Distributive Laws of Monadic Containers

Chris Purdy (Royal Holloway, University of London)

Stefania Damato (University of Nottingham)

Containers

Semantics for a wide class of strictly positive data types.

A **container**[1] $S \lhd P$ consists of a set of **shapes** S and a family of **positions** $P : S \to \mathrm{Set}$.

Containers can be interpreted as functors on **Set**

$$[\![S \lhd P]\!]\, X := \sum_{s:S} (P\, s \to X)$$

This interpretation is functorial, and fully-faithful.

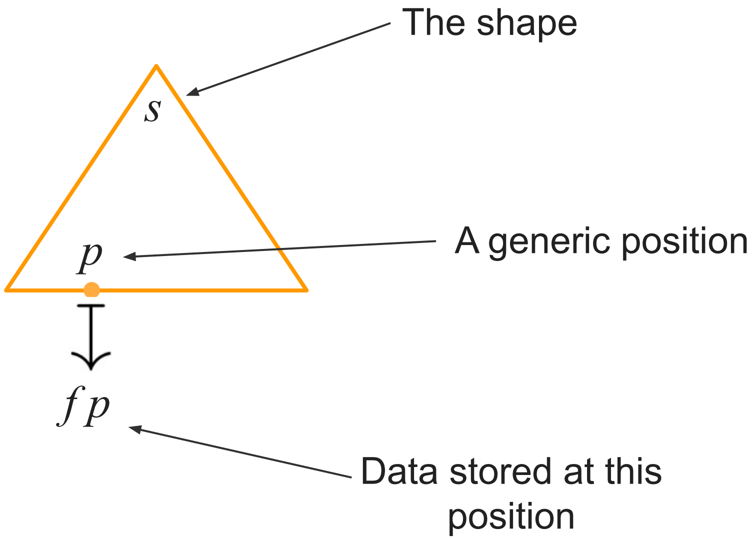
Visualised: the "Id" container
(one shape, one position)

[1] *[Abbott et al. 2005]*

Container interpretation

An element of $[\![S \triangleleft P]\!]\, X$ consists of a shape $s : S$ and a map $f : P\, s \to X$.

We can think of these using triangle diagrams.

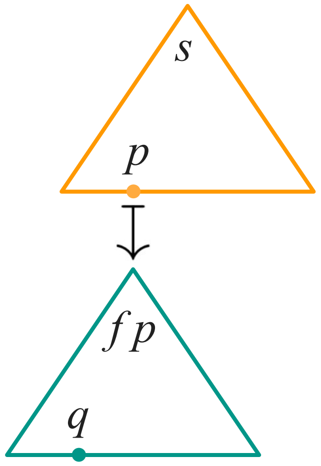


Containers compose

Containers are closed under composition.

$$(S \lhd P) \circ (T \lhd Q) := [\![S \lhd P]\!]\; T \lhd \left(\lambda(s, f).\, \sum_{p:P\,s} Q\,(f\,p) \right)$$

Intuition in terms of triangle diagrams:

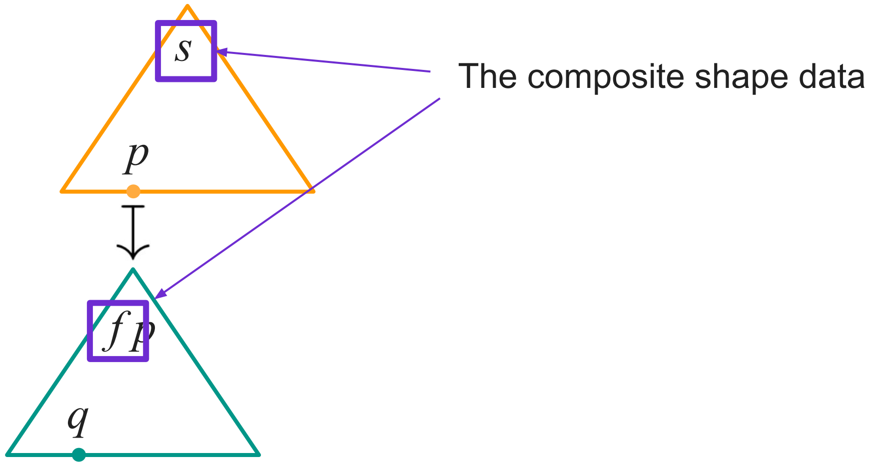


Containers compose

Containers are closed under composition.

$$(S \lhd P) \circ (T \lhd Q) := \boxed{[\![S \lhd P]\!]\ T} \lhd \left(\lambda(s, f).\ \sum_{p:P\,s} Q\,(f\,p)\right)$$

Intuition in terms of triangle diagrams:

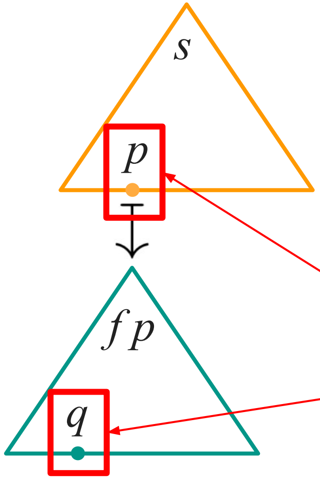


Containers compose

Containers are closed under composition.

$$(S \lhd P) \circ (T \lhd Q) := [\![S \lhd P]\!]\, T \lhd \left(\lambda(s, f).\, \boxed{\sum_{p:P\,s} Q\,(f\,p)}\right)$$

Intuition in terms of
triangle diagrams:



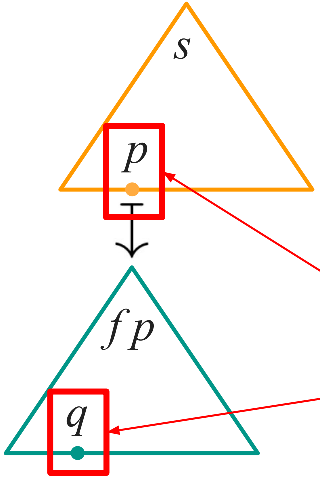
Each position in the
composite is
specified by a pair

Containers compose

Containers are closed under composition.

$$(S \lhd P) \circ (T \lhd Q) := [\![S \lhd P]\!]\; T \lhd \left(\lambda(s, f).\, \boxed{\sum_{p : P\, s} Q\, (f\, p)} \right)$$

Intuition in terms of triangle diagrams:



Each position in the composite is specified by a pair

Monadic containers

Briefly: *“**containers** whose functor interpretation carries a **monad** structure”*

A monadic container[1] is a container $S \triangleleft P$ along with the data:

$$\iota : S$$

$$\sigma : \prod_{s:S} (P\, s \to S) \to S$$

$$\mathsf{pr} : \prod_{\{s:S\}} \prod_{\{f:P\, s \to S\}} P\,(\sigma\, s\, f) \to \sum_{p:P\, s} P\,(f\, p)$$

+ 8 monoid-esque equalities

Monadic containers are in *bijection* with monads on **Set** whose underlying functor is a container[1].

[1] *[Uustalu 2017]*

Monadic containers

Briefly: *"**containers** whose functor interpretation carries a **monad** structure"*

A monadic container[1] is a container $S \triangleleft P$ along with the data:

$\boxed{\iota : S}$ ← This specifies the monad unit

$$\sigma : \prod_{s:S} (P\,s \to S) \to S$$

$$\mathrm{pr} : \prod_{\{s:S\}} \prod_{\{f:P\,s \to S\}} P\,(\sigma\,s\,f) \to \sum_{p:P\,s} P\,(f\,p)$$

Monadic containers are in *bijection* with monads on **Set** whose underlying functor is a container[1].

