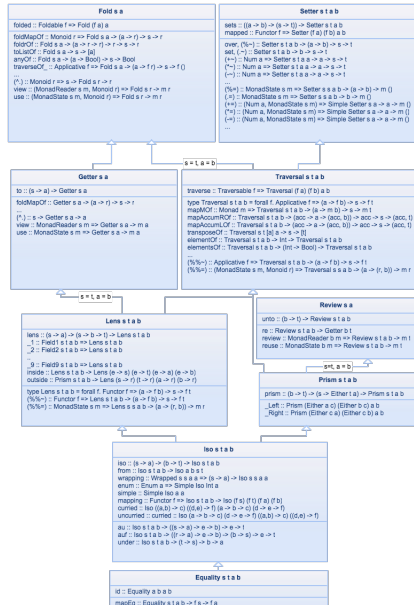


# Optics and Type Equivalences

Jacques Carette, Amr Sabry



Based on a 'reversible' core:

## **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
```

# Lens in Haskell

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

# Lens in Haskell

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

Example?

# Lens in Haskell

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

Example? Laws? Optimizations?

# 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.

# 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

where

**record** **isqinv** { $\ell \ell'$ } { $A : \text{Set } \ell$ } { $B : \text{Set } \ell'$ } ( $f : A \rightarrow B$ ) :

**Set** ( $\ell \sqcup \ell'$ ) **where**

**constructor** **qinv**

**field**

$g : B \rightarrow A$

$\alpha : (f \circ g) \sim \text{id}$

$\beta : (g \circ f) \sim \text{id}$

$\underline{\simeq} : \forall \{ \ell \ell' \} \rightarrow \text{Set } \ell \rightarrow \text{Set } \ell' \rightarrow \text{Set } (\ell \sqcup \ell')$

$A \simeq B = \Sigma (A \rightarrow B) \text{ isqinv}$

# 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)
```

`complete` requires moving to `Setoid`, which we'll elide here – see online code.