Order Theory and Concept Lattices in Ordered Categories Without Meets

AContext-2.0

Wolfram Kahl

kahl@cas.mcmaster.ca

Musa Al-hassy

alhassy@gmail.com

Department of Computing and Software, McMaster University
Hamilton, Ontario, Canada L8S 4K1

2015-05-03

Abstract

Using the the dependently-typed programming language Agda, we formalise orders, membership relations, and a category of algebraic contexts with relational homomorphisms presented by Jipsen (2012); Moshier (2013) together with the dual functors connecting this category with the category of complete semilattices with meet-preserving homomorphisms.

We do this in the abstract setting of locally ordered categories with converse (OCCs) with residuals and symmetric quotients but without requiring meets (as in allegories) or joins (as in Kleene categories). The abstract formalisation has the advantage that it can be used both for theoretical reasoning, and for executable implementations, by instantiating it with appropriate choices of concrete OCCs.

This research is supported by the National Science and Engineering Research Council (NSERC), Canada

Contents

1 Introduction

Order-theoretic concepts have long been a topic where relation algebra is fruitfully applied, Schmidt and Ströhlein (1993) devoted a whole chapter to "Transitivity", treating in particular orders, closures, bounds and extrema in the formal setting of relation algebra, where joins (unions), meets (intersections), and complements (negations) of relations are always available.

Building on the experience of Kahl (2014a), where a category of "contexts" (see below) was formalised in the setting of ordered categories with converse (OCCs), residuals, and power operators, but without meets (assumed in all allegories) and joins (assumed in all Kleene categories), we now show a treatment of a sizeable body of order theory in a similar setting. This may at first seem surprising: How can we express antisymmetry without meets? It turns out that assuming symmetric quotients is sufficient for expressing antisymmetry, and we use the definition provided by Furusawa and Kahl (1998) for symmetric quotient, which does not assume meets. Symmetric quotients, where they exist in an OCC with residuals, are meets of two residuals, but since symmetric quotients are always difunctional, and in most relevant OCCs, most morphisms are not difunctional, assuming just residuals and symmetric quotients is still far from assuming all meets.

In our development of order theory, we start with preorders, which do not require antisymmetry, and can therefore be formalised in a weaker setting. In turns out that for many purposes, we do net even need identities, and therefore set the parts of our development where this is possible in ordered semigroupoids with converse (OSGCs), see (Kahl, 2008). Our formalisation includes also the usual definition of membership relations and direct powers using symmetric quotients.

A useful tool for building up significant parts of order theory are Galois connections. We use, for example, the Galois connection between the upper- and lower-bound operators ubd and lbd in Categoric.OSGC.Preorder (Sect. 5.1) to obtain a number of derived properties of these — since ubd and lbd are operators that map morphisms of the underlying OSGC to morphisms, and satisfy the Galois connection properties with respect to the order of the relevant hom-posets, we call the formalisation of Galois connections used here "external". We also formalise Galois connections produced by OSGC/OCC mappings satisfying the Galois connection properties with respect to order "relations" embodied by endomorphisms on the source and target objects of these mappings — this kind of Galois connections is called "internal".

Formal concept analysis (FCA) Wille (2005) typically starts from a *context* (E, A, R) consisting of a set E of *entities* (or "objects"), a set A of *attributes*, and an *incidence* relation R from entities to attributes. In such a context, "concepts" arise as "Galois-closed" subsets of E respectively A, and form complete "concept lattices".

In a recent development, Moshier (2013) defined a novel *relational* context homomorphism concept that gives rise to a category of contexts that is dual to the category of complete meet semilattices; this is in contrast with the FCA literature, which typically derives the context homomorphism concept from that used for the concept lattices, as for example by Hitzler et al. (2006), with the notable exception of Erné (2005), who studied context homomorphisms consisting of pairs of mappings. Jipsen (2012) published the central definitions of Moshier's (2013) approach, and developed it further to obtain categories of context representations of not only complete lattices, but also different kinds of semirings.

Kahl (2014a) formalised this context homomorphism concept and the resulting category in OSGCs with residuals, and in addition and power transposes, which are a slightly weaker formalisation of set membership than direct powers. In the current work, we continue the project started in (Kahl, 2014a) by also formalising the category of complete lower semilattices and implementing the dual functors connecting this with the context category as outlined by Moshier (2013).

Overview

"External" Galois connections in partial orders formalised as relations (binary predicates) between elements of Agda types are presented in Chapter 2.

For reference, we include all theories of residuals and symmetric quotients that only need ordered categories with converse (OCCs) in Chapter 3.