+ 8 monoid-esque equalities

[1] *[Uustalu 2017]*

Monadic containers

Briefly: *"**containers** whose functor interpretation carries a **monad** structure"*

A monadic container[1] is a container $S \triangleleft P$ along with the data:

$\boxed{\iota : S}$ This specifies the monad unit

These specify the monad multiplication

$$\sigma : \prod_{s:S} (P\, s \to S) \to S$$

$$\mathsf{pr} : \prod_{\{s:S\}} \prod_{\{f:P\, s \to S\}} P\,(\sigma\, s\, f) \to \sum_{p:P\, s} P\,(f\, p)$$

Monadic containers are in *bijection* with monads on **Set** whose underlying functor is a container[1].

+ 8 monoid-esque equalities

[1] *[Uustalu 2017]*

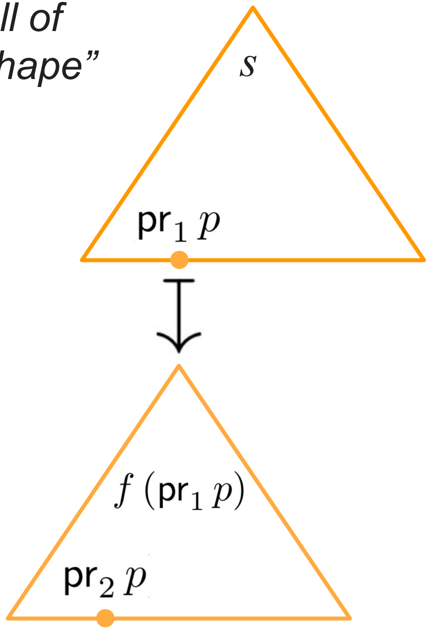
Monadic containers

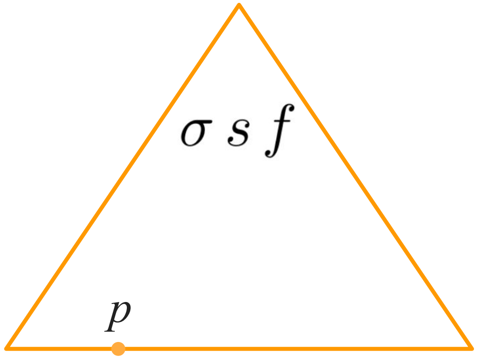
$$\iota : S$$

$$\sigma : \prod_{s:S} (P\, s \to S) \to S$$

$$\mathsf{pr} : \prod_{\{s:S\}} \prod_{\{f : P\, s \to S\}} P\,(\sigma\; s\; f) \to \sum_{p : P\, s} P\,(f\; p)$$

"A shape-full of shapes is a shape"





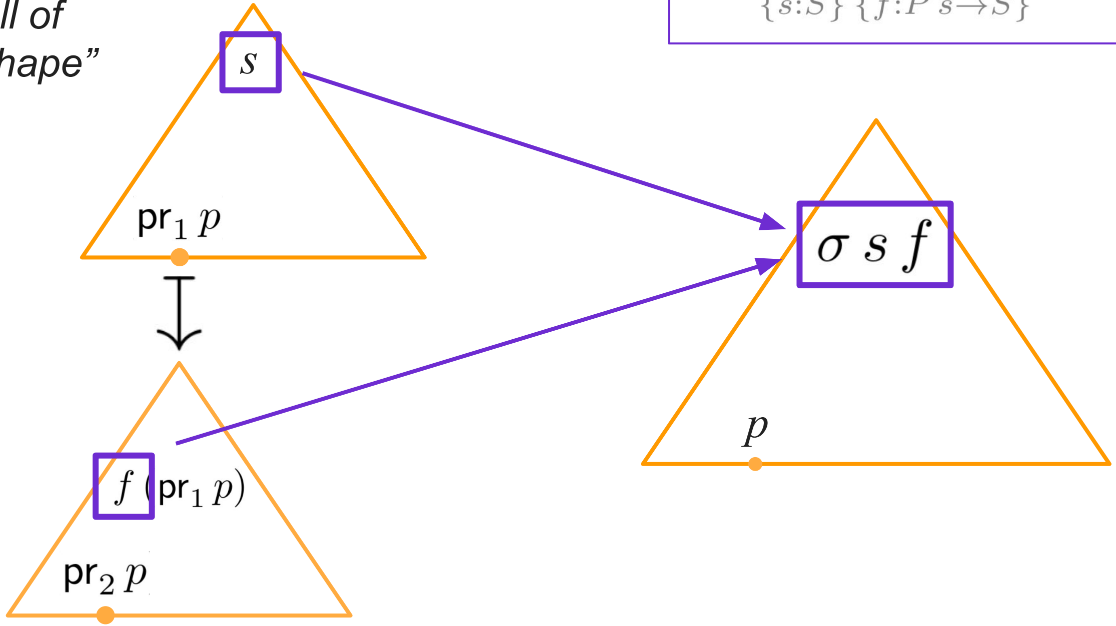
Monadic containers

$$\iota : S$$

$$\sigma : \prod_{s:S} (P\,s \to S) \to S$$

$$\mathsf{pr} : \prod_{\{s:S\}} \prod_{\{f:P\,s \to S\}} P\,(\sigma\,s\,f) \to \sum_{p:P\,s} P\,(f\,p)$$

"A shape-full of shapes is a shape"



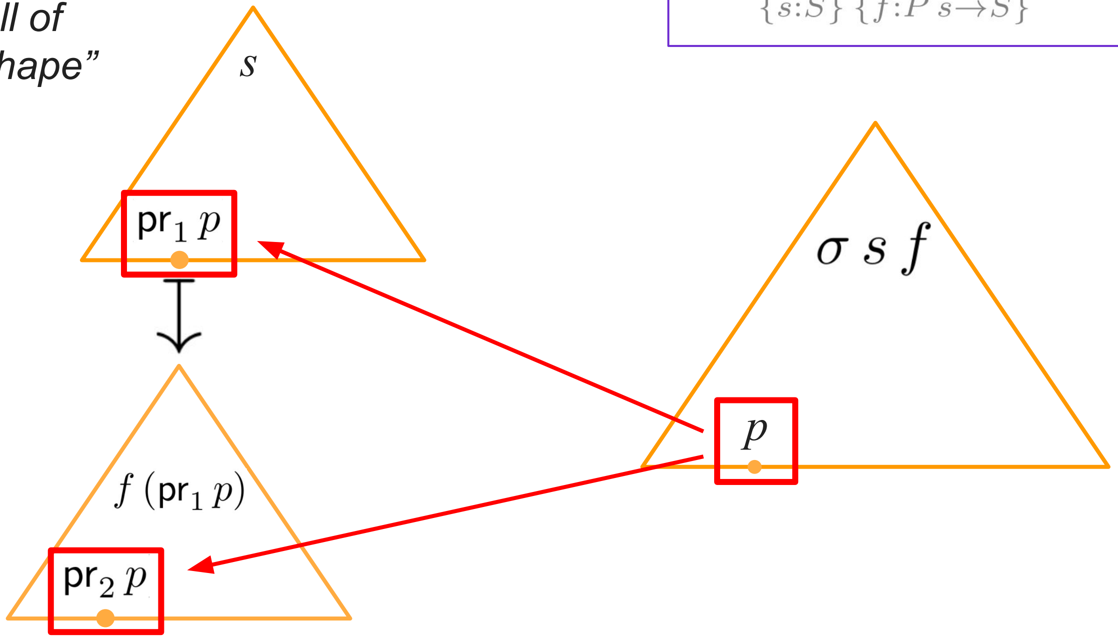
Monadic containers

$$\iota : S$$

$$\sigma : \prod_{s:S} (P\,s \to S) \to S$$

$$\mathsf{pr} : \prod_{\{s:S\}} \prod_{\{f:P\,s \to S\}} P\,(\sigma\,s\,f) \to \sum_{p:P\,s} P\,(f\,p)$$

