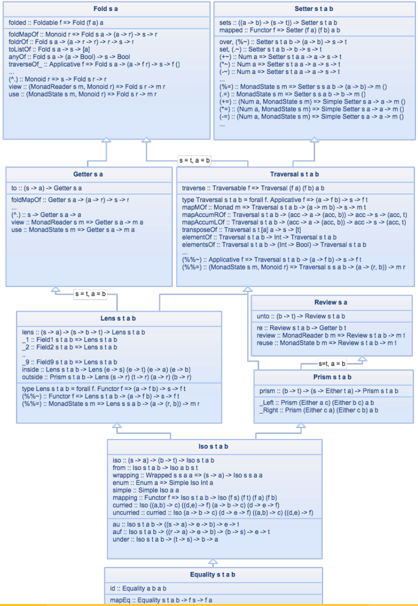
Optics and Type Equivalences

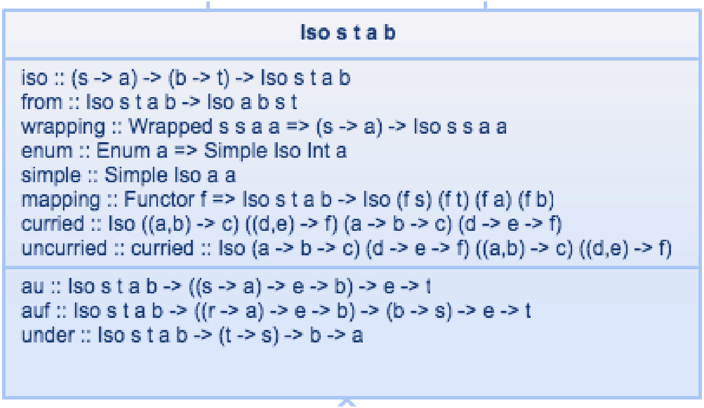
Jacques Carette, Amr Sabry with some help from Chao-Hong Chen