Since formal concept analysis concentrates on subsets of the constituent sets of the contexts we are interested in, we formalise, in Chapter 4, the abstract version of element relations presented for example by Bird and de Moor (1997) directly in the setting of locally ordered semigroupoids with converse (OSGCs). Adding also residuals to that setting is sufficient for the formalisation of the *polarities* (Sect. 4.4) needed for formal concept analysis.

We then move to formalisation of order relations; although many of the concepts in the standard presentation, as for example by Schmidt and Ströhlein (1993), are there formulated using meets, we are able to essentially "port" all this material to the setting of OSGCs respectively OCCs with residuals and symmetric quotients. This development is in Chapter 5, and also includes the abstract version of element relations corresponding to the direct powers of Berghammer, Schmidt, and Zierer (1986; 1989) or the power allegories of Freyd and Scedrov (1990), which is slightly stronger than the power operators of Chapter 4; according to Bird and de Moor (1997, p. 106) (where the development uses at least allegories), proving antisymmetry of the inclusion relation induced by membership in their power transpose axiomatisation requires tabular allegories, while in direct powers it follows directly, even in our OCC setting. We conclude Chapter 5 with additional properties of the polarities of Sect. 4.4 that hold when their power operator is derived from a direct power.

Chapter 6 studies "internal" Galois connections in the context of orders as defined in Chapter 5, and uses them to derive useful results about polarities.

The definition of the context category following Moshier (2013) — see also (Jipsen, 2012) — is contained in Chapter 7; the publication (Kahl, 2014a) covers essentially Chapters 4 and 7.

After defining this category of contexts, Moshier (2013) goes on to prove its duality with the category of complete lower semilattices with meet-preserving homomorphisms. Chapter 8 contains the definition of this category, and currently only the object and morphism components of the functors constituting the duality. The completion of the duality is still future work.

The Agda source code for this development is available on-line at the following URL:

`http://relmics.mcmaster.ca/RATH-Agda/#AContext`

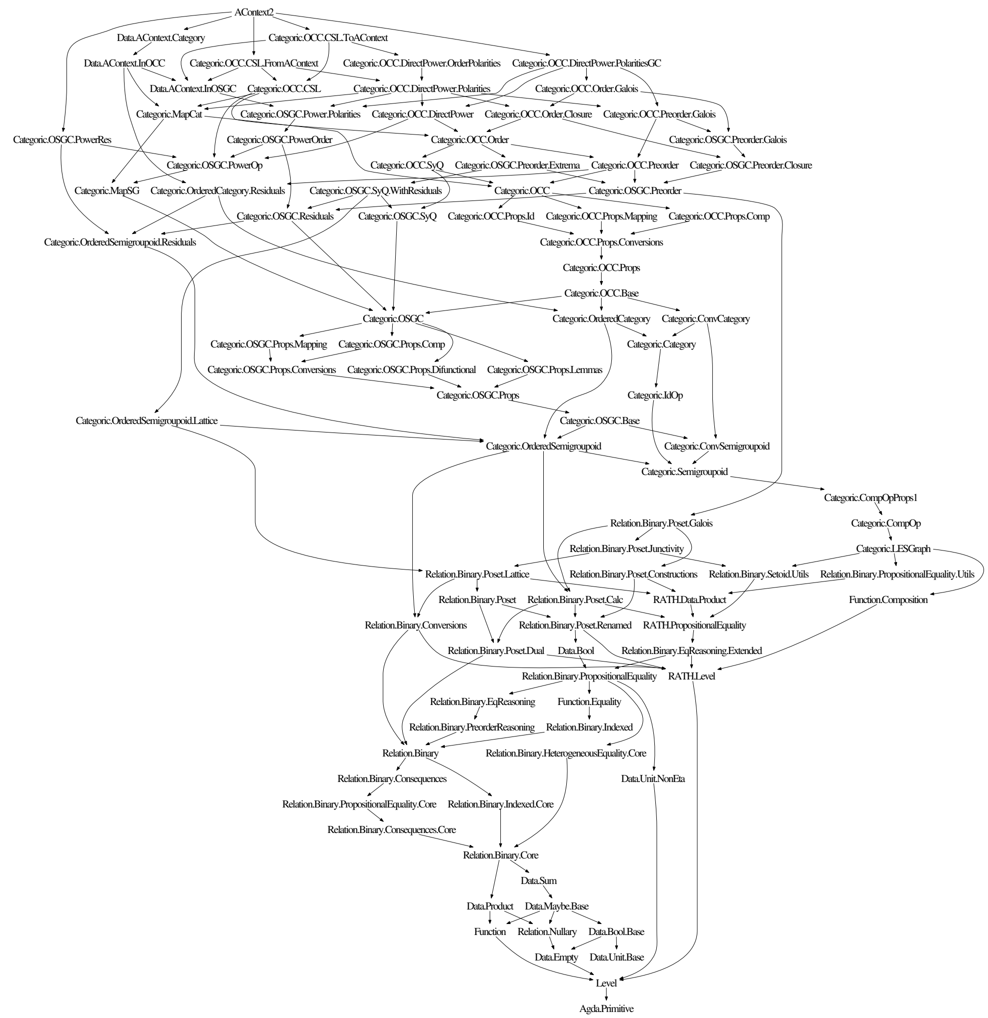
The source code available there includes the OSGC and OCC material (and everything needed for that) of the RATH-Agda library of Kahl (2011, 2014b), but we do not include this in the current document. The developments of external and internal Galois connections together with parts of (pre-)order theory originate from the M.Sc. thesis of Al-hassy (2015).