"A shape-full of shapes is a shape"



Monadic containers

One example is the **coproduct** monadic container:

$$\iota : S$$

$$\sigma : \prod_{s:S} (P\,s \to S) \to S$$

$$\mathsf{pr} : \prod_{\{s:S\}} \prod_{\{f:P\,s \to S\}} P\,(\sigma\,s\,f) \to \sum_{p:P\,s} P\,(f\,p)$$

▶ **Example 13 (⚙).** The container $(\top + E) \triangleleft \mathrm{Tr}$ of coproducts with E, where

$$\mathrm{Tr}\,(\mathsf{inl}\,\star) := \top$$

$$\mathrm{Tr}\,(\mathsf{inr}\,\star) := \bot$$

can be extended to a monadic container by taking

$$\iota := \mathsf{inl}\,\star$$

$$\sigma\,(\mathsf{inl}\,\star)\,f := f\,\star$$

$$\sigma\,(\mathsf{inr}\,e)\,\_ := e$$

$$\mathsf{pr}\,\{\mathsf{inl}\,\star\}\,\star := (\star, \star)$$

Monadic containers

One example is the **coproduct** monadic container:

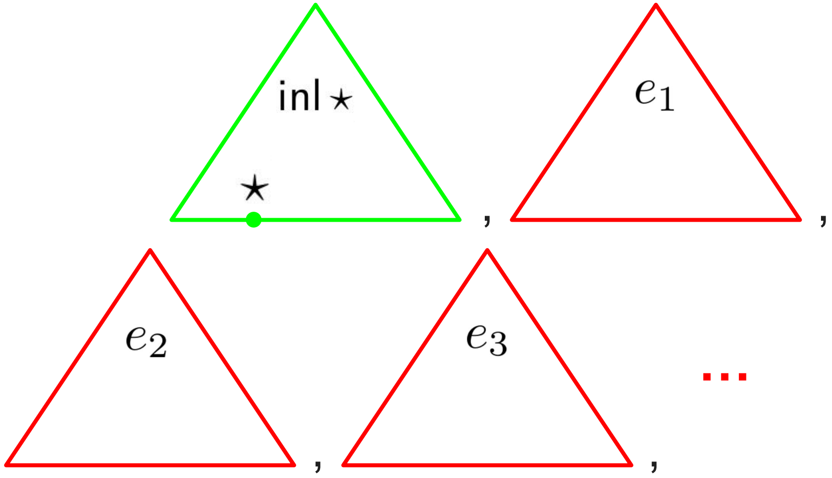
$$\iota : S$$
$$\sigma : \prod_{s:S} (P\,s \to S) \to S$$
$$\mathsf{pr} : \prod_{\{s:S\}} \prod_{\{f:P\,s \to S\}} P\,(\sigma\,s\,f) \to \sum_{p:P\,s} P\,(f\,p)$$

▶ **Example 13 (⚙).** The container $(\top + E) \triangleleft \mathrm{Tr}$ of coproducts with E, where

$$\mathrm{Tr}\,(\mathsf{inl}\,\star) := \top$$
$$\mathrm{Tr}\,(\mathsf{inr}\,\star) := \bot$$

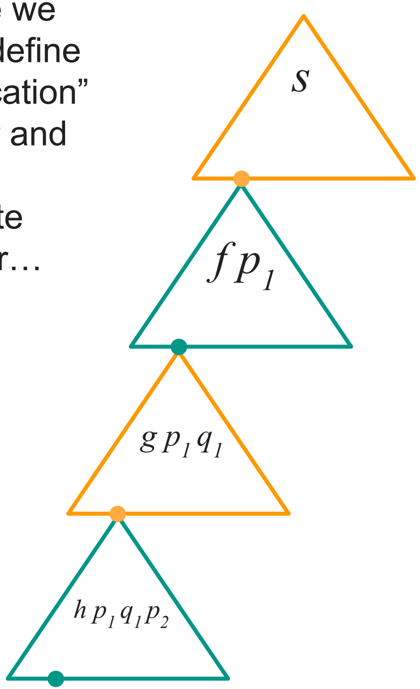
can be extended to a monadic container by taking

$$\iota := \mathsf{inl}\,\star$$
$$\sigma\,(\mathsf{inl}\,\star)\,f := f\,\star$$
$$\sigma\,(\mathsf{inr}\,e)\,\_ := e$$
$$\mathsf{pr}\,\{\mathsf{inl}\,\star\}\,\star := (\star, \star)$$



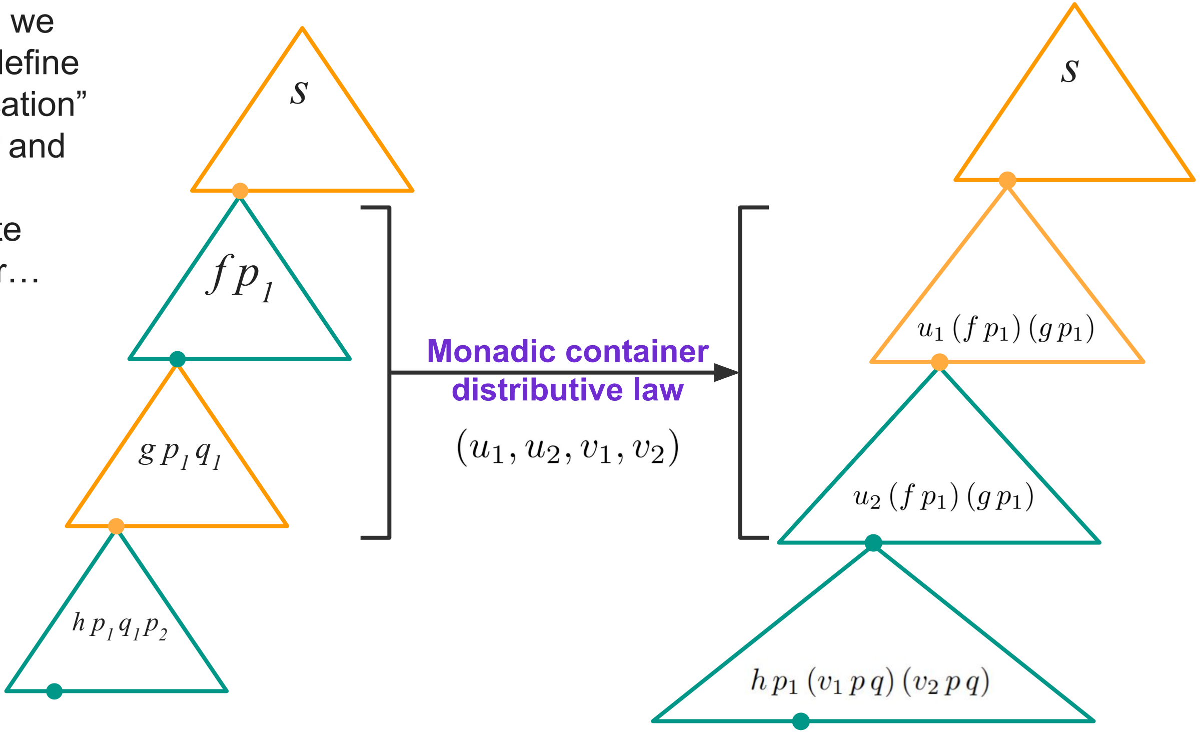
Composing monadic containers…?

