Shattered lens

Oleg Grenrus

MSFP 2020 · online · 2020-09-01

The record update problem

Lenses are very powerful and useful things in FP. They represent “first class” fields for a data structure.

If we have a `User`, which has a `Name`, which consists of first name (a `String`) and some other data:

```
data User = MkUser { userName :: Name , ... }
data Name = MkName { firstName :: String , ... }
```

then to update user’s first name we need to see some trouble.

Lenses offer a solution

If we use lenses

```
userNameL  :: Lens User Name
firstNameL :: Lens Name String
```

which compose

```
userNameL % firstNameL :: Lens User String
```

Then using set operation

```
set :: Lens a b -> b -> a -> a
```

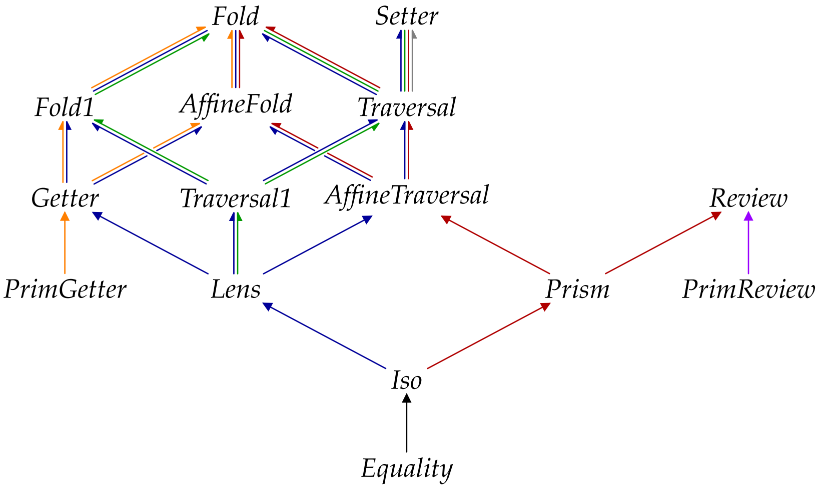
we can set new first name concisely:

```
setFirstName :: String -> User -> User
setFirstName = set (userNameL % firstNameL)
```

Proving facts about lenses is not simple. We usually need **functional extensionality** and some **proof irrelevance**.

Homotopy Type Theory (HoTT) gives a new perspective to look at things. And **Cubical Agda** allows to play with the ideas.

Lenses are not the only optics out there



Isomorphisms

Isomorphism

A function $f : A \to B$ is an **isomorphism**, if there exists $g : B \to A$, such that $f \circ g = 1_B$ and $g \circ f = 1_A$.

It is an easy exercise to show that inverse g is unique if it exists, so we don't need to require that explicitly.

However, together with equality proofs, a **quasi-inverse** of f

$$\mathrm{qinv}\, f :\equiv \sum_{g:B \to A} (f \circ g = 1_B) \times (g \circ f = 1_A)$$

is not unique (we don't have UIP).

Equivalence

A function $f : A \to B$ is an **equivalence** if for all $b : B$ the fibers of f over b are contractible.

$$\mathrm{isEquiv}\, f := \prod_{b:B} \mathrm{isContr}\,(\mathrm{fiber}_f\, b)$$

$$A \simeq B := \sum_{f:A\to B} \mathrm{isEquiv}\, f$$

where

$$\mathrm{isContr}\, A := \sum_{x:A} \prod_{y:A} x =_A y$$ “exists unique”

$$\mathrm{fiber}_f\, b := \sum_{a:A} f\, b =_A a$$ preimage of a point

Mere proposition

For all A, isEquiv A is a **mere proposition**, which means that all values of isEquiv A are equal. For example proving that composition of equivalences is associative reduces to proving that function composition is associative.

```
compEquiv-assoc
  : {ab : A ≃ B} → {bc : B ≃ C} → {cd : C ≃ D}
  → compEquiv (compEquiv ab bc) cd
  ≡ compEquiv ab (compEquiv bc cd)
compEquiv-assoc = ΣProp≡ isPropIsEquiv refl
```

Note: mere propositions are normal Types, they are not in a proof-irrelevant universe.

Prisms

Prisms

A **prism** from S to V consists of

- a **matcher** $f : S \to \text{Maybe}\, V$ and
- a **builder** $g : V \to S$

satisfying following laws:

MATCHBUILD	$\forall(v : V), f\,(g\,v) = \text{just}\, v$
BUILDMATCH	$\forall(s : S), f\,s = \text{just}\, v \Rightarrow g\,v = s$

Counting prisms between finite sets

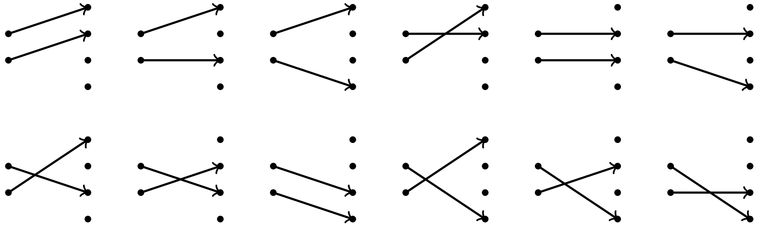
```
[ (f , g)
| f <- gen , g <- gen
, and [ f (g v) == Just v | v <- gen ]
, and [ g v == s | s <- gen , Just v <- [f s ] ] ]
```

Prisms between Fin 4 and Fin 2:

Counting prisms between finite sets

```
[ (f , g)
| f <- gen , g <- gen
, and [ f (g v) == Just v | v <- gen ]
, and [ g v == s | s <- gen , Just v <- [f s ] ] ]
```

Prisms between Fin 4 and Fin 2:



There are $12 = 4!/(4-2)!$.

