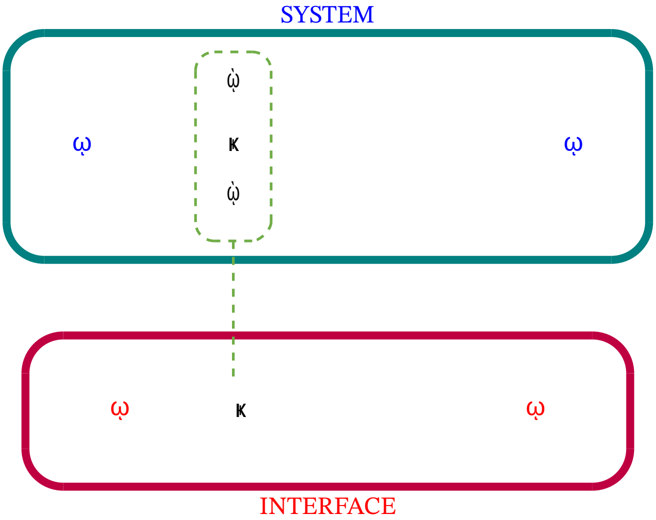
Limits and Colimits in a Category of Lenses

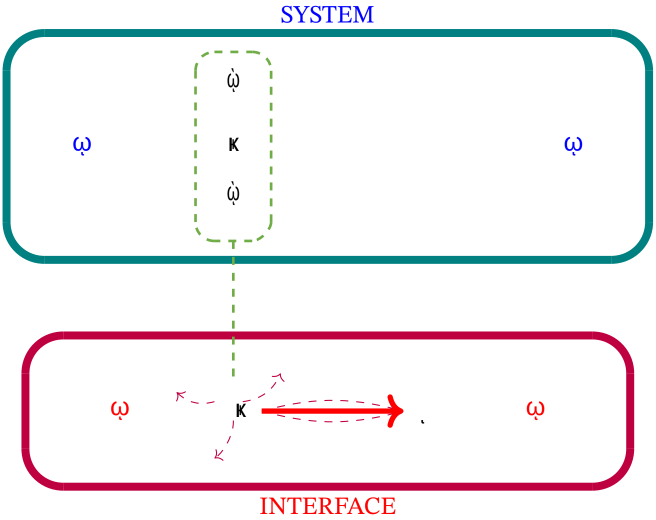
Emma Chollet, Bryce Clarke, Michael Johnson, Maurine Songa,
Vincent Wang, **Gioele Zardini**

4th International Conference on Applied Category Theory (ACT2021)
Cambridge (UK)