McMaster and Indiana

Optics


Fold s a
folded :: Foldable f => Fold (f a) a
foldMapOf :: Monoid r => Fold s a -> (a -> r) -> s -> r
foldrOf :: Fold s a -> (a -> r -> r) -> r -> s -> r
toListOf :: Fold s a -> s -> [a]
anyOf :: Fold s a -> (a -> Bool) -> s -> Bool
traverseOf_ :: Applicative f => Fold s a -> (a -> f r) -> s -> f ()
...
(^.) :: Monoid r => s -> Fold s r -> r
view :: (MonadReader s m, Monoid r) => Fold s r -> m r
use :: (MonadState s m, Monoid r) => Fold s r -> m r
Setter s t a b
sets :: ((a -> b) -> (s -> t)) -> Setter s t a b
mapped :: Functor f => Setter (f a) (f b) a b
over, (%~) :: Setter s t a b -> (a -> b) -> s -> t
set, (.~) :: Setter s t a b -> b -> s -> t
(+~) :: Num a => Setter s t a a -> a -> s -> t
(*~) :: Num a => Setter s t a a -> a -> s -> t
(-~) :: Num a => Setter s t a a -> a -> s -> t
...
(%=) :: MonadState s m => Setter s s a b -> (a -> b) -> m ()
(.=) :: MonadState s m => Setter s s a b -> b -> m ()
(+=) :: (Num a, MonadState s m) => Simple Setter s a -> a -> m ()
(*=) :: (Num a, MonadState s m) => Simple Setter s a -> a -> m ()
(-=) :: (Num a, MonadState s m) => Simple Setter s a -> a -> m ()
...
s = t, a = b
Getter s a
to :: (s -> a) -> Getter s a
foldMapOf :: Getter s a -> (a -> r) -> s -> r
...
(^.) :: s -> Getter s a -> a
view :: MonadReader s m => Getter s a -> m a
use :: MonadState s m => Getter s a -> m a
Traversal s t a b
traverse :: Traversable f => Traversal (f a) (f b) a b
type Traversal s t a b = forall f. Applicative f => (a -> f b) -> s -> f t
mapMOf :: Monad m => Traversal s t a b -> (a -> m b) -> s -> m t
mapAccumROf :: Traversal s t a b -> (acc -> a -> (acc, b)) -> acc -> s -> (acc, t)
mapAccumLOf :: Traversal s t a b -> (acc -> a -> (acc, b)) -> acc -> s -> (acc, t)
transposeOf :: Traversal s t [a] a -> s -> [t]
elementOf :: Traversal s t a b -> Int -> Traversal s t a b
elementsOf :: Traversal s t a b -> (Int -> Bool) -> Traversal s t a b
...
(%%~) :: Applicative f => Traversal s t a b -> (a -> f b) -> s -> f t
(%%=) :: (MonadState s m, Monoid r) => Traversal s s a b -> (a -> (r, b)) -> m r
s = t, a = b
Review s a
unto :: (b -> t) -> Review s t a b
re :: Review s t a b -> Getter b t
review :: MonadReader b m => Review s t a b -> m t
reuse :: MonadState b m => Review s t a b -> m t
Lens s t a b
lens :: (s -> a) -> (s -> b -> t) -> Lens s t a b
_1 :: Field1 s t a b => Lens s t a b
_2 :: Field2 s t a b => Lens s t a b
..
_9 :: Field9 s t a b => Lens s t a b
inside :: Lens s t a b -> Lens (e -> s) (e -> t) (e -> a) (e -> b)
outside :: Prism s t a b -> Lens (s -> r) (t -> r) (a -> r) (b -> r)
type Lens s t a b = forall f. Functor f => (a -> f b) -> s -> f t
(%%~) :: Functor f => Lens s t a b -> (a -> f b) -> s -> f t
(%%=) :: MonadState s m => Lens s s a b -> (a -> (r, b)) -> m r
s=t, a = b
Prism s t a b
prism :: (b -> t) -> (s -> Either t a) -> Prism s t a b
_Left :: Prism (Either a c) (Either b c) a b
_Right :: Prism (Either c a) (Either c b) a b
Iso s t a b
iso :: (s -> a) -> (b -> t) -> Iso s t a b
from :: Iso s t a b -> Iso a b s t
wrapping :: Wrapped s s a a => (s -> a) -> Iso s s a a
enum :: Enum a => Simple Iso Int a
simple :: Simple Iso a a
mapping :: Functor f => Iso s t a b -> Iso (f s) (f t) (f a) (f b)
curried :: Iso ((a,b) -> c) ((d,e) -> f) (a -> b -> c) (d -> e -> f)
uncurried :: curried :: Iso (a -> b -> c) (d -> e -> f) ((a,b) -> c) ((d,e) -> f)
au :: Iso s t a b -> ((s -> a) -> e -> b) -> e -> t
auf :: Iso s t a b -> ((r -> a) -> e -> b) -> (b -> s) -> e -> t
under :: Iso s t a b -> (t -> s) -> b -> a
Equality s t a b
id :: Equality a b a b
mapEq :: Equality s t a b -> f s -> f a

Based on a ‘reversible’ core:



Lens in Haskell

```haskell
data Lens s a = Lens { view :: s -> a, set :: s -> a -> s }
```

Lens in Haskell

```haskell
data Lens s a = Lens { view :: s -> a, set :: s -> a -> s }
```

Example:

```haskell
_1 :: Lens (a , b) a
_1 = Lens { view = fst , set = \s a -> (a, snd s) }
```

Lens in Haskell

```haskell
data Lens s a = Lens { view :: s -> a, set :: s -> a -> s }
```

Example:

```haskell
_1 :: Lens (a , b) a
_1 = Lens { view = fst , set = \s a -> (a, snd s) }
```

Laws? Optimizations?

```haskell
view (set s a) == a
set s (view s) == s
set (set s a) a' == set s a'
```

Lens in Agda

```
record GS-Lens {ℓs ℓa : Level} (S : Set ℓs) (A : Set ℓa) : Set (ℓs ⊔ ℓa) where
  field
    get    : S → A
    set    : S → A → S
    getput : {s : S} {a : A} → get (set s a) ≡ a
    putget : (s : S)         → set s (get s) ≡ s
    putput : (s : S) (a a' : A) → set (set s a) a' ≡ set s a'
open GS-Lens
```

Works... but the proofs can be tedious in larger examples (later)