Suppose we want to define “multiplication” maps (σ and `pr`) for a composite container…



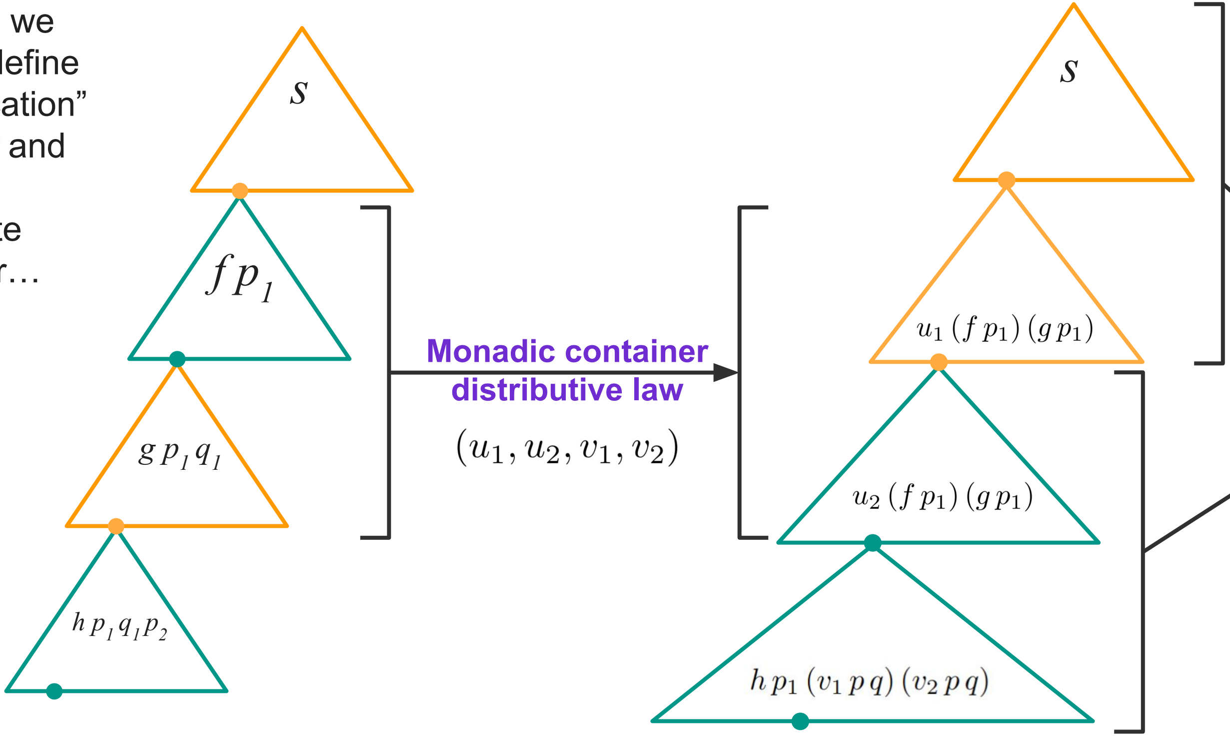
Composing monadic containers…?

Suppose we want to define “multiplication” maps (σ and `pr`) for a composite container…



Composing monadic containers…?

Suppose we want to define "multiplication" maps (σ and `pr`) for a composite container…



Can now "multiply" using the σ for each monadic container

Monadic container distributive laws

$$\boxed{\gamma : TS \Rightarrow ST}$$

$$\begin{tikzcd}
 & T \arrow[dl, "T\eta^S"'] \arrow[dr, "\eta^S T"] & \\
TS \arrow[rr, "\gamma"'] & & ST
\end{tikzcd}
\qquad
\begin{tikzcd}
 & S \arrow[dl, "\eta^T S"'] \arrow[dr, "S\eta^T"] & \\
TS \arrow[rr, "\gamma"'] & & ST
\end{tikzcd}$$

$$\begin{tikzcd}
TSS \arrow[r, "\gamma S"] \arrow[d, "T\mu^S"'] & STS \arrow[r, "S\gamma"] & SST \arrow[d, "\mu^S T"] \\
TS \arrow[rr, "\gamma"'] & & ST
\end{tikzcd}
\qquad
\begin{tikzcd}
TTS \arrow[r, "T\gamma"] \arrow[d, "\mu^T S"'] & TST \arrow[r, "\gamma T"] & STT \arrow[d, "S\mu^T"] \\
TS \arrow[rr, "\gamma"'] & & ST
\end{tikzcd}$$

[Beck 1969]

"... it can be rather difficult to prove the defining axioms of a distributive law."

[Bonsangue et al. 2015]

$$\boxed{\begin{aligned}
u_1 &: \prod_{s:S} (P\,s \to T) \to T \\
u_2 &: \prod_{s:S} \prod_{f:P\,s \to T} Q\,(u_1\,s\,f) \to S \\
v_1 &: \prod_{\{s:S\}} \prod_{\{f:P\,s \to T\}} \prod_{q:Q\,(u_1\,s\,f)} P\,(u_2\,s\,f\,q) \to P\,s \\
v_2 &: \prod_{\{s:S\}} \prod_{\{f:P\,s \to T\}} \prod_{q:Q\,(u_1\,s\,f)} \prod_{p:P\,(u_2\,s\,f\,q)} Q\,(f\,(v_1\,q\,p))
\end{aligned}}$$

$$u_1\,\iota^S\,(\lambda\_.t) = t$$
$$u_2\,\iota^S\,(\lambda\_.t) = \lambda\_.\iota^S$$
$$v_1\,\{\iota^S\}\,\{\lambda\_.t\}\,q\,p = p$$
$$v_2\,\{\iota^S\}\,\{\lambda\_.t\}\,q\,p = q$$

$$u_1\,(\sigma^S\,s\,f)\,(g \circ \mathsf{pr}^S) = u_1\,s\,(\lambda p.u_1\,(f\,p)\,(g \circ (p,-)))$$
$$u_2\,(\sigma^S\,s\,f)\,(g \circ \mathsf{pr}^S)\,q = \sigma^S\,(u_2\,s\,(\lambda p.u_1\,(f\,p)\,(g \circ (p,-)))\,q)\,(\lambda p.u_2\,(f\,(v_1\,q\,p))\,(g \circ (v_1\,q\,p,-))\,(v_2\,q\,p))$$
$$\mathsf{pr}_1^S\,(v_1\,q\,p) = v_1\,q\,(\mathsf{pr}_1^S\,p)$$
$$\mathsf{pr}_2^S\,(v_1\,q\,p) = v_1\,(v_2\,q\,(\mathsf{pr}_1^S\,p))\,(\mathsf{pr}_2^S\,p)$$
$$v_2\,q\,p = v_2\,(v_2\,q\,(\mathsf{pr}_1^S\,p))\,(\mathsf{pr}_2^S\,p)$$