Top-Level Module

Loading this module forces typechecking of all theories contained in the AContext-2.0 document.

```
module AContext2 where
open import Categoric.OCC.CSL.FromAContext public
open import Categoric.OCC.CSL.ToAContext public
open import Categoric.OCC.DirectPower.PolaritiesGC
open import Categoric.OSGC.PowerRes
open import Data.AContext.Category
```

In the following module dependency graph, exactly Data.Bool and the modules it depends on are part of the Agda standard library.



2 External Galois Connections

The module Relation.Binary.Poset.Renamed (Sect. 2.1) is included only for reference, as it provides the naming conventions we use for posets.

The modules Relation.Binary.Poset.Constructions (Sect. 2.2), Relation.Binary.Poset.Junctivity (Sect. 2.3), and Relation.Binary.Poset.Galois (Sect. 2.4) contain new material; the latter a formalisation of Galois connections based on Agda functions between the carriers of Posets.

2.1 Relation.Binary.Poset.Renamed

The Posets defined in the standard library module Relation.Binary are Setoids, with equivalence relation ≈, with an additional compatible ordering relation ≤. For convenience, we rename the properties of these two relations so that the names refer to the relations, and bring them all into a single scope.

```
module Poset′ {j k₁ k₂ : Level} (poset : Poset j k₁ k₂) where
  open Poset poset public renaming
    (antisym  to ≤-antisym
    ; refl      to ≤-refl
    ; reflexive to ≤-reflexive
    ; trans     to ≤-trans
    )
  open IsEquivalence isEquivalence public renaming
    (refl      to ≈-refl
    ; sym       to ≈-sym
    ; trans     to ≈-trans
    ; reflexive to ≈-reflexive
    )
```

We also add some derived properties that will be used to abbreviate many proofs.

```
  ≤-reflexive′ : {R S : Carrier} → R ≈ S → S ≤ R
  ≤-reflexive′ eq = ≤-reflexive (≈-sym eq)

  ≤-trans₁ : {Q R S : Carrier} → Q ≤ R → R ≈ S → Q ≤ S
  ≤-trans₁ leq eq = ≤-trans leq (≤-reflexive eq)

  ≤-trans₂ : {Q R S : Carrier} → Q ≈ R → R ≤ S → Q ≤ S
  ≤-trans₂ eq leq = ≤-trans (≤-reflexive eq) leq

  infixl 1 _⟨≈≈⟩_ _⟨≈≈˘⟩_ _⟨≈˘≈⟩_ _⟨≈˘≈˘⟩_ _⟨≤≤⟩_ _⟨≤≈⟩_ _⟨≤≈˘⟩_ _⟨≈≤⟩_ _⟨≈˘≤⟩_
  _⟨≈≈⟩_ : {Q R S : Carrier} → Q ≈ R → R ≈ S → Q ≈ S
  _⟨≈≈⟩_ = ≈-trans

  _⟨≈≈˘⟩_ : {Q R S : Carrier} → Q ≈ R → S ≈ R → Q ≈ S
  _⟨≈≈˘⟩_ x y = ≈-trans x (≈-sym y)

  _⟨≈˘≈⟩_ : {Q R S : Carrier} → R ≈ Q → R ≈ S → Q ≈ S
  _⟨≈˘≈⟩_ x y = ≈-trans (≈-sym x) y

  _⟨≈˘≈˘⟩_ : {Q R S : Carrier} → R ≈ Q → S ≈ R → Q ≈ S
  _⟨≈˘≈˘⟩_ x y = ≈-trans (≈-sym x) (≈-sym y)

  _⟨≤≤⟩_ : {Q R S : Carrier} → Q ≤ R → R ≤ S → Q ≤ S
  _⟨≤≤⟩_ = ≤-trans
  _⟨≤≈⟩_ : {Q R S : Carrier} → Q ≤ R → R ≈ S → Q ≤ S
  _⟨≤≈⟩_ = ≤-trans₁
  _⟨≤≈˘⟩_ : {Q R S : Carrier} → Q ≤ R → S ≈ R → Q ≤ S
  _⟨≤≈˘⟩_ x y = ≤-trans₁ x (≈-sym y)
  _⟨≈≤⟩_ : {Q R S : Carrier} → Q ≈ R → R ≤ S → Q ≤ S
  _⟨≈≤⟩_ = ≤-trans₂
  _⟨≈˘≤⟩_ : {Q R S : Carrier} → R ≈ Q → R ≤ S → Q ≤ S
  _⟨≈˘≤⟩_ x = ≤-trans₂ (≈-sym x)
```