Lens in Agda

```
record GS-Lens {ℓs ℓa : Level} (S : Set ℓs) (A : Set ℓa) : Set (ℓs ⊔ ℓa) where
  field
    get    : S → A
    set    : S → A → S
    getput : {s : S} {a : A} → get (set s a) ≡ a
    putget : (s : S)         → set s (get s) ≡ s
    putput : (s : S) (a a' : A) → set (set s a) a' ≡ set s a'
open GS-Lens
```

Works... but the proofs can be tedious in larger examples (later)

```
fst : {A B : Set} → GS-Lens (A × B) A
fst = record { get = proj₁
             ; set = λ s a → (a , proj₂ s)
             ; getput = refl
             ; putget = λ _ → refl
             ; putput = λ _ _ _ → refl }
```

Lens in Agda 2

Or, the return of constant-complement lenses:

```
record Lens₁ {ℓ : Level} (S : Set ℓ) (A : Set ℓ) : Set (suc ℓ) where
  constructor ∃-lens
  field
    {C} : Set ℓ
    iso : S ≃ (C × A)
```

Lens in Agda 2

Or, the return of constant-complement lenses:

```
record Lens₁ {ℓ : Level} (S : Set ℓ) (A : Set ℓ) : Set (suc ℓ) where
  constructor ∃-lens
  field
    {C} : Set ℓ
    iso : S ≃ (C × A)

fst : {A B : Set} → Lens₁ (A × B) A
fst = ∃-lens swap⋆equiv
```

Lens in Agda 2

```
  where
record isqinv {ℓ ℓ'} {A : Set ℓ} {B : Set ℓ'} (f : A → B) :
  Set (ℓ ⊔ ℓ') where
  constructor qinv
  field
    g : B → A
    α : (f ∘ g) ∼ id
    β : (g ∘ f) ∼ id

_≃_ : ∀ {ℓ ℓ'} → Set ℓ → Set ℓ' → Set (ℓ ⊔ ℓ')
A ≃ B = Σ (A → B) isqinv
```

Lens in Agda 2

Or, the return of constant-complement lenses:

```
record Lens₁ {ℓ : Level} (S : Set ℓ) (A : Set ℓ) : Set (suc ℓ) where
  constructor ∃-lens
  field
    {C} : Set ℓ
    iso : S ≃ (C × A)
```

Lens in Agda 2

Or, the return of constant-complement lenses:

```
record Lens₁ {ℓ : Level} (S : Set ℓ) (A : Set ℓ) : Set (suc ℓ) where
  constructor ∃-lens
  field
    {C} : Set ℓ
    iso : S ≃ (C × A)
```

```
sound : {ℓ : Level} {S A : Set ℓ} → Lens₁ S A → GS-Lens S A
```

Lens in Agda 2

Or, the return of constant-complement lenses:

```
record Lens₁ {ℓ : Level} (S : Set ℓ) (A : Set ℓ) : Set (suc ℓ) where
  constructor ∃-lens
  field
    {C} : Set ℓ
    iso : S ≃ (C × A)
```

```
sound : {ℓ : Level} {S A : Set ℓ} → Lens₁ S A → GS-Lens S A
```

complete requires moving to Setoid – see online code.

```
_≈_under_ : ∀ {ℓ} {S A : Set ℓ} → (s t : S) (l : GS-Lens S A) → Set ℓ
_≈_under_ s t l = ∀ a → set l s a ≡ set l t a
```

Exploiting type equivalences

```
module _ {A B D : Set} where
  l₁ : Lens₁ A A
  l₁ = ∃-lens uniti⋆equiv
  l₂ : Lens₁ (B × A) A
  l₂ = ∃-lens id≃
  l₃ : Lens₁ (B × A) B
  l₃ = ∃-lens swap⋆equiv
  l₄ : Lens₁ (D × (B × A)) A
  l₄ = ∃-lens assocl⋆equiv
  l₅ : Lens₁ ⊥ A
  l₅ = ∃-lens factorzequiv
  l₆ : Lens₁ ((D × A) ⊎ (B × A)) A
  l₆ = ∃-lens factorequiv
```

uniti⋆equiv : $A \simeq (\top \times A)$

id≃ : $A \simeq A$

swap⋆equiv : $A \times B \simeq B \times A$

assocl⋆equiv : $(A \times B) \times C \simeq A \times (B \times C)$

factorzequiv : $\bot \simeq (\bot \times A)$

factorequiv : $((A \times D) \uplus (B \times D)) \simeq ((A \uplus B) \times D)$