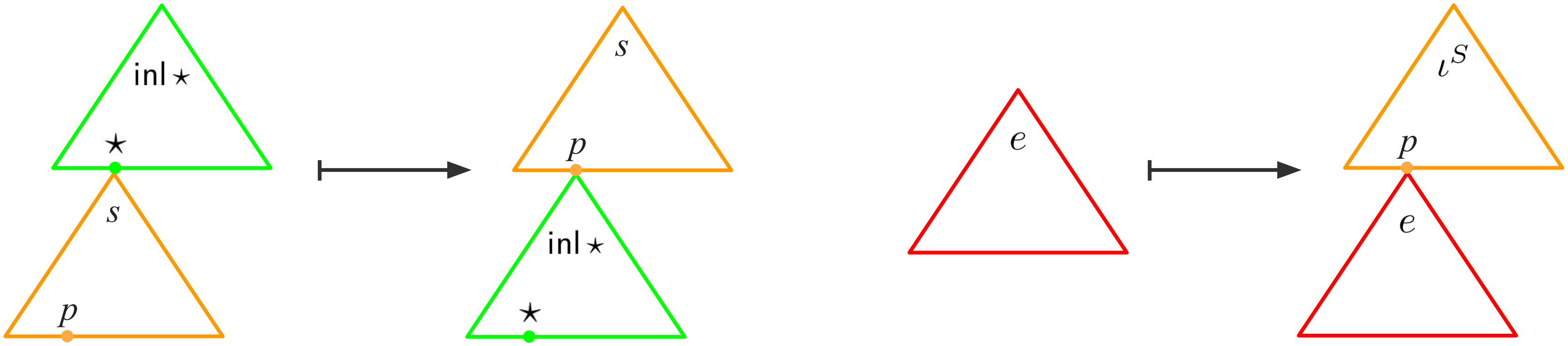
$$u_1\,s\,(\lambda\_.\iota^T) = \iota^T$$
$$u_2\,s\,(\lambda\_.\iota^T) = \lambda\_.s$$
$$v_1\,\{s\}\,\{\lambda\_.\iota^T\}\,q\,p = p$$
$$v_2\,\{s\}\,\{\lambda\_.\iota^T\}\,q\,p = q$$

$$u_1\,s\,(\lambda p.\sigma^T\,(f\,p)\,(g \circ (p,-))) = \sigma^T\,(u_1\,s\,f)\,(\lambda q.u_1\,(u_2\,s\,f\,q)\,(g \circ v\,q))$$
$$u_2\,s\,(\lambda p.\sigma^T\,(f\,p)\,(g \circ (p,-)))\,q = u_2\,(u_2\,s\,f\,(\mathsf{pr}_1^T\,q))\,(g \circ v\,q)\,(\mathsf{pr}_2^T\,q)$$
$$v_1\,(\mathsf{pr}_1^T\,q)\,(v_1\,(\mathsf{pr}_2^T\,q)\,p) = v_1\,q\,p$$
$$v_2\,(\mathsf{pr}_1^T\,q)\,(v_1\,(\mathsf{pr}_2^T\,q)\,p) = \mathsf{pr}_1^T\,(v_2\,q\,p)$$
$$v_2\,(\mathsf{pr}_2^T\,q)\,p = \mathsf{pr}_2^T\,(v_2\,q\,p)$$

Monadic container distributive laws

For example, there is a monadic container distributive law of any monadic container $(S \triangleleft P, \iota^S, \sigma^S, \mathsf{pr}^S)$ over the coproduct monadic container:

$$
\begin{aligned}
u\,(\mathsf{inl}\,\star)\,f &:= (f\,\star, \lambda\_.\mathsf{inl}\,\star)\\
u\,(\mathsf{inr}\,e)\,\_ &:= (\iota^S, \lambda\_.\mathsf{inr}\,e)\\
v\,\{\mathsf{inl}\,\star\}\,\{f\}\,p\,\star &:= (\star, p)
\end{aligned}
$$

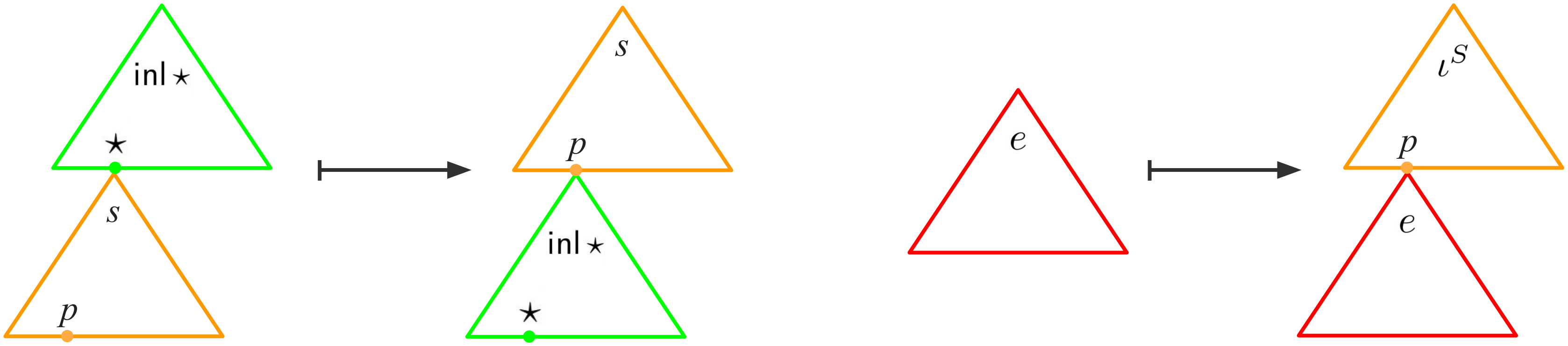


Monadic container distributive laws

For example, there is a monadic container distributive law of any monadic container $(S \triangleleft P, \iota^S, \sigma^S, \mathsf{pr}^S)$ over the coproduct monadic container:

$$u\,(\mathsf{inl}\star)\;f := (f\star, \lambda\_.\mathsf{inl}\star)$$
$$u\,(\mathsf{inr}\;e)\;\_ := (\iota^S, \lambda\_.\mathsf{inr}\;e)$$
$$v\;\{\mathsf{inl}\star\}\;\{f\}\;p\star := (\star, p)$$

Lemma: it is the **unique** one of this type!

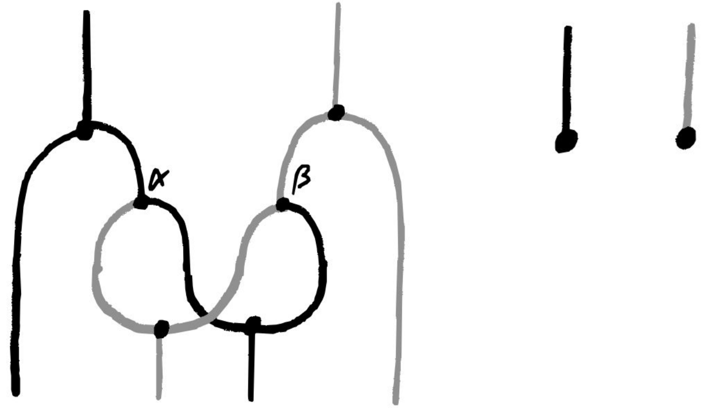


Composite monadic container

Given a distributive law, we can construct the *composite monadic container*:

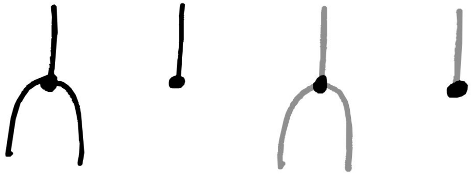
$$
\begin{aligned}
\iota &:= (\iota^S, \lambda\_.\iota^T) \\
\sigma\,(s, f)\,g &:= (\sigma^S\, s\,(\lambda p.u_1\,(f\,p)\,(g_1 \circ (p, -))), \\
&\qquad \lambda p.\sigma^T\,(u_2\,(f\,(\mathsf{pr}_1^S\,p))\,(g_1 \circ (\mathsf{pr}_1^S\,p, -))\,(\mathsf{pr}_2^S\,p)) \\
&\qquad\quad (\lambda q.g_2\,(\mathsf{pr}_1^S\,p, v_1\,(\mathsf{pr}_2^S\,p)\,q)\,(v_2\,(\mathsf{pr}_2^S\,p)\,q))) \\
\mathsf{pr}\,(p, q) &:= ((\mathsf{pr}_1^S\,p, v_1\,(\mathsf{pr}_2^S\,p)\,\,(\mathsf{pr}_1^T\,q)), \\
&\qquad (v_2\,(\mathsf{pr}_2^S\,p)\,(\mathsf{pr}_1^T\,q), \mathsf{pr}_2^T\,q))
\end{aligned}
$$

