}(-)^*$ is a monad: the free idempotent semiring monad.
- Distributivity and absorption law are enforced by design!
- $\mathcal{D}$ with $\otimes$ given by the product measure makes the residual diagrams for associativity, units and commutativity commute...
- but neither for IDEMPOTENCY, nor for DISTRIBUTIVITY!

Application: combining $(-)^*$, $\mathcal{P}$ and $\mathcal{D}$

- $\mathcal{P}$ sends monoids to monoids $\Rightarrow \mathcal{P}(-)^*$ is a monad: the free idempotent semiring monad.
- Distributivity and absorption law are enforced by design!
- $\mathcal{D}$ with $\otimes$ given by the product measure makes the residual diagrams for associativity, units and commutativity commute...
- but neither for IDEMPOTENCY, nor for DISTRIBUTIVITY!
- Troubleshooting: remove those axioms

Application: combining $(-)^*$, $\mathcal{P}$ and $\mathcal{D}$

- $\mathcal{P}$ sends monoids to monoids $\Rightarrow \mathcal{P}(-)^*$ is a monad: the free idempotent semiring monad.
- Distributivity and absorption law are enforced by design!
- $\mathcal{D}$ with $\otimes$ given by the product measure makes the residual diagrams for associativity, units and commutativity commute...
- but neither for IDEMPOTENCY, nor for DISTRIBUTIVITY!
- Troubleshooting: remove those axioms
- There IS a distributive law over $\mathcal{D}$ of the monad defined by

$$\mathrm{p}; \mathtt{skip} = \mathtt{skip}; \mathrm{p} = \mathrm{p}, \quad (\mathrm{p}; \mathrm{q}); \mathrm{r} = \mathrm{p}(\mathrm{q}; \mathrm{r}),$$

$$\mathrm{p} + \mathtt{abort} = \mathtt{abort} + \mathrm{p} = \mathrm{p}, \quad \mathrm{p} + \mathrm{q} = \mathrm{q} + \mathrm{p}, \quad (\mathrm{p} + \mathrm{q}) + \mathrm{r} = \mathrm{p} + (\mathrm{q} + \mathrm{r}),$$

$$\mathrm{p}; \mathtt{abort} = \mathtt{abort}; \mathrm{p} = \mathtt{abort}$$

Application: combining $(-)^*$, $\mathcal{P}$ and $\mathcal{D}$

- $\mathcal{P}$ sends monoids to monoids $\Rightarrow \mathcal{P}(-)^*$ is a monad: the free idempotent semiring monad.
- Distributivity and absorption law are enforced by design!
- $\mathcal{D}$ with $\otimes$ given by the product measure makes the residual diagrams for associativity, units and commutativity commute...
- but neither for IDEMPOTENCY, nor for DISTRIBUTIVITY!
- Troubleshooting: remove those axioms
- There IS a distributive law over $\mathcal{D}$ of the monad defined by

$$\mathrm{p}; \mathtt{skip} = \mathtt{skip}; \mathrm{p} = \mathrm{p}, \quad (\mathrm{p}; \mathrm{q}); \mathrm{r} = \mathrm{p}(\mathrm{q}; \mathrm{r}),$$

$$\mathrm{p} + \mathtt{abort} = \mathtt{abort} + \mathrm{p} = \mathrm{p}, \quad \mathrm{p} + \mathrm{q} = \mathrm{q} + \mathrm{p}, \quad (\mathrm{p} + \mathrm{q}) + \mathrm{r} = \mathrm{p} + (\mathrm{q} + \mathrm{r}),$$

$$\mathrm{p}; \mathtt{abort} = \mathtt{abort}; \mathrm{p} = \mathtt{abort}$$

- Completely consistent with the semantics of ProbNetKAT

Theorem

Let T be a commutative, relevant and affine monad. For all u and v, T preserves u = v.

Fixing composition – Method 1: changing the inner layer

Idea: remove the faulty laws from the inner layer.

Fixing composition – Method 1: changing the inner layer

Idea: remove the faulty laws from the inner layer.
$\mathbf{EM}(S) \simeq \mathbf{Alg}(\Sigma_S, E_S)$, $\mathbf{EM}(T) \simeq \mathbf{Alg}(\Sigma_T, E_T)$. Let E'_S be the subset of E_S containing the equations preserved by T.

- Obtain S' from $\mathbf{Alg}(\Sigma_S, E'_S)$
- Compose T with S', obtain a (Σ, E) algebra, where:

$$\Sigma = (\Sigma_T \cup \Sigma_S)$$

$$E = (E_T \cup E'_S \cup \text{ distributivity of } \Sigma_S \text{ over } \Sigma_T)$$

Method 1: fix the inner layer

Example

$\mathcal{D}$ does not preserve idempotency nor distributivity. Drop them and obtain a (Σ, E) algebra where $\Sigma = \{;, 1, +, 0, \oplus_\lambda\}$ and $E =$

- associativity, commutativity, unit laws for $+$
- equations of $(-)^*$
- absorption $\mathrm{p}; 0 = 0; \mathrm{p} = 0$
- equations of $\mathcal{D}$ (convex algebras)
- $p \,;\, (q \oplus_\lambda r) = (p \,;\, q) \oplus_\lambda (p \,;\, r)$
- $(q \oplus_\lambda r) \,;\, p = (q \,;\, p) \oplus_\lambda (r \,;\, p)$
- $p + (q \oplus_\lambda r) = (p + q) \oplus_\lambda (p + r)$
- $(q \oplus_\lambda r) + p = (q + p) \oplus_\lambda (r + p)$

Method 2: Change the outer layer

Idea: consider the largest submonad preserving the faulty equations.

Method 2: Change the outer layer

Idea: consider the largest submonad preserving the faulty equations.

Example

$\mathcal{P}\mathcal{D}$ is not a monad as $\mathcal{P}$ does not preserve idempotency. The largest submonad of $\mathcal{P}$ preserving it is the *convex powerset* $\mathcal{P}_c$

Method 2: Change the outer layer

Idea: consider the largest submonad preserving the faulty equations.

Example

$\mathcal{P}\mathcal{D}$ is not a monad as $\mathcal{P}$ does not preserve idempotency. The largest submonad of $\mathcal{P}$ preserving it is the *convex powerset* $\mathcal{P}_c$

Two options to fix $\mathcal{P}\mathcal{D}$:

1. Build a monad *PD* that preserves the relevant equations.
2. Replace $\mathcal{P}$ by $\mathcal{P}_c$ and then composition works: $\mathcal{P}_c\mathcal{D}$.

Conclusions

- A principled approach to constructing (equational) languages layer by layer.
- Conditions on existence of distributive laws and potential fixing strategies.
- Note: other troubleshooting strategies are possible!

Thanks

Thank you! Questions?