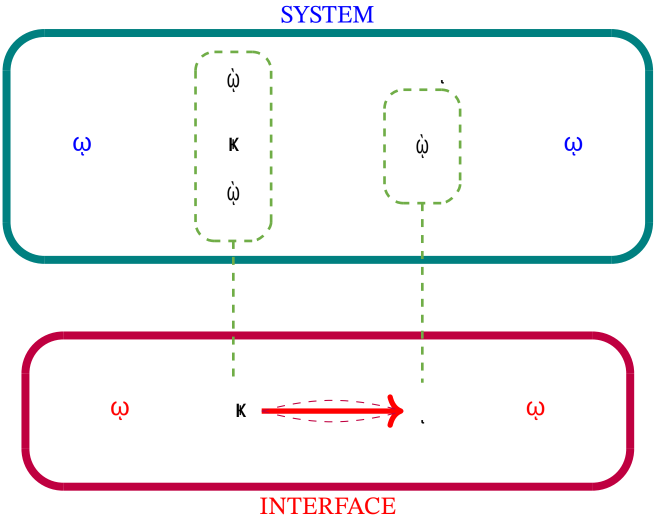
Lenses, informally



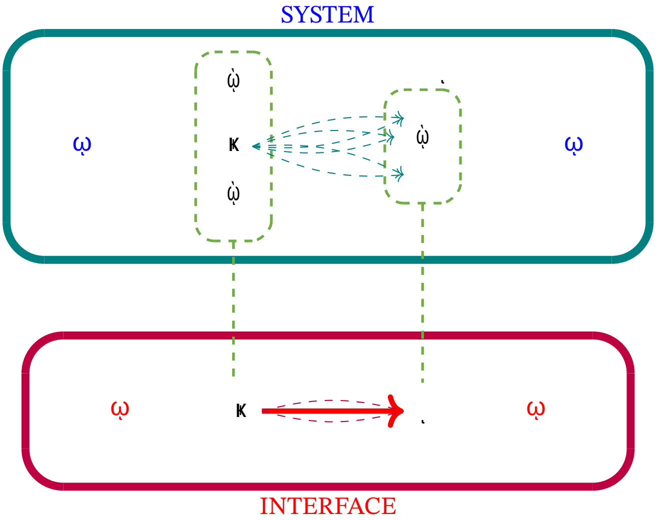
Lenses, informally



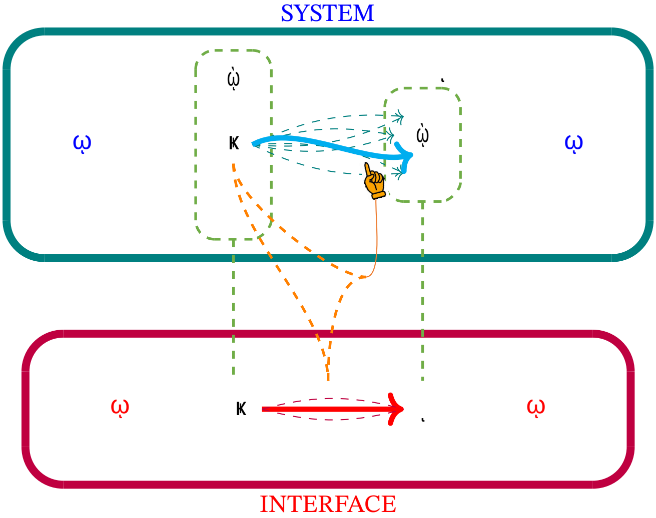
Lenses, informally



Lenses, informally



Lenses, informally



“Lenses”

- **Lawful**
- **Category-Based**
- **Asymmetric**

“Lenses”

- **Lawful**
  - (œŐű “Lawless”): Wild-West of Machine Learning, Game Theory, Economics…
- **Category-Based**
- **Asymmetric**

“Lenses”

- **Lawful**
  - (œŐű “Lawless”): Wild-West of Machine Learning, Game Theory, Economics…
- **Category-Based**
  - Strictly generalises Set-Based Lenses
  - Also known as “Delta Lenses”
- **Asymmetric**

“Lenses”

- **Lawful**
  - (œŐŭ “Lawless”): Wild-West of Machine Learning, Game Theory, Economics…
- **Category-Based**
  - Strictly generalises Set-Based Lenses
  - Also known as “Delta Lenses”
- **Asymmetric**
  - Asymmetric: One system knows everything the other does
  - Symmetric: Either system may know something the other doesn’t
  - All Symmetric Lenses can be constructed from Asymmetric ones

“Lenses”

- **Lawful (Specification for *ŋ̦ŀŀŋŀőő*)**
  - (œőŭ “Lawless”): Wild-West of Machine Learning, Game Theory, Economics…
- **Category-Based (Generalisation)**
  - Strictly generalises Set-Based Lenses
  - Also known as “Delta Lenses”
- **Asymmetric (Sufficiency)**
  - Asymmetric: One system knows everything the other does
  - Symmetric: Either system may know something the other doesn’t
  - All Symmetric Lenses can be constructed from Asymmetric ones

- *̦ŀŀ* but *Ńŀŋŀŏĺ’n* Lenses

Lenses, formally (Nuts-and-Bolts)

Definition

Let and be categories. A consists of a functor and a lifting operation,

which satisfies the following axioms:

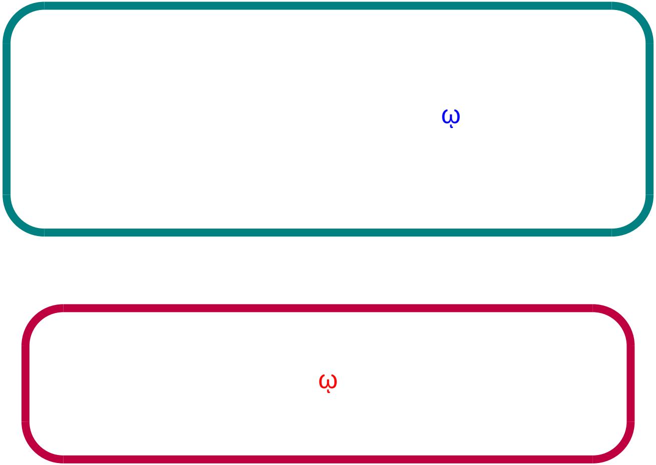
1. ;
2. ;
3. .