Proof relevance

Different proofs that $A \times A \simeq A \times A$ give different lenses:

$\mathsf{l}_7 : \mathsf{Lens}_1\ (A \times A)\ A$
$\mathsf{l}_7 = \exists\text{-lens}\ \mathsf{id}{\simeq}$
$\mathsf{l}_8 : \mathsf{Lens}_1\ (A \times A)\ A$
$\mathsf{l}_8 = \exists\text{-lens}\ \mathsf{swap}{\star}\mathsf{equiv}$

Plain Curry-Howard: $A \wedge A \equiv A$ implies $A \times A \longleftrightarrow A$ (equi-inhabitation).

Type Equivalences

Semiring

$$
\begin{aligned}
a &= a \\
\\
0 + a &= a \\
a + b &= b + a \\
a + (b + c) &= (a + b) + c \\
\\
1 \cdot a &= a \\
a \cdot b &= b \cdot a \\
a \cdot (b \cdot c) &= (a \cdot b) \cdot c \\
\\
0 \cdot a &= 0 \\
(a + b) \cdot c &= (a \cdot c) + (b \cdot c)
\end{aligned}
$$

Type Equivalences

Semiring	Types
$a = a$	$A \simeq A$
$0 + a = a$	$\bot \uplus A \simeq A$
$a + b = b + a$	$A \uplus B \simeq B \uplus A$
$a + (b + c) = (a + b) + c$	$A \uplus (B \uplus C) \simeq (A \uplus B) \uplus C$
$1 \cdot a = a$	$\top * A \simeq A$
$a \cdot b = b \cdot a$	$A * B \simeq B * A$
$a \cdot (b \cdot c) = (a \cdot b) \cdot c$	$A * (B * C) \simeq (A * B) * C$
$0 \cdot a = 0$	$\bot * A \simeq \bot$
$(a + b) \cdot c = (a \cdot c) + (b \cdot c)$	$(A \uplus B) * C \simeq (A * C) \uplus (B * C)$

Categorify: weak symmetric Rig Groupoids!

Let $c_1 : t_1 \leftrightarrow t_2$, $c_2 : t_3 \leftrightarrow t_4$, $c_3 : t_1 \leftrightarrow t_2$, and $c_4 : t_3 \leftrightarrow t_4$.
Let $a_1 : t_5 \leftrightarrow t_1$, $a_2 : t_6 \leftrightarrow t_2$, $a_3 : t_1 \leftrightarrow t_3$, and $a_4 : t_2 \leftrightarrow t_4$.

$$\frac{c_1 \Leftrightarrow c_3 \quad c_2 \Leftrightarrow c_4}{c_1 \oplus c_2 \Leftrightarrow c_3 \oplus c_4} \qquad \frac{c_1 \Leftrightarrow c_3 \quad c_2 \Leftrightarrow c_4}{c_1 \otimes c_2 \Leftrightarrow c_3 \otimes c_4}$$

$$(a_1 \odot a_3) \oplus (a_2 \odot a_4) \Leftrightarrow (a_1 \oplus a_2) \odot (a_3 \oplus a_4)$$

$$(a_1 \odot a_3) \otimes (a_2 \odot a_4) \Leftrightarrow (a_1 \otimes a_2) \odot (a_3 \otimes a_4)$$

Categorify: weak symmetric Rig Groupoids!

Let $c_1 : t_1 \leftrightarrow t_2$, $c_2 : t_2 \leftrightarrow t_3$, and $c_3 : t_3 \leftrightarrow t_4$:

$$
\begin{array}{c}
c_1 \odot (c_2 \odot c_3) \Leftrightarrow (c_1 \odot c_2) \odot c_3 \\
(c_1 \oplus (c_2 \oplus c_3)) \odot \mathit{assocl}_+ \Leftrightarrow \mathit{assocl}_+ \odot ((c_1 \oplus c_2) \oplus c_3) \\
(c_1 \otimes (c_2 \otimes c_3)) \odot \mathit{assocl}_\times \Leftrightarrow \mathit{assocl}_\times \odot ((c_1 \otimes c_2) \otimes c_3) \\
((c_1 \oplus c_2) \oplus c_3) \odot \mathit{assocr}_+ \Leftrightarrow \mathit{assocr}_+ \odot (c_1 \oplus (c_2 \oplus c_3)) \\
((c_1 \otimes c_2) \otimes c_3) \odot \mathit{assocr}_\times \Leftrightarrow \mathit{assocr}_\times \odot (c_1 \otimes (c_2 \otimes c_3)) \\
\mathit{assocr}_+ \odot \mathit{assocr}_+ \Leftrightarrow ((\mathit{assocr}_+ \oplus \mathit{id}{\leftrightarrow}) \odot \mathit{assocr}_+) \odot (\mathit{id}{\leftrightarrow} \oplus \mathit{assocr}_+) \\
\mathit{assocr}_\times \odot \mathit{assocr}_\times \Leftrightarrow ((\mathit{assocr}_\times \otimes \mathit{id}{\leftrightarrow}) \odot \mathit{assocr}_\times) \odot (\mathit{id}{\leftrightarrow} \otimes \mathit{assocr}_\times)
\end{array}
$$

Categorify: weak symmetric Rig Groupoids!

Let $c_1 : t_1 \leftrightarrow t_2$, $c_2 : t_3 \leftrightarrow t_4$, and $c_3 : t_5 \leftrightarrow t_6$:

$$((c_1 \oplus c_2) \otimes c_3) \odot \mathit{dist} \Leftrightarrow \mathit{dist} \odot ((c_1 \otimes c_3) \oplus (c_2 \otimes c_3))$$
$$(c_1 \otimes (c_2 \oplus c_3)) \odot \mathit{distl} \Leftrightarrow \mathit{distl} \odot ((c_1 \otimes c_2) \oplus (c_1 \otimes c_3))$$
$$((c_1 \otimes c_3) \oplus (c_2 \otimes c_3)) \odot \mathit{factor} \Leftrightarrow \mathit{factor} \odot ((c_1 \oplus c_2) \otimes c_3)$$
$$((c_1 \otimes c_2) \oplus (c_1 \otimes c_3)) \odot \mathit{factorl} \Leftrightarrow \mathit{factorl} \odot (c_1 \otimes (c_2 \oplus c_3))$$

Categorify: weak symmetric Rig Groupoids!

Let $c_0, c_1, c_2, c_3 : t_1 \leftrightarrow t_2$ and $c_4, c_5 : t_3 \leftrightarrow t_4$:

$$id\!\!\leftrightarrow \odot\, c_0 \Leftrightarrow c_0 \quad c_0 \odot id\!\!\leftrightarrow \Leftrightarrow c_0 \quad c_0 \odot !\, c_0 \Leftrightarrow id\!\!\leftrightarrow \quad !\, c_0 \odot c_0 \Leftrightarrow id\!\!\leftrightarrow$$

$$id\!\!\leftrightarrow \oplus\, id\!\!\leftrightarrow \Leftrightarrow id\!\!\leftrightarrow \qquad id\!\!\leftrightarrow \otimes\, id\!\!\leftrightarrow \Leftrightarrow id\!\!\leftrightarrow$$

$$c_0 \Leftrightarrow c_0 \qquad \frac{c_1 \Leftrightarrow c_2 \quad c_2 \Leftrightarrow c_3}{c_1 \Leftrightarrow c_3} \qquad \frac{c_1 \Leftrightarrow c_4 \quad c_2 \Leftrightarrow c_5}{c_1 \odot c_2 \Leftrightarrow c_4 \odot c_5}$$

Categorify: weak symmetric Rig Groupoids!

Let $c_0 : 0 \leftrightarrow 0$, $c_1 : 1 \leftrightarrow 1$, and $c_3 : t_1 \leftrightarrow t_2$:

$$\begin{array}{rr}
\mathit{unite}_{+}l \odot c_3 \Leftrightarrow (c_0 \oplus c_3) \odot \mathit{unite}_{+}l & \mathit{uniti}_{+}l \odot (c_0 \oplus c_3) \Leftrightarrow c_3 \odot \mathit{uniti}_{+}l \\
\mathit{unite}_{+}r \odot c_3 \Leftrightarrow (c_3 \oplus c_0) \odot \mathit{unite}_{+}r & \mathit{uniti}_{+}r \odot (c_3 \oplus c_0) \Leftrightarrow c_3 \odot \mathit{uniti}_{+}r \\
\mathit{unite}_{\times}l \odot c_3 \Leftrightarrow (c_1 \otimes c_3) \odot \mathit{unite}_{\times}l & \mathit{uniti}_{\times}l \odot (c_1 \otimes c_3) \Leftrightarrow c_3 \odot \mathit{uniti}_{+}l \\
\mathit{unite}_{\times}r \odot c_3 \Leftrightarrow (c_3 \otimes c_1) \odot \mathit{unite}_{\times}r & \mathit{uniti}_{\times}r \odot (c_3 \otimes c_1) \Leftrightarrow c_3 \odot \mathit{uniti}_{\times}r
\end{array}$$

$$\mathit{unite}_{\times}l \Leftrightarrow \mathit{distl} \odot (\mathit{unite}_{\times}l \oplus \mathit{unite}_{\times}l)$$

$$\mathit{unite}_{+}l \Leftrightarrow \mathit{swap}_{+} \odot \mathit{unite}_{+}r \qquad \mathit{unite}_{\times}l \Leftrightarrow \mathit{swap}_{\times} \odot \mathit{unite}_{\times}r$$

Categorify: weak symmetric Rig Groupoids!

Let $c_1 : t_1 \leftrightarrow t_2$ and $c_2 : t_3 \leftrightarrow t_4$:

$$swap_{+} \odot (c_1 \oplus c_2) \Leftrightarrow (c_2 \oplus c_1) \odot swap_{+} \quad swap_{\times} \odot (c_1 \otimes c_2) \Leftrightarrow (c_2 \otimes c_1) \odot$$

$$(assocr_{+} \odot swap_{+}) \odot assocr_{+} \Leftrightarrow ((swap_{+} \oplus id{\leftrightarrow}) \odot assocr_{+}) \odot (id{\leftrightarrow} \oplus sw$$

$$(assocl_{+} \odot swap_{+}) \odot assocl_{+} \Leftrightarrow ((id{\leftrightarrow} \oplus swap_{+}) \odot assocl_{+}) \odot (swap_{+} \oplus$$

$$(assocr_{\times} \odot swap_{\times}) \odot assocr_{\times} \Leftrightarrow ((swap_{\times} \otimes id{\leftrightarrow}) \odot assocr_{\times}) \odot (id{\leftrightarrow} \otimes sw$$

$$(assocl_{\times} \odot swap_{\times}) \odot assocl_{\times} \Leftrightarrow ((id{\leftrightarrow} \otimes swap_{\times}) \odot assocl_{\times}) \odot (swap_{\times} \otimes$$

$$unite_{+}r \oplus id{\leftrightarrow} \quad \Leftrightarrow \quad assocr_{+} \odot (id{\leftrightarrow} \oplus unite_{+}l)$$

$$unite_{\times}r \otimes id{\leftrightarrow} \quad \Leftrightarrow \quad assocr_{\times} \odot (id{\leftrightarrow} \otimes unite_{\times}l)$$

Categorify: weak symmetric Rig Groupoids!

Let $c : t_1 \leftrightarrow t_2$:

$$(c \otimes id\leftrightarrow) \odot absorbl \Leftrightarrow absorbl \odot id\leftrightarrow \qquad (id\leftrightarrow \otimes c) \odot absorbr \Leftrightarrow absorbr \odot$$

$$id\leftrightarrow \odot factorzl \Leftrightarrow factorzl \odot (id\leftrightarrow \otimes c) \qquad id\leftrightarrow \odot factorzr \Leftrightarrow factorzr \odot (c \otimes$$

$$absorbr \Leftrightarrow absorbl$$

$$absorbr \Leftrightarrow (distl \odot (absorbr \oplus absorbr)) \odot unite_+l$$

$$unite_\times r \Leftrightarrow absorbr \qquad absorbl \Leftrightarrow swap_\times \odot absorbr$$

$$absorbr \Leftrightarrow (assocl_\times \odot (absorbr \otimes id\leftrightarrow)) \odot absorbr$$

$$(id\leftrightarrow \otimes absorbr) \odot absorbl \Leftrightarrow (assocl_\times \odot (absorbl \otimes id\leftrightarrow)) \odot absorbr$$

$$id\leftrightarrow \otimes unite_+l \Leftrightarrow (distl \odot (absorbl \oplus id\leftrightarrow)) \odot unite_+l$$