We add the usual heuristics for proofs in a partially ordered space.

```
  -- “indirect-inclusion, from the right, to inclusion”
  indir-≤→≤ : {x y : Carrier} → (∀ {z} → y ≤ z → x ≤ z) → x ≤ y
  indir-≤→≤ {x} {y} pf = pf ≤-refl
  -- “indirect-inclusion, from the left, to inclusion”
  ≤-indir→≤ : {x y : Carrier} → (∀ {z} → z ≤ x → z ≤ y) → x ≤ y
  ≤-indir→≤ {x} {y} pf = pf ≤-refl
  -- “indirect equality, from the right, to equality”
  indir-≤→≈ : {x y : Carrier} → (∀ {z} → y ≤ z → x ≤ z) → (∀ {z} → x ≤ z → y ≤ z) → x ≈ y
  indir-≤→≈ to fro = ≤-antisym (indir-≤→≤ to) (indir-≤→≤ fro)
    -- “indirect equality, from the left, to equality”
  ≤-indir→≈ : {x y : Carrier} → (∀ {z} → z ≤ y → z ≤ x) → (∀ {z} → z ≤ x → z ≤ y) → x ≈ y
  ≤-indir→≈ to fro = ≤-antisym ((≤-indir→≤ fro)) ((≤-indir→≤ to))
```

Of course other forms of indirect equality can be obtained by mixing the indirect inclusions.

We also add a convenient alias for Poset.Carrier, while avoiding a name clash with Relation.Binary.Setoid.Utils.⌊_⌋.

Any two ordered elements give rise to an order homomorphism from the natural order on Bool:

```
  ≤-to-𝔹→ : {R S : Carrier} → R ≤ S → (Bool → Carrier)
  ≤-to-𝔹→ {R} {S} _ b = if b then S else R
```

Occasionally the following abbreviation is useful:

```
⌊_≤⌋ : {i j k : Level} → Poset i j k → Set i
⌊_≤⌋ = Poset.Carrier
```