Lenses, formally (Nuts-and-Bolts)

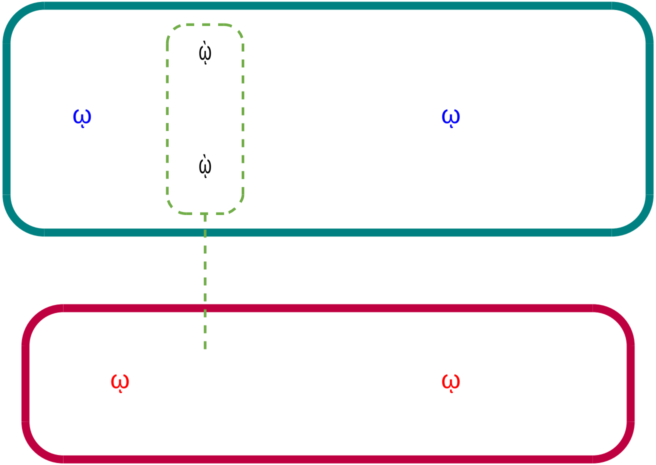
ῳ

ῳ

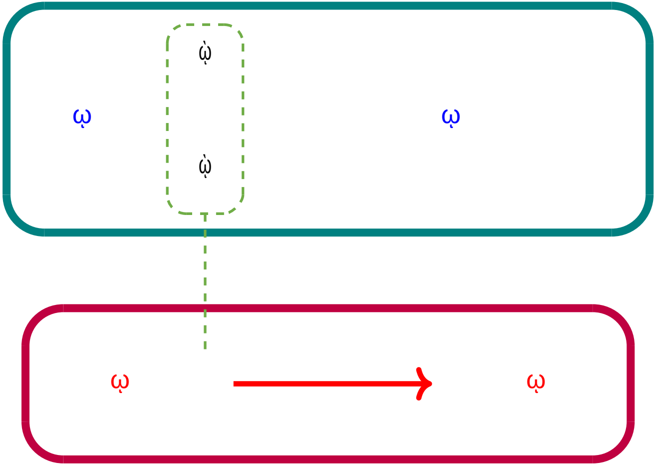
Lenses, formally (Nuts-and-Bolts)



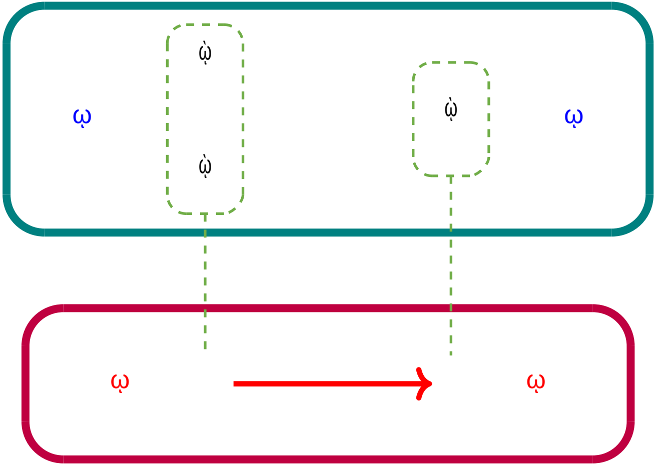
Lenses, formally (Nuts-and-Bolts)



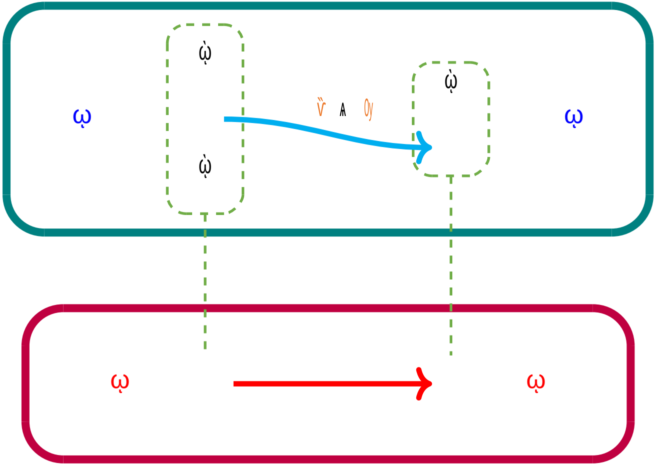
Lenses, formally (Nuts-and-Bolts)



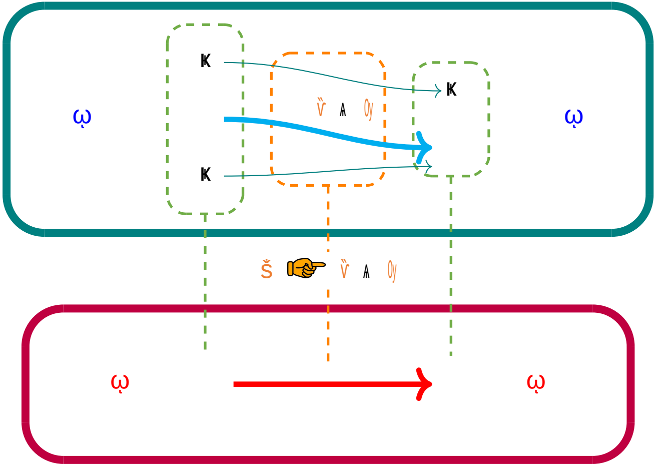
Lenses, formally (Nuts-and-Bolts)



Lenses, formally (Nuts-and-Bolts)



Lenses, formally (Nuts-and-Bolts)



Lenses, formally (Nuts-and-Bolts)

Definition

Let and be categories. A consists of a functor and a lifting operation,

which satisfies the following axioms:

1.
2.
3.