This is analogous to the composite monoid (Zappa–Szép product) obtained from a matching pair:

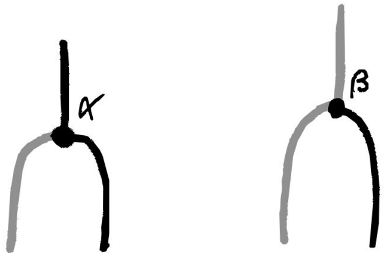


Matching pairs

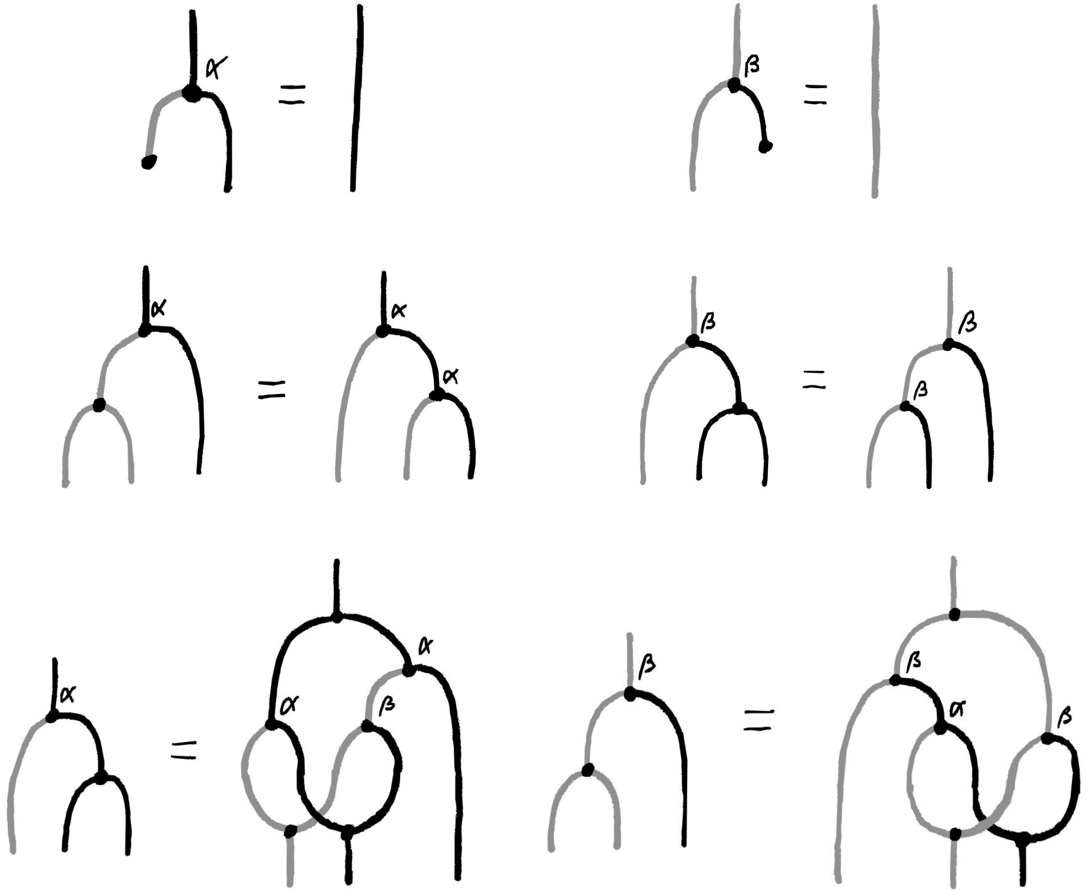
Given two monoids:



A *matching pair* is a pair of monoid actions:



such that the following equalities hold:



Cartesian monadic containers and type universes

We can see *cartesian* monadic containers as Tarski-style type universes[1] closed under singleton and dependent sum types.

$s : \mathcal{U}$ is a code

$P\ s$ is the type (set) coded for by s

un : $P\ \iota \cong \top$

pr : $\prod_{\{s:S\}} \prod_{\{f:P\ s \to S\}} P\,(\sigma\ s\ f) \cong \sum_{p:P\ s} P\,(f\ p)$

Under this lens, the distributive law on slide 9 becomes a uniform way to extend a type universe with *refinement types*.

$(s, f) : \sum_{s:\mathcal{U}} P\ s \to 2$ is a code

$\Diamond^P_{\mathrm{Tr}}(s, f)$ is the type coded for by (s, f)

$$\Diamond^P_{\mathrm{Tr}}(s, f) := \sum_{x:P\ s} \mathrm{Tr}\,(f\ x)$$

[1] *[Altenkirch and Pinyo 2017]*

Cartesian monadic containers and type universes

We can see *cartesian* monadic containers as Tarski-style type universes[1] closed under singleton and dependent sum types.

$s : \mathcal{U}$ is a code

$P\,s$ is the type (set) coded for by s

un : $P\,\iota \cong \top$

pr : $\prod_{\{s:S\}} \prod_{\{f:P\,s \to S\}} P\,(\sigma\,s\,f) \cong \sum_{p:P\,s} P\,(f\,p)$

Under this lens, the distributive law on slide 9 becomes a uniform way to extend a type universe with *refinement types*.

$(s, f) : \sum_{s:\mathcal{U}} P\,s \to 2$ is a code

$\Diamond^P_{\mathrm{Tr}}(s, f)$ is the type coded for by (s, f)

$$\Diamond^P_{\mathrm{Tr}}(s, f) := \boxed{\sum_{x:P\,s} \mathrm{Tr}\,(f\,x)}$$

Elements of $P\,s$ for which the predicate holds

[1] *[Altenkirch and Pinyo 2017]*

Cartesian monadic containers and type universes

Under this lens, the distributive law on slide 9 becomes a uniform way to extend a type universe with *refinement types*.

$$(s, f) : \sum_{s:\mathcal{U}} P\, s \to 2$$ is a code

$$\Diamond^{P}_{\mathrm{Tr}}(s, f) := \boxed{\sum_{x:P\, s} \mathrm{Tr}\,(f\, x)}$$

Elements of $P\, s$ for which the predicate holds

$\Diamond^{P}_{\mathrm{Tr}}(s, f)$ is the type coded for by (s, f)

The codes for singleton and dependent sum types in the "refinement type" universe:

$$\iota := (\iota^{\mathcal{U}}, \lambda\_.\mathrm{true})$$

$$\sigma\,(s, f)\, g := \left(\sigma^{\mathcal{U}}\, s \left(\lambda p. \begin{cases} g_1\,(p, \star) & \text{if } f\, p = \mathrm{true} \\ \iota^{\mathcal{U}} & \text{otherwise} \end{cases}\right), \lambda p. \begin{cases} g_2\,(\mathsf{pr}^{\mathcal{U}}_1\, p, \star)\,(\mathsf{pr}^{\mathcal{U}}_2\, p) & \text{if } f\,(\mathsf{pr}^{\mathcal{U}}_1\, p) = \mathrm{true} \\ \mathrm{false} & \text{otherwise} \end{cases}\right)$$

Cartesian monadic containers and type universes

We can see *cartesian* monadic containers as Tarski-style type universes closed under singleton and dependent sum types.