The following renamings do not re-export the Setoid material since in OrderedSemigroupoid, that is obtained separately — this may be organised differently in the future.

```
module Poset-round {j k₁ k₂ : Level} (poset : Poset j k₁ k₂) where
  open Poset′ poset public using () renaming
    (_≤_          to _⊆_           -- : Rel Carrier k₂
    ; Carrier     to ⊆-Carrier     -- : Setj
    ; ≤-antisym   to ⊆-antisym     -- : {R S : Carrier} → R ⊆ S → S ⊆ R → R ≈ S
    ; ≤-refl      to ⊆-refl        -- : {R : Carrier} → R ⊆ R
    ; ≤-reflexive to ⊆-reflexive   -- : {R S : Carrier} → R ≈ S → R ⊆ S
    ; ≤-trans     to ⊆-trans       -- : {Q R S : Carrier} → Q ⊆ R → R ⊆ S → Q ⊆ S
    ; ≤-reflexive′ to ⊆-reflexive′ -- : {R S : Carrier} → R ≈ S → S ⊆ R
    ; ≤-trans₁    to ⊆-trans₁      -- : {Q R S : Carrier} → Q ⊆ R → R ≈ S → Q ⊆ S
    ; ≤-trans₂    to ⊆-trans₂      -- : {Q R S : Carrier} → Q ≈ R → R ⊆ S → Q ⊆ S
    ; _⟨≤≤⟩_      to _⟨⊆⊆⟩_        -- : {Q R S : Carrier} → Q ⊆ R → R ⊆ S → Q ⊆ S
    ; _⟨≤≈⟩_      to _⟨⊆≈⟩_        -- : {Q R S : Carrier} → Q ⊆ R → R ≈ S → Q ⊆ S
    ; _⟨≤≈˘⟩_     to _⟨⊆≈˘⟩_       -- : {Q R S : Carrier} → Q ⊆ R → S ≈ R → Q ⊆ S
    ; _⟨≈≤⟩_      to _⟨≈⊆⟩_        -- : {Q R S : Carrier} → Q ≈ R → R ⊆ S → Q ⊆ S
```

```
    ; _⟨≈˘≤⟩_    to _⟨≈˘⊆⟩_     -- : {Q R S : Carrier} → R ≈ Q → R ⊆ S → Q ⊆ S
    ; indir-≤→≤  to indir-⊆→⊆   -- : {x y : Carrier} → (∀ {z} → y ≤ z → x ≤ z) → x ≤ y
    ; ≤-indir→≤  to ⊆-indir→⊆   -- : {x y : Carrier} → (∀ {z} → z ≤ x → z ≤ y) → x ≤ y
    ; indir-≤→≈  to indir-⊆→≈   -- : {x y : Carrier} → (∀ {z} → y ≤ z → x ≤ z)
                                -- → (∀ {z} → x ≤ z → y ≤ z) → x ≈ y
    ; ≤-indir→≈  to ⊆-indir→≈   -- : {x y : Carrier} → (∀ {z} → z ≤ y → z ≤ x) → (∀ {z}
                                -- → z ≤ x → z ≤ y) → x ≈ y
    )

module Poset-square {j k₁ k₂ : Level} (poset : Poset j k₁ k₂) where
  open Poset′ poset public using () renaming
    ( _≤_          to _⊑_           -- : Rel Carrier k₂
    ; Carrier      to ⊑-Carrier     -- : Set j
    ; ≤-antisym    to ⊑-antisym     -- : {R S : Carrier} → R ⊑ S → S ⊑ R → R ≈ S
    ; ≤-refl       to ⊑-refl        -- : {R : Carrier} → R ⊑ R
    ; ≤-reflexive  to ⊑-reflexive   -- : {R S : Carrier} → R ≈ S → R ⊑ S
    ; ≤-trans      to ⊑-trans       -- : {Q R S : Carrier} → Q ⊑ R → R ⊑ S → Q ⊑ S
    ; ≤-reflexive′ to ⊑-reflexive′  -- : {R S : Carrier} → R ≈ S → S ⊑ R
    ; ≤-trans₁     to ⊑-trans₁      -- : {Q R S : Carrier} → Q ⊑ R → R ≈ S → Q ⊑ S
    ; ≤-trans₂     to ⊑-trans₂      -- : {Q R S : Carrier} → Q ≈ R → R ⊑ S → Q ⊑ S
    ; _⟨≤≤⟩_       to _⟨⊑⊑⟩_        -- : {Q R S : Carrier} → Q ⊑ R → R ⊑ S → Q ⊑ S
    ; _⟨≤≈⟩_       to _⟨⊑≈⟩_        -- : {Q R S : Carrier} → Q ⊑ R → R ≈ S → Q ⊑ S
    ; _⟨≤≈˘⟩_      to _⟨⊑≈˘⟩_       -- : {Q R S : Carrier} → Q ⊑ R → S ≈ R → Q ⊑ S
    ; _⟨≈≤⟩_       to _⟨≈⊑⟩_        -- : {Q R S : Carrier} → Q ≈ R → R ⊑ S → Q ⊑ S
    ; _⟨≈˘≤⟩_      to _⟨≈˘⊑⟩_       -- : {Q R S : Carrier} → R ≈ Q → R ⊑ S → Q ⊑ S
    ; indir-≤→≤    to indir-⊑→⊑     -- : {x y : Carrier} → (∀ {z} → y ≤ z → x ≤ z) → x ≤ y
    ; ≤-indir→≤    to ⊑-indir→⊑     -- : {x y : Carrier} → (∀ {z} → z ≤ x → z ≤ y) → x ≤ y
    ; indir-≤→≈    to indir-⊑→≈     -- : {x y : Carrier} → (∀ {z} → y ≤ z → x ≤ z)
                                    -- → (∀ {z} → x ≤ z → y ≤ z) → x ≈ y
    ; ≤-indir→≈    to ⊑-indir→≈     -- : {x y : Carrier} → (∀ {z} → z ≤ y → z ≤ x) → (∀ {z}
                                    -- → z ≤ x → z ≤ y) → x ≈ y
    )
```

2.2 Relation.Binary.Poset.Constructions

Two simple poset constructions, both starting from a parameter poset P:

```
module _ {i j k : Level} (P : Poset i j k) where
  open Poset′ P renaming (Carrier to P₀)
```

Functions to the carrier of P for a poset, where order is defined point-wise:

```
  pointwisePoset : {ℓ : Level} (A : Set ℓ) → Poset (i ⊍ ℓ) (j ⊍ ℓ) (k ⊍ ℓ)
  pointwisePoset A = record
    {Carrier        = A → P₀
    ; _≈_           = λ f g → ({x : A} → f x ≈ g x)
    ; _≤_           = λ f g → ({x : A} → f x ≤ g x)
    ; isPartialOrder = record
      {isPreorder   = record
        {isEquivalence = record
          {refl     = ≈-refl
          ; sym     = λ f≈g      → ≈-sym     f≈g
          ; trans   = λ f≈g g≈h  → ≈-trans   f≈g g≈h
          }
        ; reflexive = λ f≈g      → ≤-reflexive f≈g
        ; trans     = λ f≤g g≤h  → ≤-trans   f≤g g≤h
        }
      ; antisym     = λ f≤g g≤f  → ≤-antisym f≤g g≤f
      }
    }
```