Categorify: weak symmetric Rig Groupoids!

$$((assocl_+ \otimes id\leftrightarrow) \odot dist) \odot (dist \oplus id\leftrightarrow) \Leftrightarrow (dist \odot (id\leftrightarrow \oplus dist)) \odot assocl_+$$

$$assocl_\times \odot distl \Leftrightarrow ((id\leftrightarrow \otimes distl) \odot distl) \odot (assocl_\times \oplus assocl_\times)$$

$$\begin{aligned}(distl \odot (dist \oplus dist)) \odot assocl_+ \quad &\Leftrightarrow \quad dist \odot (distl \oplus distl) \odot assocl_+ \odot \\ &\quad (assocr_+ \oplus id\leftrightarrow) \odot \\ &\quad ((id\leftrightarrow \oplus swap_+) \oplus id\leftrightarrow) \odot \\ &\quad (assocl_+ \oplus id\leftrightarrow)\end{aligned}$$

$$\begin{aligned}(id\leftrightarrow \otimes swap_+) \odot distl \quad &\Leftrightarrow \quad distl \odot swap_+ \\ dist \odot (swap_\times \oplus swap_\times) \quad &\Leftrightarrow \quad swap_\times \odot distl\end{aligned}$$

So what?

Up shot: coherence theorem (like the one for monoidal categories).

Sound and complete set of rewrites / optimization rules for type equivalences.

More lens programs

data Colour : Set where red green blue : Colour

$\exists$-Colour-in-A+A+A : $(A : \mathsf{Set}) \to \mathsf{Lens}_1\ (A \uplus A \uplus A)$ Colour

More lens programs

data Colour : Set where red green blue : Colour

$\exists$-Colour-in-A+A+A : $(A : \text{Set}) \to \text{Lens}_1\ (A \uplus A \uplus A)$ Colour
GS-Colour-in-A+A+A : $(A : \text{Set}) \to$ GS-Lens $(A \uplus A \uplus A)$ Colour

For $n \in \mathbb{N}$ and nA, Lens_1 proof terms $O(n)$, GS-Lens are $O(n^3)$.

More lens programs

data Colour : Set where red green blue : Colour

$\exists$-Colour-in-A+A+A : $(A : \mathsf{Set}) \to \mathsf{Lens}_1\ (A \uplus A \uplus A)$ Colour
GS-Colour-in-A+A+A : $(A : \mathsf{Set}) \to$ GS-Lens $(A \uplus A \uplus A)$ Colour

For $n \in \mathbb{N}$ and nA, Lens_1 proof terms $O(n)$, GS-Lens are $O(n^3)$.

Inspired by Quantum Computing:
gcnot-equiv : $\{A\ B\ C : \mathsf{Set}\} \to$
$((A \uplus B) \times (C \uplus C)) \simeq ((A \uplus B) \times (C \uplus C))$
gcnot-equiv = factorequiv $\bullet$ id$\simeq$ $\uplus\simeq$ (id$\simeq$ $\times\simeq$ swap$_+$equiv) $\bullet$ distequiv

gcnot-lens : $\{A\ B\ C : \mathsf{Set}\} \to \mathsf{Lens}_1\ ((A \uplus B) \times (C \uplus C))\ (C \uplus C)$
gcnot-lens $\{A\}\ \{B\}$ = $\exists$-lens gcnot-equiv

Optics

Equality	$S = A$
Iso	$S \simeq A$
Lens	$\exists C.S \simeq C \times A$
Prism	$\exists C.S \simeq C + A$
Achroma	$\exists C.S \simeq (\top \uplus C) \times A)$
Affine	$\exists C, D.S \simeq D \uplus (C \times A)$
Grate	$\exists I.S \simeq I \to A$
Setter	$\exists F : Set \to Set.S \simeq FA$

Interfaces for Lens, Prism, etc: $\exists$-free equivalent!

Geometric interpretation

Iso	$S \simeq A$	S is A up to change of coordinates (CoC)
Lens	$\exists C. S \simeq C \times A$	S is a *cartesian product* over A, up to CoC
Prism	$\exists C. S \simeq C + A$	S has a *partition* into A and C, up to CoC

Take home: Really dig into PL for change of coordinates