$s : \mathcal{U}$ is a code

$P\, s$ is the type (set) coded for by s

un : $P\, \iota \cong \top$

pr : $\prod_{\{s:S\}} \prod_{\{f:P\, s \to S\}} P\,(\sigma\, s\, f) \cong \sum_{p:P\, s} P\,(f\, p)$

Further work: adapt monadic container distributive laws to distributive laws between type universes with singleton, dependent sum **and dependent product** types.

$$\pi : \prod_{s:S} (P\, s \to S) \to S$$

$$\mathsf{app} : \prod_{\{s:S\}} \prod_{\{f:P\, s \to S\}} P\,(\pi\, s\, f) \cong \prod_{p:P\, s} P\,(f\, p)$$

$$\pi\,(\sigma\, s\, f)\,(g \circ \mathsf{pr}) = \pi\, s\,(\lambda x.\pi\,(f\, x)\,(g\, x))$$

Container characterisation landscape

Monadic containers
(containers with monad structure)
[Uustalu 2017]

Directed containers
(containers with comonad structure)
[Ahman et al. 2012]

Directed container distributive laws
(characterisation of comonad distributive laws in terms of directed containers)
[Ahman and Uustalu 2013]

Container characterisation landscape

Monadic containers
(containers with monad structure)
[Uustalu 2017]

Directed containers
(containers with comonad structure)
[Ahman et al. 2012]

Monadic container distributive laws
(characterisation of monad distributive laws in terms of monadic containers)

Directed container distributive laws
(characterisation of comonad distributive laws in terms of directed containers)
[Ahman and Uustalu 2013]

Container characterisation landscape

Monadic containers
(containers with monad structure)
[Uustalu 2017]

Directed containers
(containers with comonad structure)
[Ahman et al. 2012]

Monadic container distributive laws
(characterisation of monad distributive laws in terms of monadic containers)

Directed container distributive laws
(characterisation of comonad distributive laws in terms of directed containers)
[Ahman and Uustalu 2013]

Mixed (monadic-directed, directed-monadic) **container distributive laws**

Corresponding monoid structures

		$A \times -$	$B \to -$
		Writer monadic container	Reader directed container
$A \times -$	Writer monadic container	Matching pairs	Functional monoid actions
$B \to -$	Reader directed container	Matching pairs	Matching pairs

Corresponding monoid structures

		$A \times -$	$B \to -$
		Writer monadic container	Reader directed container
$A \times -$	Writer monadic container	Matching pairs	$\boxed{\text{Functional monoid actions}}$
$B \to -$	Reader directed container	Matching pairs	Matching pairs

A "functional monoid action" is a map $\alpha : (A \to B) \to A \to A$ satisfying:

$$\alpha\, f\, e^A = e^A$$

$$\alpha\, f\, (a \oplus^A a') = \alpha\, f\, a \oplus^A \alpha\, (\lambda x. f\, (\alpha\, f\, a \oplus^A x))\, a'$$

$$\alpha\, (\lambda\_.e^B)\, a = a$$

$$\alpha\, (\lambda x. f\, x \oplus^B g\, x)\, a = \alpha\, f\, (\alpha\, (\lambda x. g\, (\alpha\, f\, x))\, a)$$

Cubical Agda formalisation

This characterisation is good for formalising distributive laws in Cubical Agda.

```
MaybeDistr : ∀ {ℓs ℓp} (S : Set ℓs) (P : S → Set ℓp) (MC : MndContainer ℓs ℓp (S ▷ P)) →
           MndDistributiveLaw ℓs ℓp 𝟚 JustOrNothing S P MaybeM MC
u₁ (MaybeDistr S P MC) true f = f tt
u₁ (MaybeDistr S P MC) false f = MC .ι
u₂ (MaybeDistr S P MC) true f _ = true
u₂ (MaybeDistr S P MC) false f _ = false
v₁ (MaybeDistr S P MC) {true} _ x = tt
v₂ (MaybeDistr S P MC) {true} {f} p x = p
unit-ιB-shape₁ (MaybeDistr S P MC) true = refl
unit-ιB-shape₁ (MaybeDistr S P MC) false = refl
unit-ιB-shape₂ (MaybeDistr S P MC) true = refl
unit-ιB-shape₂ (MaybeDistr S P MC) false = refl
unit-ιB-pos₁ (MaybeDistr S P MC) true i q tt = tt
unit-ιB-pos₂ (MaybeDistr S P MC) true i q tt = q
unit-ιA-shape₁ (MaybeDistr S P MC) _ = refl
unit-ιA-shape₂ (MaybeDistr S P MC) _ = refl
unit-ιA-pos₁ (MaybeDistr S P MC) s i q tt = tt
unit-ιA-pos₂ (MaybeDistr S P MC) s i q tt = q
mul-A-shape₁ (MaybeDistr S P MC) true f g = refl
mul-A-shape₁ (MaybeDistr S P MC) false f g = refl
mul-A-shape₂ (MaybeDistr S P MC) true f g = refl
mul-A-shape₂ (MaybeDistr S P MC) false f g = refl
mul-A-pos₁ (MaybeDistr S P MC) true f g = refl
mul-A-pos₁ (MaybeDistr {ℓs} {ℓp} S P MC) false f g i q ()
mul-A-pos₂₁ (MaybeDistr S P MC) true f g = refl
mul-A-pos₂₁ (MaybeDistr {ℓs} {ℓp} S P MC) false f g i q ()
mul-A-pos₂₂ (MaybeDistr S P MC) true f g = refl
mul-A-pos₂₂ (MaybeDistr S P MC) false f g i q ()
mul-B-shape₁ (MaybeDistr S P MC) true f g = refl
mul-B-shape₁ (MaybeDistr S P MC) false f g i = unit-r (isMndContainer MC) (MC .ι) (~ i)
mul-B-shape₂ (MaybeDistr S P MC) true f g = refl
mul-B-shape₂ (MaybeDistr S P MC) false f g i = λ _ → false
mul-B-pos₁ (MaybeDistr S P MC) true f g i q tt = tt
mul-B-pos₁ (MaybeDistr S P MC) false f g i q ()
mul-B-pos₂₁ (MaybeDistr S P MC) true f g i q tt = (MC .pr₁) (f tt) (g tt) q
mul-B-pos₂₁ (MaybeDistr S P MC) false f g i q ()
mul-B-pos₂₂ (MaybeDistr S P MC) true f g i q tt = (MC .pr₂) (f tt) (g tt) q
mul-B-pos₂₂ (MaybeDistr S P MC) false f g i q ()
```

```
module WriterCReaderMDistrUnique {ℓs ℓp : Level} (A : Set ℓs) (B : Set ℓp)
  (L : MndDirectedDistributiveLaw ℓs ℓp A (const (⊤ {ℓp})) (⊤ {ℓs}) (const B) (WriterC A) (ReaderM B)) where

  L₀ = WriterCReaderMDistr A B

  lem⊤ : (a : A) (f : ⊤ {ℓp} → ⊤ {ℓs}) → f ≡ const tt
  lem⊤ a f i p = ⊤-singleton (f p) i

  u1 : (a : A) (f : ⊤ {ℓp} → ⊤ {ℓs}) → u₁ L₀ a f ≡ u₁ L a f
  u1 a f i = ⊤-singleton (u₁ L a f) (~ i)

  u2 : (a : A) (f : ⊤ {ℓp} → ⊤ {ℓs}) (b : B) → u₂ L₀ a f b ≡ u₂ L a f b
  u2 a f b i = hcomp (λ j → λ { (i = i0) → a
                            ; (i = i1) → u₂ L a (lem⊤ a f (~ j)) b }) (unit-ιB-shape₂ L a (~ i) b)

  v1 : (a : A) (f : ⊤ {ℓp} → ⊤ {ℓs}) (b : B) (x : ⊤ {ℓp}) → v₁ L₀ {a} {f} b x ≡ v₁ L {a} {f} b x
  v1 a f b tt i = hcomp (λ j → λ { (i = i0) → tt
                              ; (i = i1) → v₁ L {a} {lem⊤ a f (~ j)} b tt }) (unit-ιB-pos₁ L a (~ i) b tt)

  v2 : (a : A) (f : ⊤ {ℓp} → ⊤ {ℓs}) (b : B) (x : ⊤ {ℓp}) → v₂ L₀ {a} {f} b x ≡ v₂ L {a} {f} b x
  v2 a f b tt i = hcomp (λ j → λ { (i = i0) → b
                              ; (i = i1) → v₂ L {a} {lem⊤ a f (~ j)} b tt }) (unit-ιB-pos₂ L a (~ i) b tt)
```