Given a predicate S on the carrier P_0, we obtain a sub-poset of P with carrier $\{x : P_0 \bullet (x, S\,x)\}$ and order $(x, sx) \le (y, sy) :\equiv x \le y$.

```
  subPoset : {l : Level} (S : P₀ → Set l) → Poset (l ⊍ i) j k
  subPoset S = record
    {Carrier = Σ x : P₀ • S x
     ; _≈_ = _≈_ on proj₁
     ; _≤_ = _≤_ on proj₁
     ; isPartialOrder = record
       {isPreorder = record
         {isEquivalence = record {refl = ≈-refl; sym = ≈-sym; trans = ≈-trans}
         ; reflexive = ≤-reflexive; trans = ≤-trans
         }
       ; antisym = ≤-antisym
       }
    }
```

2.3 Relation.Binary.Poset.Junctivity

Let us recall the notion of order-preserving mappings and some related properties.

```
open PosetMeetI using (IsMeetI; module IsMeetI; ≤-to-IsMeetI)
open PosetJoinI using (IsJoinI; module IsJoinI)
```

Let consider an arbitrary pair of posets, and a mapping between their carriers,

```
module _ {i j k i′ j′ k′}
      (A : Poset i j k)
      (B : Poset i′ j′ k′)
      (let open Poset′ A renaming (Carrier to A₀))
      (let open Poset-square B renaming (⊑-Carrier to B₀))
      (f : A₀ → B₀)
  where
  private
    module A = Poset′ A
    module B = Poset′ B
```

Then we formalize that f is an order preserving mapping as

```
  IsMonotone : Set (k′ ⊍ (k ⊍ i))
  IsMonotone = {x y : A₀} → x ≤ y → f x ⊑ f y
```

If f is monotone then for any g: $f\,(\sqcap\,g) \sqsubseteq \sqcap\,(f \circ g) \sqsubseteq \sqcup\,(f \circ g) \sqsubseteq f\,(\sqcup\,g)$

```
  monotone→⊓-bound : IsMonotone
                   → {gℓ : Level} {I : Set gℓ} {g : I → A₀} {m : A₀} {m′ : B₀}
                   → IsMeetI A g m → IsMeetI B (f ∘ g) m′ → f m ⊑ m′
  monotone→⊓-bound f-mono {m = m} {m′ = m′} ⊓g ⊓fg =
    IsMeetI.universal ⊓fg (λ x → f-mono (IsMeetI.bound ⊓g x))

  monotone→⊔-bound : IsMonotone
    → {gℓ : Level} {I : Set gℓ} {g : I → A₀} {m : A₀} {m′ : B₀}
    → IsJoinI A g m → IsJoinI B (f ∘ g) m′ → m′ ⊑ f m
  monotone→⊔-bound f-mono {m = m} {m′ = m′} ⊔g ⊔fg =
    IsMeetI.universal ⊔fg ((λ x → f-mono ((IsMeetI.bound ⊔g x))))
```

Finally, order isomorphisms are existentially junctive.

```
module order-isos-are-junctive {i j k i′ j′ k′} (A : Poset i j k) (B : Poset i′ j′ k′)
    (let open Poset′ A renaming (Carrier to A₀))
    (let open Poset-square B renaming (⊑-Carrier to B₀))
    (let open SetoidB (posetSetoid B) using (_≈B_))
    (f : A₀ → B₀) (f-monotone : {x y : A₀} → x ≤ y → f x ⊑ f y)
    (f-isotone : {x y : A₀} → f x ⊑ f y → x ≤ y)
    (f-surj : {y : B₀} → Σ x : A₀ • y ≈B f x)
    {gℓ : Level} {I : Set gℓ} {g : I → A₀} {m : A₀}
  where
  ⊓-junctive : IsMeetI A g m → IsMeetI B (f ∘ g) (f m)
  ⊓-junctive ⊓g = let open IsMeetI A ⊓g in record
    {bound = f-monotone ∘ bound
    ; universal = λ {y} y⊑fI → proj₂ f-surj ⟨≈⊑⟩
       (f-monotone ∘ universal) (λ x → f-isotone (proj₂ f-surj ⟨≈˘⊑⟩ y⊑fI x))}
  ⊔-junctive : IsJoinI A g m → IsJoinI B (f ∘ g) (f m)
  ⊔-junctive ⊔I = let open IsMeetI (dualPoset A) ⊔I in record
    {bound = f-monotone ∘ bound
    ; universal = λ {y} fI⊑y →
       (f-monotone ∘ universal) (λ x → f-isotone (fI⊑y x ⟨⊑≈⟩ proj₂ f-surj))
         ⟨⊑≈˘⟩ proj₂ f-surj
    }
```