Embedding

The HoTT version of **injection** is an **embedding**.

A function $f : A \to B$ is an embedding if for all $b : B$ the fibres of f over b are mere propositions.

$$\mathrm{hasPropFibers}\, f := \prod_{b:B} \mathrm{isProp}\left(\mathrm{fiber}_f\, b\right)$$

where

$$\mathrm{isProp}\, A := \prod_{x:A} \prod_{y:A} x =_A y$$

Decidable Embedding

The isProp value tells us only that it the value is unique if it exists, but it doesn't give any means to construct it! We need something stronger.

Using

$$
\begin{aligned}
& \mathrm{isDecProp}\, A :\equiv \mathrm{isProp}\, A \times (A + \neg A) \\
\text{or} \qquad & \mathrm{isDecProp}\, A :\equiv \mathrm{isContr}\, A + \neg A
\end{aligned}
$$

we can define

$$
\mathrm{isBuilder}\, g :\equiv \prod_{b:B} \mathrm{isDecProp}\left(\mathrm{fiber}_g\, b\right)
$$

Corollaries

- Every equivalence is a builder
- Builder composition is associative, ...
- Builder **uniquely** determines the matcher part of a prism.

Corollaries

- Every equivalence is a builder
- Builder composition is associative, ...
- Builder **uniquely** determines the matcher part of a prism.

A **Prism** is just a **decidable embedding**.

Lenses

Lenses

A **lens** from S to V consists of

- a **getter** $f : S \to V$ and
- a **setter** $g : S \to V \to S$

satisfying following laws

$$
\begin{array}{ll}
\textsc{GetPut} & \forall (s : S)\,(v : V),\, f\,(g\,s\,v) = v \\
\textsc{PutGet} & \forall (s : S),\, g\,s\,(f\,s) = s \\
\textsc{PutPut} & \forall (s : S)\,(v : V)\,(v' : V),\, g\,(g\,s\,v')\,v = g\,s\,v
\end{array}
$$

Higher lenses

We can have different lens variants:

- barely behaving lens (GETPUT)
- well behaved lens (GETPUT + PUTGET)
- very well behaved lens (GETPUT + PUTGET + PUTPUT)
- weak lens (GETPUT and weaker notion of PUTGET and PUTPUT)

Counting lenses between finite sets

Let’s take as a getter $\mathrm{fst} : \mathrm{Fin}\ 2 \times \mathrm{Fin}\ 2 \to \mathrm{Fin}\ 2$

Counting lenses between finite sets

Let's take as a getter $\mathsf{fst} : \mathsf{Fin}\ 2 \times \mathsf{Fin}\ 2 \to \mathsf{Fin}\ 2$

- Out of $4^{4\times 2} =$ **66536** setter candidates:

Counting lenses between finite sets

Let's take as a getter $\mathsf{fst} : \mathsf{Fin}\ 2 \times \mathsf{Fin}\ 2 \to \mathsf{Fin}\ 2$

- Out of $4^{4\times 2} =$ **66536** setter candidates:
- **256** barely behaving lenses.

$$\mathsf{isBareLens}\, f := \prod_{v_1 : V} \prod_{v_2 : V} \left(\mathsf{fiber}_f\, v_1 \to \mathsf{fiber}_f\, v_2\right)$$

Counting lenses between finite sets

Let's take as a getter $\mathsf{fst} : \mathsf{Fin}\ 2 \times \mathsf{Fin}\ 2 \to \mathsf{Fin}\ 2$

- Out of $4^{4\times 2} =$ **66536** setter candidates:
- **256** barely behaving lenses.

$$\mathsf{isBareLens}\, f := \prod_{v_1 : V} \prod_{v_2 : V} (\mathsf{fiber}_f\, v_1 \to \mathsf{fiber}_f\, v_2)$$

$$= \prod_{v_1 : V} \prod_{v2 : V} \sum_{s_1 : S} (f\, s_1 = v_1) \to \sum_{s_2 : S} (f\, s_2 = v_2)$$

Counting lenses between finite sets

Let's take as a getter $\mathrm{fst} : \mathrm{Fin}\ 2 \times \mathrm{Fin}\ 2 \to \mathrm{Fin}\ 2$

- Out of $4^{4\times 2} =$ **66536** setter candidates:
- **256** barely behaving lenses.