Lenses, formally (Nuts-and-Bolts)

Definition

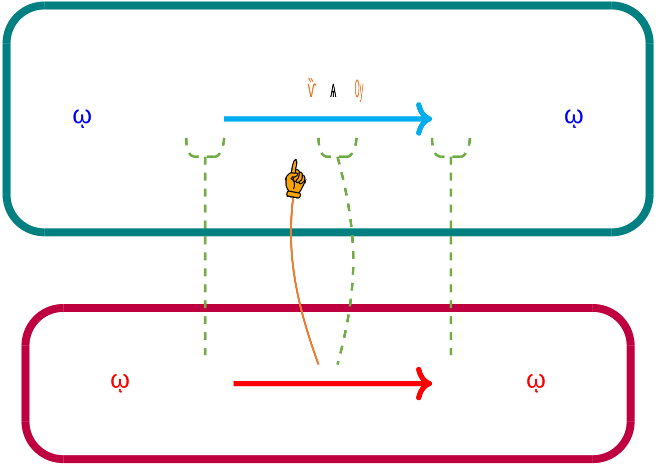
Let [illegible] and [illegible] be categories. A [illegible] consists of a functor [illegible] and a lifting operation,

$$[illegible]$$

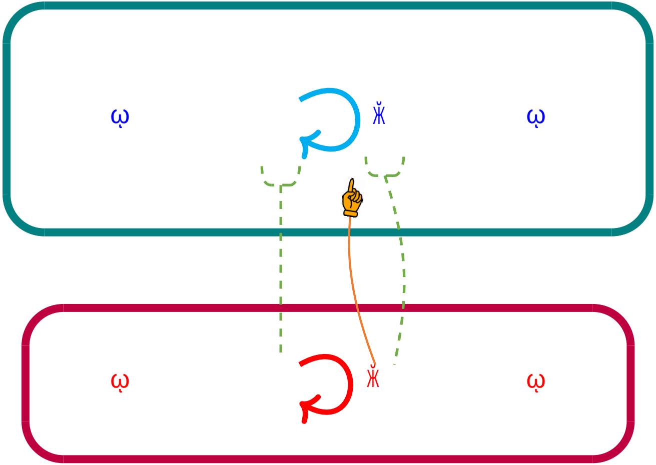
which satisfies the following axioms:

1. [illegible] **Put followed by Get is trivial for morphisms**
2. [illegible] **Get followed by Put preserves identities**
3. [illegible] **The Put of composites is the composite of Puts**

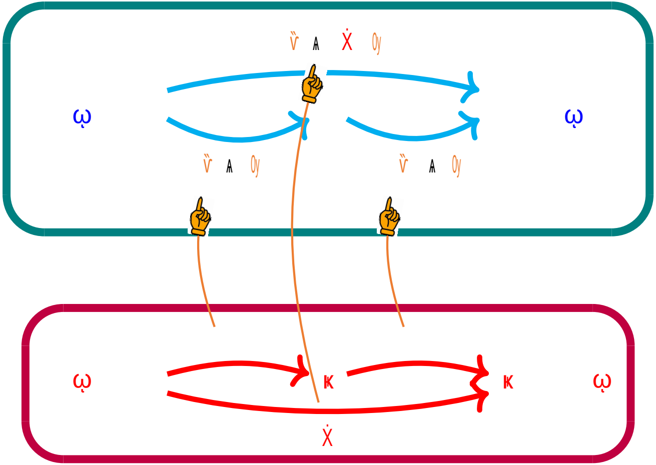
Lenses, formally (Nuts-and-Bolts)



Lenses, Formally (Nuts-and-Bolts)



Lenses, formally (Nuts-and-Bolts)



Lenses, formally (slick)

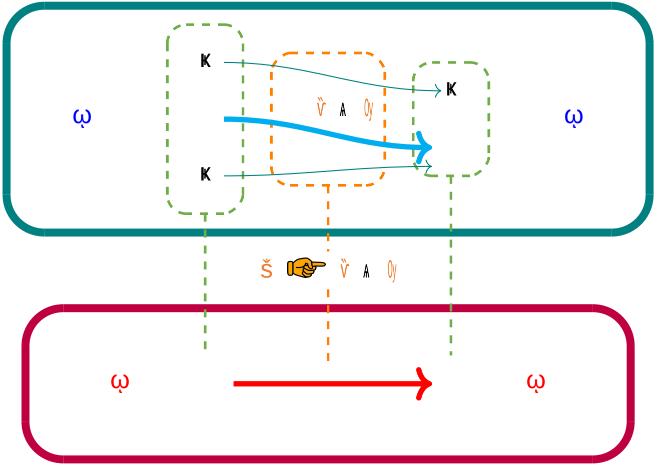
Proposition (Lenses as Functors and Cofunctors)