2.4 Relation.Binary.Poset.Galois

Let us turn to the notion of (monotone) Galois connections, as popularised by Backhouse *et al.* (Aarts et al., 1992). We shall not dwell in great length on this pointwise, or external, presentation as it has already been done informally. Our contribution here is its mechanisation within Agda, which we then employ in the internal setting, e.g., Sect. 5.1.3.

2.4.1 Ubiquity of Galois Connections

The concept of Galois connections presents its importance as a tool when specifying definitions and as an interface for the derivation of further results. This section is devoted to demonstrate the concept's use as a specification tool and then exhibiting a variety of scenarios in which the concept arises.

Specification by Galois Connection

Galois connections may be used as a tool for *specifying* complicated notions by associating their properties with those of simpler notions. Since such a definition is expressed via inclusions, an equality is usually proved by appealing to the rule of indirect equality; hence, posets, and not preorders, are needed. Let us give a few examples to demonstrate this idea.

Integer Division: Rather than an overly-detailed implementation, we have an eloquent specification:

$$k \le m \div n \equiv k \times n \le m$$

A connection is used to specify the *difficult* notion of integer division with the *simpler* notion of multiplication. Now properties of the former correspond of those of the latter; e.g., $(n \div m) \div d = n \div (d \times m)$ since multiplication is associative.

Floor: The complicated notion of integer-rounding downwards, i.e., the floor function, may be specified by the embedding of the integers into the reals:

$$\forall\, n : \mathbb{Z}; x : \mathbb{R} \bullet n \le \lfloor x \rfloor \equiv n \le x$$

— dually, $\lceil x \rceil \le n \equiv x \le n$. It can then be shown that $\lfloor x \rfloor = n \equiv n \le x < n + 1$; a condition easier for verification of candidates but less amiable to calculation.

Take: The informally specified operation $\mathsf{take}\ m\ [x_1, \ldots, x_n] = [x_1, \ldots, x_m]$, for $m \le n$, can be formally specified as

$$ys \sqsubseteq \mathsf{take}\ (n, xs) \equiv \mathsf{length}\ ys \le n \wedge ys \sqsubseteq xs$$

Where $\_\sqsubseteq\_$ denotes prefix relation; and the right hand side is essentially a poset product space. With a recursive definition, properties would be proved with induction, but with the connection, properties of take would be proved, generally more elegantly, by indirect equality.

Supremum: The notion of "greatest upper bound" is a bit difficult to say, let alone comprehend. However, the concept of a constant, or diagonal, function is trivial: $\mathsf{K}\ x\ y = x$. So we work with the complex by shifting to the manageable:

$$\sqcup \dashv \mathsf{K} \text{ that is, } \forall\, z \bullet \sqcup f \le z \equiv f \mathrel{\dot{\le}} \mathsf{K}\ z$$

Note that the connection is between a poset and its pointwise extension on functions.

- The notion of suprema is traditionally formulated by sets and formulated awkwardly as: $\bigsqcup S = s$ precisely when s is an upper bound of S and the least such bound. However, as types and functions are more the primitive, for us, it only seems reasonable that we discuss extrema in such terms. As the notion of subset does not correspond nicely to sub-type, functions seem more appropriate. Indeed, every set S corresponds to a function, the identity on S — the categorical imperative! Hence, we phrase extrema via functions. Additionally the connection formulation is not only more calculation-friendly and succinct, but is in-fact equivalent; the reader would do well to observe this.
  In Agda we write $\mathsf{IsJoinI}\ A\ f\ j$ precisely when $\sqcup f = j$ in poset A; the suffix I to remind us these are *I*ndexed joins — dually for meets —, but will use the 'awkward formulation' as an equivalence is represented in Agda as a pair of implications — in the internal setting, equivalences become equations.
- Hence, lattices are posets where certain adjoints exist! Whence, (complete) lattices can be made internal via internal Galois connections.
- **Binary Joins:** Consider the mapping $f = (0 \to x \mid 1 \to y)$, from the two-point space into our poset, then this yields the notion of *binary joins*. Then, e.g., for a linear order we can specify the complicated notion of binary maximum as $\mathsf{max}\ (x, y) = \sqcup f$; or unfolding the connection:

  $$\forall\, x, y, z \bullet \mathsf{max}\ (x, y) \le z \equiv x \le z \wedge y \le z$$