$$\mathrm{isBareLens}\, f := \prod_{v_1 : V} \prod_{v_2 : V} (\mathrm{fiber}_f\, v_1 \to \mathrm{fiber}_f\, v_2)$$

$$= \prod_{v_1 : V} \prod_{v2 : V} \sum_{s_1 : S} (f\, s_1 = v_1) \to \sum_{s_2 : S} (f\, s_2 = v_2)$$

$$\approx V \to S \to S$$

Counting lenses between finite sets

Let's take as a getter $\mathsf{fst} : \mathsf{Fin}\ 2 \times \mathsf{Fin}\ 2 \to \mathsf{Fin}\ 2$

- Out of $4^{4\times 2} =$ **66536** setter candidates:
- **256** barely behaving lenses.
- **16** weak lenses.

$$\mathsf{isWeakLens}\ f := \prod_{v_1 : V} \prod_{v_2 : V} (\mathsf{fiber}_f\ v_1 \simeq \mathsf{fiber}_f\ v_2)$$

.

Counting lenses between finite sets

Let's take as a getter $\mathsf{fst} : \mathsf{Fin}\ 2 \times \mathsf{Fin}\ 2 \to \mathsf{Fin}\ 2$

- Out of $4^{4\times 2} =$ **66536** setter candidates:
- **256** barely behaving lenses.
- **16** weak lenses.

$$\mathsf{isWeakLens}\ f := \prod_{v_1 : V} \prod_{v_2 : V} (\mathsf{fiber}_f\ v_1 \simeq \mathsf{fiber}_f\ v_2)$$

  - Four equivalences: $\mathsf{fiber}_{\mathsf{fst}}\ 0_2 \simeq \mathsf{fiber}_{\mathsf{fst}}\ 0_2 \ldots$
  - Two options: id and not for each
  - In total $2^4 = 16$.

.

Counting lenses between finite sets

Let's take as a getter $\mathsf{fst} : \text{Fin } 2 \times \text{Fin } 2 \to \text{Fin } 2$

- Out of $4^{4\times 2} =$ **66536** setter candidates:
- **256** barely behaving lenses.
- **16** weak lenses.

$$\text{isWeakLens } f := \prod_{v_1 : V} \prod_{v_2 : V} (\text{fiber}_f\, v_1 \simeq \text{fiber}_f\, v_2)$$

.

Weak PutGet law

$s \mapsto g\, s\, (f\, s)$ is an equivalence

Counting lenses between finite sets

Let's take as a getter $\mathsf{fst} : \mathsf{Fin}\ 2 \times \mathsf{Fin}\ 2 \to \mathsf{Fin}\ 2$

- Out of $4^{4\times 2} =$ **66536** setter candidates:
- **256** barely behaving lenses.
- **16** weak lenses.
- **16** well behaving lenses. GETPUT and PUTGET

Counting lenses between finite sets

Let's take as a getter fst : Fin 2 $\times$ Fin 2 $\to$ Fin 2

- Out of $4^{4\times 2} =$ **66536** setter candidates:
- **256** barely behaving lenses.
- **16** weak lenses.
- **16** well behaving lenses.
- **Two** very well-behaved lenses: `g (x , y) v = (v , y)` and `g (x , y) v = (v , y ‘xor‘ x ‘xor‘ v)`

$$\text{isHigherLens}\, f := \sum_{P : \|V\| \to \text{Type}} \prod_{v:V} (\text{fiber}_f\, v \simeq P|v|)$$

Counting lenses between finite sets

Let's take as a getter $\mathrm{fst} : \mathrm{Fin}\ 2 \times \mathrm{Fin}\ 2 \to \mathrm{Fin}\ 2$

- Out of $4^{4\times 2} =$ **66536** setter candidates:
- **256** barely behaving lenses.
- **16** weak lenses.
- **16** well behaving lenses.
- **Two** very well-behaved lenses:

$$\mathrm{isHigherLens}\, f := \sum_{P : \|V\| \to \mathrm{Type}} \prod_{v : V} (\mathrm{fiber}_f\, v \simeq P|v|)$$

Another problem: Multiple values for the same setter:
$(\mathrm{const\ Bool}, \lambda_v \ldots \bullet \mathrm{idEquiv})$ and
$(\mathrm{const\ Bool}, \lambda_v \ldots \bullet \mathrm{notEquiv})$.

Summary

- Prisms are simply decidable embeddings
  - isDecProp is a useful concept in programming. For example dec-$\leq$: $\prod_{n:\mathbb{N}} \prod_{m:\mathbb{N}}$ isDecProp $(n \leq m)$ and
    `(<=?) :: Natural -> Natural -> Maybe Natural`
- No satisfying results for lenses
  - Getter doesn't determine whole lens.
  - isHigherLens has a "degree of freedom"
  - Are weak lenses useful?

Extra slides: hasGetter

For $g : S \to V \to S$:

$$\mathrm{hasGetter}\, g := \prod_{s:S} \mathrm{isContr}\left(\mathrm{fiber}_{g\,s}\, s\right)$$