Proof that the reader directed container over writer monadic container mixed distributive law is unique.

Cubical Agda formalisation

This characterisation is good for formalising distributive laws in Cubical Agda.

```agda
MaybeDistr : ∀ {ℓs ℓp} (S : Set ℓs) (P : S → Set ℓp) (MC : MndContainer ℓs ℓp (S ▷ P)) →
             MndDistributiveLaw ℓs ℓp 𝟚 JustOrNothing S P MaybeM MC
u₁ (MaybeDistr S P MC) true f = f tt
u₁ (MaybeDistr S P MC) false f = MC .ι
u₂ (MaybeDistr S P MC) true f _ = true
u₂ (MaybeDistr S P MC) false f _ = false
v₁ (MaybeDistr S P MC) {true} _ x = tt
v₂ (MaybeDistr S P MC) {true} {f} p x = p
unit-ιB-shape₁ (MaybeDistr S P MC) true = refl
unit-ιB-shape₁ (MaybeDistr S P MC) false = refl
unit-ιB-shape₂ (MaybeDistr S P MC) true = refl
unit-ιB-shape₂ (MaybeDistr S P MC) false = refl
unit-ιB-pos₁ (MaybeDistr S P MC) true i q tt = tt
unit-ιB-pos₂ (MaybeDistr S P MC) true i q tt = q
unit-ιA-shape₁ (MaybeDistr S P MC) _ = refl
unit-ιA-shape₂ (MaybeDistr S P MC) _ = refl
unit-ιA-pos₁ (MaybeDistr S P MC) s i q tt = tt
unit-ιA-pos₂ (MaybeDistr S P MC) s i q tt = q
mul-A-shape₁ (MaybeDistr S P MC) true f g = refl
mul-A-shape₁ (MaybeDistr S P MC) false f g = refl
mul-A-shape₂ (MaybeDistr S P MC) true f g = refl
mul-A-shape₂ (MaybeDistr S P MC) false f g = refl
mul-A-pos₁ (MaybeDistr S P MC) true f g = refl
mul-A-pos₁ (MaybeDistr {ℓs} {ℓp} S P MC) false f g i q ()
mul-A-pos₂₁ (MaybeDistr S P MC) true f g = refl
mul-A-pos₂₁ (MaybeDistr {ℓs} {ℓp} S P MC) false f g i q ()
mul-A-pos₂₂ (MaybeDistr S P MC) true f g = refl
mul-A-pos₂₂ (MaybeDistr S P MC) false f g i q ()
mul-B-shape₁ (MaybeDistr S P MC) true f g = refl
mul-B-shape₁ (MaybeDistr S P MC) false f g i = unit-r (isMndContainer MC) (MC .ι) (~ i)
mul-B-shape₂ (MaybeDistr S P MC) true f g = refl
mul-B-shape₂ (MaybeDistr S P MC) false f g i = λ _ → false
mul-B-pos₁ (MaybeDistr S P MC) true f g i q tt = tt
mul-B-pos₁ (MaybeDistr S P MC) false f g i q ()
mul-B-pos₂₁ (MaybeDistr S P MC) true f g i q tt = (MC .pr₁) (f tt) (g tt) q
mul-B-pos₂₁ (MaybeDistr S P MC) false f g i q ()
mul-B-pos₂₂ (MaybeDistr S P MC) true f g i q tt = (MC .pr₂) (f tt) (g tt) q
mul-B-pos₂₂ (MaybeDistr S P MC) false f g i q ()
```

All other equalities hold trivially!

```agda
module WriterCReaderMDistrUnique {ℓs ℓp : Level} (A : Set ℓs) (B : Set ℓp)
  (L : MndDirectedDistributiveLaw ℓs ℓp A (const (⊤ {ℓp})) (⊤ {ℓs}) (const B) (WriterC A) (ReaderM B)) where

  L₀ = WriterCReaderMDistr A B

  lem⊤ : (a : A) (f : ⊤ {ℓp} → ⊤ {ℓs}) → f ≡ const tt
  lem⊤ a f i p = ⊤-singleton (f p) i

  u1 : (a : A) (f : ⊤ {ℓp} → ⊤ {ℓs}) → u₁ L₀ a f ≡ u₁ L a f
  u1 a f i = ⊤-singleton (u₁ L a f) (~ i)

  u2 : (a : A) (f : ⊤ {ℓp} → ⊤ {ℓs}) (b : B) → u₂ L₀ a f b ≡ u₂ L a f b
  u2 a f b i = hcomp (λ j → λ { (i = i0) → a
                              ; (i = i1) → u₂ L a (lem⊤ a f (~ j)) b }) (unit-ιB-shape₂ L a (~ i) b)

  v1 : (a : A) (f : ⊤ {ℓp} → ⊤ {ℓs}) (b : B) (x : ⊤ {ℓp}) → v₁ L₀ {a} {f} b x ≡ v₁ L {a} {f} b x
  v1 a f b tt i = hcomp (λ j → λ { (i = i0) → tt
                                 ; (i = i1) → v₁ L {a} {lem⊤ a f (~ j)} b tt }) (unit-ιB-pos₁ L a (~ i) b tt)

  v2 : (a : A) (f : ⊤ {ℓp} → ⊤ {ℓs}) (b : B) (x : ⊤ {ℓp}) → v₂ L₀ {a} {f} b x ≡ v₂ L {a} {f} b x
  v2 a f b tt i = hcomp (λ j → λ { (i = i0) → b
                                 ; (i = i1) → v₂ L {a} {lem⊤ a f (~ j)} b tt }) (unit-ιB-pos₂ L a (~ i) b tt)
```

Proof that the reader directed container over writer monadic container mixed distributive law is unique.

Link to our paper
www.arxiv.org/abs/2503.17191

Summary

Our contributions:

- Characterisation of monadic container distributive laws
- Characterisation of mixed container distributive laws (monadic-directed and directed-monadic)
- Uniqueness proofs for various (simple) monadic and mixed container distributive laws
- A no-go theorem for monadic container distributive laws *[Zwart and Marsden 2018]*
- Formalisation in Cubical Agda of the characterisation and proofs of uniqueness

Future work:

- Extend distributive laws between cartesian monadic containers (type universes) to those with codes for dependent products
- Explore further no-go theorems
- Extend all characterisations to groupoid and categorical containers *[Gylterud 2011]*

Thank you!