Every lens may be represented as a commutative diagram of functors,

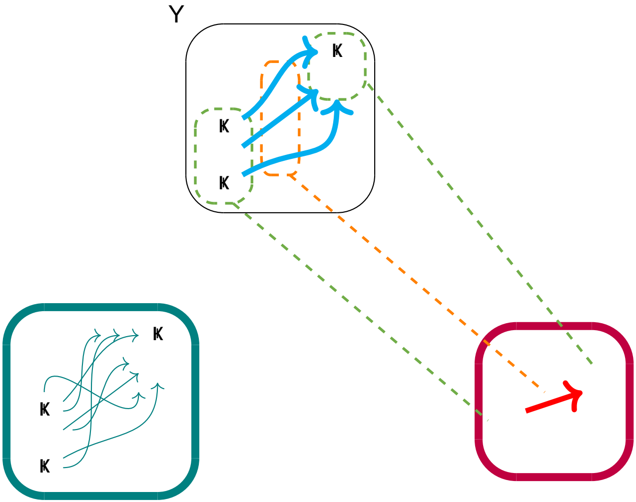
$$
\begin{tikzcd}
 & Y \arrow[dl] \arrow[dr] & \\
{} \arrow[rr] & & {}
\end{tikzcd},
$$

where is a faithful, identity-on-objects functor and , is a discrete opfibration.

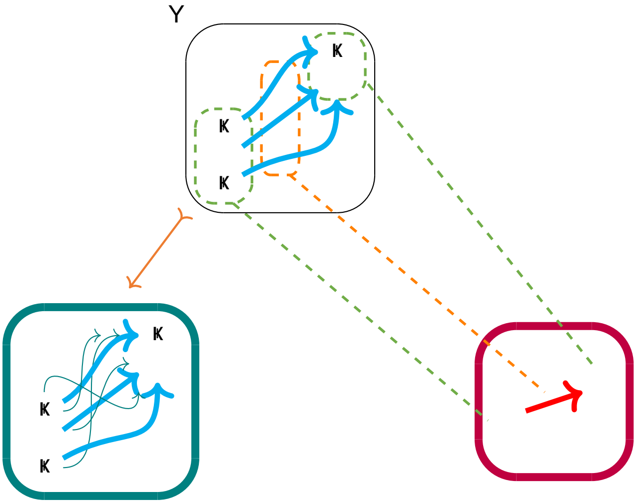
Lenses, formally (slick)



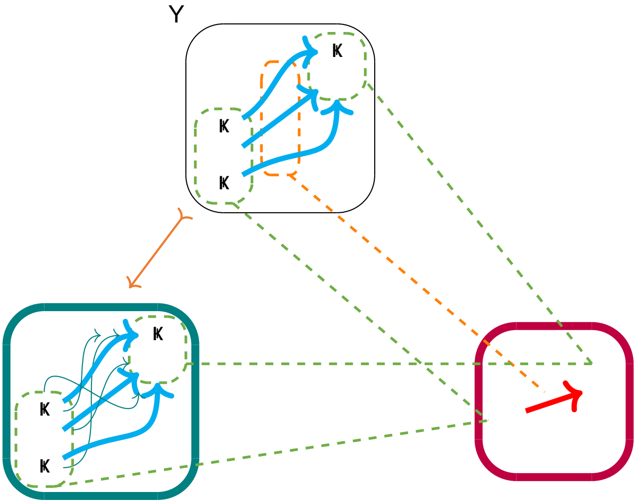
Lenses, formally (slick)



Lenses, formally (slick)



Lenses, formally (slick)



Lenses, formally (slick)

$$
\begin{tikzcd}
& Y \arrow[dl, "\text{Id-on-Obj.}"'] \arrow[dr, "\text{D.Opf.}"] & \\
{} \arrow[rr] & & {}
\end{tikzcd}
$$

The category ⁚ ŁŋŐ

Definition

Let ⁚ ŁŋŐ denote the category whose objects are categories and whose morphisms are lenses. Given a pair of lenses ꜿ ꜷ ꜿ Ư k̲ and ꜿ ꜷ ꜿ Ư k̲ , their composite is given by the functor Ẋ Ư ˜ together with the lifting operation:

ꜿ T̰ ꜷ Ư ˜ T̰ ꜿ ꜿ V̈ ꜷ V̈ ꜷ ŐyŐyਨ

Actually, ˸ Łŋő is a pretty place

Actually, ˙ łŋ0̋ is a pretty place

- ˙ łŋ0̋ has **initial** and **terminal** objects;

Actually, ː Łŋő is a pretty place

- ː Łŋő has **initial** and **terminal** objects;
- ː Łŋő has **small coproducts**;

Actually, ˈ Łŋ0̋ is a pretty place

- ˈ Łŋ0̋ has **initial** and **terminal** objects;
- ˈ Łŋ0̋ has **small coproducts**;
- ˈ Łŋ0̋ has **equalisers**;

Actually, ˸ ƚŋő is a pretty place

- ˸ ƚŋő has **initial** and **terminal** objects;
- ˸ ƚŋő has **small coproducts**;
- ˸ ƚŋő has **equalisers**;
- ˸ ƚŋő has an **orthogonal factorisation system**, which factors every lens into a surjective-on-objects lens (epic) followed by a injective-on-objects discrete opfibration (monic);

Actually, ˸ Łŋő is a pretty place

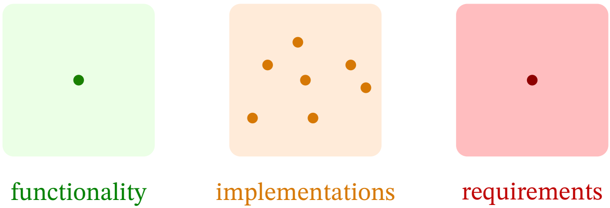
- ˸ Łŋő has **initial** and **terminal** objects;
- ˸ Łŋő has **small coproducts**;
- ˸ Łŋő has **equalisers**;
- ˸ Łŋő has an **orthogonal factorisation system**, which factors every lens into a surjective-on-objects lens (epic) followed by a injective-on-objects discrete opfibration (monic);
- Limit constructions imported from Ľő behave well, even if they are missing universal property in ˸ Łŋő: we have **distributivity** of ŅŊōŌŏőŁŀ products over coproducts, and **extensivity**.

Actually, ˈ Łŋő is a pretty place

- ˈ Łŋő has **initial** and **terminal** objects;
- ˈ Łŋő has **small coproducts**;
- ˈ Łŋő has **equalisers**;
- ˈ Łŋő has an **orthogonal factorisation system**, which factors every lens into a surjective-on-objects lens (epic) followed by a injective-on-objects discrete opfibration (monic);
- Limit constructions imported from Ľő behave well, even if they are missing universal property in ˈ Łŋő: we have **distributivity** of ŅŊōŌŏőŁŀ products over coproducts, and **extensivity**.
- (Next Talk): ˈ Łŋő has **(certain) coequalisers**

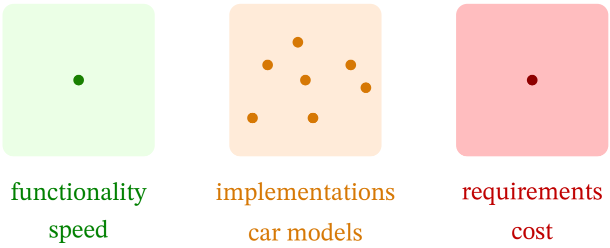
Engineering co-design as a guiding example

- Design is characterised by three spaces:
  - **implementation space**: the options we can choose from;
  - **functionality space**: what we need to provide/achieve;
  - **requirements/costs space**: resources we need to have available;



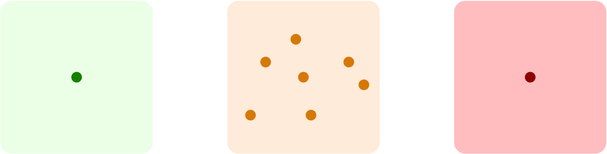
Engineering co-design as a guiding example

- Design is characterised by three spaces:
  - **implementation space**: the options we can choose from;
  - **functionality space**: what we need to provide/achieve;
  - **requirements/costs space**: resources we need to have available;



Engineering co-design as a guiding example

- Design is characterised by three spaces:
  - **implementation space**: the options we can choose from;
  - **functionality space**: what we need to provide/achieve;
  - **requirements/costs space**: resources we need to have available;



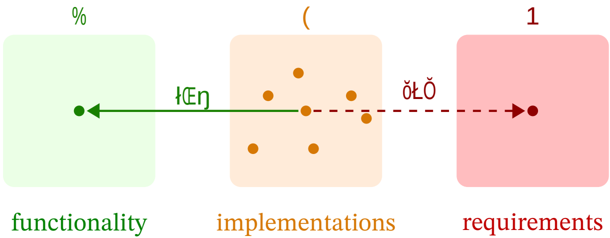
functionality | implementations | requirements
speed | car models | cost
capacity $\times$ max current | battery models | mass $\times$ cost

Design problems, formally

Definition

A *design problem with implementation* (DPI) is a tuple $\langle F, R, I, \mathsf{fun}, \mathsf{req}\rangle$, where:

- F is a poset, called *functionality space*;
- R is a poset, called *requirements space*;
- I is a set, called *implementation space*;
- the map $\mathsf{fun}: I \to F$ maps an implementation to the functionality it provides;
- the map $\mathsf{req}: I \to R$ maps an implementation to the resources it requires.



Practically, design problems can be understood as feasibility relations

- For design purposes, we need to know **how** something is done: we need the implementations
- For the algorithmic solution of co-design problems, we consider **feasibility relations** directly;
- A [illegible] is a **boolean profunctor**:

[illegible]

[illegible]

Practically, design problems can be understood as feasibility relations

- For design purposes, we need to know **how** something is done: we need the implementations
- For the algorithmic solution of co-design problems, we consider **feasibility relations** directly;
- A *I·ŁŐŅŃŋ ōŏŌľ'ȟŁŊ* is a **boolean profunctor**:

Ư %ŌŌ ⁎ 1 ˜ ōő ˏ ŌŌȟ

˧ ᵂA ˧ d̲ Ṫ Ṯ (Ư v̏ ڤ łŒŋv̏ 0y0y Ǎ v̏ŏŁŎv̏ 0y ڤ 0yӂ

- This is a **monotone** map (morphism in ŌŐ):
  - **Lower** functionalities do not require **more** requirements;
  - **Higher** requirements do not provide **less** functionalities

- Design problems form the category **DP**:
  - Objects are posets, morphisms are design problems;
  - Covered in detail in ACT4E (https://applied-compositional-thinking.engineering)

Realizing design problems as lenses

- Consider the design problem related to buying a car based on its speed:

Realizing design problems as lenses

- Consider the design problem related to buying a car based on its speed:

- We consider Ų %ŌŌ × 1 ˜ ŌŐ ¸ ŌŌħ with posets

fast
|
average

slow

%

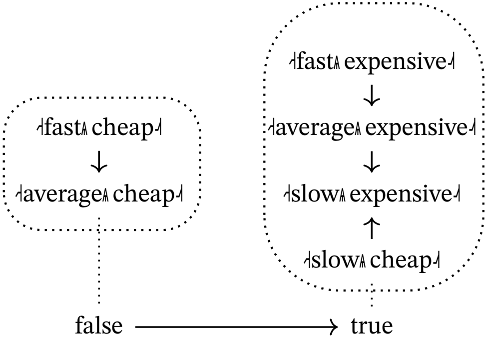
expensive
|
cheap

1

- *2ħŌŔ* vehicles are the only *Ŀ:ńŁĿ'ō* ones, the rest are *ŁŕōŁŋŐŅœŁ*.

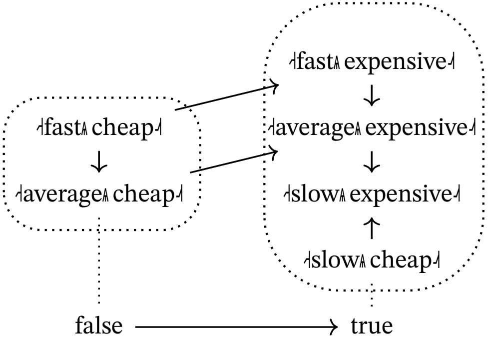
Realising design problems as lenses

- We can represent the functor [illegible].



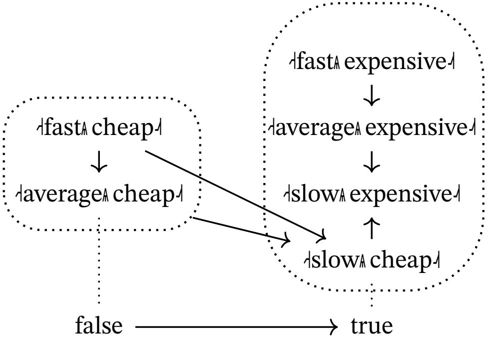
Realising design problems as lenses

- We can represent the functor %ŌŌ ⨯ 1 ˜ ŌŐ ¸ ŌŌh ˣ Iʼ̆oŁŔŅŐŁ.
- A lens over the functor provides a ŒŋŅŎŒŁủ ŏŁĽĿńĽ Iʼ̍nŁ pair in %ŌŌ ⨯ 1 from each infeasible pair.
- A lens models **feasibility** and informs **compromises** to make the unfeasible feasible.



Realising design problems as lenses

- We can represent the functor %ŌŌ × 1 ̃ ŌŐ ̦ ŌŌh ˣ l’ŏŁŔŅŐŁ.
- A lens over the functor provides a ŒŋŅŎŒŁủ ŏŁĽĿńĽ l’hŁ pair in %ŌŌ × 1 from each infeasible pair.
- A lens models **feasibility** and informs **compromises** to make the unfeasible feasible.



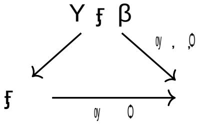
Łŋő has small coproducts

- Given lenses ꓧ ᴀ ꓧ Ụ ḵ , ꓧ ᴀ ꓧ Ụ ḵ , take the coproduct in Ľő: ʄ ;
- In Ľő, coproduct injection functors are **injective-on-objects discrete opfibrations**
- Given lenses ꓧ ᴀ ꓧ Ụ ḵ , ꓧ ᴀ ꓧ Ụ ḵ , we have a **unique** lens ʄ ḵ with:

$$\begin{tikzcd}
 & \text{Y ʄ β} \arrow[dl] \arrow[dr, "\text{ŋ , ,0}"] & \\
\text{ʄ} \arrow[rr, "\text{ŋ 0}"'] & & {}
\end{tikzcd}$$

has small coproducts

- Given lenses , , take the coproduct in : ;
- In , coproduct injection functors are **injective-on-objects discrete opfibrations**
- Given lenses , , we have a **unique** lens   with:



Example

- From speed × cost and seats × weight you get

$$\text{speed} \times \text{cost} \quad \text{seats} \times \text{weight}$$

ŁŋŐ has equalizers

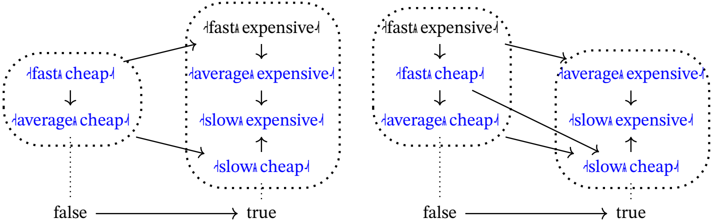
- Consider lenses Ⲻ ⲁ Ⲻ Ụ k̲ and Ⲻ ⲁ Ⲻ Ụ k̲
- One can construct the equaliser Ụ ⁻ of the **underlying functors** in ĽŐ.
- Then the equaliser is the largest subobject Ụ Ḍ such that Ẋ Ụ k̲ is a discrete opfibration which forms a cone over the parallel pair in ŁŋŐ.

ŁŋŐ has equalizers

- Consider lenses ⫞ ₳ ⫞ Ư k̲ and ⫞ ₳ ⫞ Ư k̲
- One can construct the equaliser Ư ˜ of the **underlying functors** in ĽŐ.
- Then the equaliser is the largest subobject Ư Ḍ such that Ẋ Ư k̲ is a discrete opfibration which forms a cone over the parallel pair in ŁŋŐ.

Example

- Consider two design problems (two experts) ⫞ ₳ ⫞ Ư %ŌŌ ⨯ 1 k̲ ¸ ŌŌħ, ⫞ ₳ ⫞ Ư %ŌŌ ⨯ 1 k̲ ¸ ŌŌħ
- Their equalizer k̲ %ŌŌ ⨯ 1:
  - **embeds** into %ŌŌ ⨯ 1, and selects pairs in %ŌŌ ⨯ 1 for which experts **agree**
  - In the worst case, **total disagreement**, i.e. d l.



Łŋő has an orthogonal factorisation system

- Johnson & Rosebrugh showed that Łŋő admits a **proper orthogonal factorisation system**
- This is actually an **(epi, mono)-factorisation system**, factoring every lens into:
  - A surjective-on-object lens (epimorphism), and
  - A cosieve (monomorphism).

Example

- Consider a lens ⱶ ⱥ ⱶ Ụ speed^{ōō} × cost ḵ ¸ ŌŌħ with just őŏŒŁ values

$$\begin{tikzcd}
\text{speed}^{ōō} \times \text{cost} \arrow[rr, two heads, "\text{epimorphism}"] \arrow[dr, two heads] & & \text{ọtrueỘ} \arrow[dl, tail, "\text{monomorphism}"] \\
& \text{ọfalse ˜ trueỘ} &
\end{tikzcd}$$

Conclusion and Outlook

- We considered ŋŅĿŁ but ŃŁŋŁŏĽŉ Lenses ŐŒ ˓ ĿŅŁŋőŉŖ ŏŅĿń to model problems of:
  - synchronisation
  - coordination
  - interoperation

Conclusion and Outlook

- We considered ŋŅĿŀ but ŃŁŋŁŏĽŉ Lenses ŐŒ ˬ ĿŅŁŋőŉŖ ŏŅĿń to model problems of:
  - synchronisation
  - coordination
  - interoperation
- We studied the category ˸ Łŋő to look for canonical constructions...

Conclusion and Outlook

- We considered ŋŅĿŁ but ŃŁŋŁŏĽŉ Lenses ŐŒ ˗ ĿŅŁŋőŉŖ ŏŅĿń to model problems of:
  - synchronisation
  - coordination
  - interoperation
- We studied the category ˸ Łŋő to look for canonical constructions...
- ...and we found some.