  Likewise, the complicated notion of "greatest common divisor" is rendered trivial, via the constant function, as $\mathsf{gcd}\ (x, y) = \sqcup f$ — where the order is divisibility, of course.
- **Top and Bottom:** A poset P is bounded above by $\top$ and below by $\bot$ if and only if

  $$\langle\, \{1\} \to \{\bot\} \hookrightarrow P \,\rangle \dashv \langle\, P \to \{1\} \,\rangle \dashv \langle\, \{1\} \to \{\top\} \hookrightarrow P \,\rangle$$

  Exercise: which function f in the supremum characterization is associated with $\bot$?
- **Colimits:** Interpreting the inclusions as certain Hom functors and equivalence $\_\equiv\_$ as isomorphism $\_\cong\_$ yields the notion that "functor f has colimit $\sqcup f$".

- **Power:** In the setting of sets, union is adjoint to power,

$$\bigcup \mathcal{F} \subseteq X \Leftrightarrow \mathcal{F} \subseteq \mathbb{P} X$$

Compare this with the supremum characterization.
For those interested in allegories, exercise: a power allegory that has a lower adjoint for the power functor is necessarily complete?

Examples of Galois Connections

Let us observe a quick enumeration of common connections.

- **Residuals:** Given two relations R and S, we may form their 'residual' relations

$$x\,(R \setminus S)\,y \equiv (\forall z \bullet z\,R\,x \Rightarrow z\,S\,y) \text{ and } x\,(R \,/\, S)\,y \equiv (\forall z \bullet x\,R\,z \Rightarrow y\,S\,z)$$

Then, we find
$$\forall\, Q, R, S \bullet Q \sqsubseteq R \setminus S \Leftrightarrow R \fatsemi Q \sqsubseteq S \Leftrightarrow R \sqsubseteq S \,/\, Q$$
where
$$x\,(R \fatsemi S)\,y \equiv (\exists z \bullet x\,R\,z \wedge z\,S\,y) \text{ and } R \sqsubseteq S \equiv (\forall x, y \bullet x\,R\,y \Rightarrow x\,S\,y)$$
Then, we find three connections:

$$(R \fatsemi) \dashv (R \setminus) \text{ and } (S\,/) \dashv\breve{}\ (S \setminus) \text{ and } (\fatsemi Q) \dashv (/\,Q)$$

where we write $\dashv\breve{}$ to denote an *antitone* Galois connection; i.e.,

$$f \dashv\breve{}\ g \equiv (\forall x, y \bullet x \leq f\,y \equiv y \leq g\,x)$$

Note: $(f, \leq) \dashv\breve{}\ (g, \sqsubseteq) \equiv (f, \leq\breve{}\,) \dashv (g, \sqsubseteq)$.
See Chapter 3 for more on residuals.

  - Integer division: $k \leq m \div n \equiv k \times n \leq m$.

    Residuals are to relation composition, as integer division is to multiplication.

    Notice that, in the domain of natural numbers, the fact that division by zero is undefined becomes more explicit with this presentation. Indeed, taking $n = 0$ and noting that $k \times n = k \times 0 = 0$ along with that all naturals are at least 0, we find that the *specification* of the element $m \div n$ must satisfy

    $$\forall\, k : \mathbb{N} \bullet k \leq m \div 0$$

    This is tantamount to saying $m \div 0$ is "the largest natural number", which does not exist.
  - Polars: $R\uparrow \dashv\breve{}\ R\downarrow$ where

    $$R\uparrow(A) = \text{"the } R\text{-successors of all of } A\text{"} = \{s \mid A\,(\in \setminus R)\,s\}$$

    and likewise $R\downarrow = (R\,\breve{}\,)\uparrow$. Alternatively, if we construe R as a relating objects to their properties, then $R\uparrow(A)$ = "the properties common to all objects A" and $R\downarrow(B)$ = "the objects satisfying all properties B".
  - The original Galois connection can naively be seen as the polars between functions and elements induced by relation $f\,R\,x \equiv f\,x = 0$.
  - Syntax is adjoint to semantics: $(\vDash\uparrow) \dashv\breve{}\,(\vDash\downarrow)$, where for a given signature, a sentence s and an algebra A over that signature, the 'true of (interpretation)' relation $\vDash$ is defined: $A \vDash s$ if and only if s is true when interpreted in model / algebra A.

    Exercise: what is the relation between $(\vDash\downarrow) \fatsemi (\vDash\uparrow)$ and the notion of 'logical consequence'?
  - Connections between powersets: $f \dashv\breve{}\ g$, between powersets, if and only if $\langle f, g\rangle = \langle R\uparrow, R\downarrow\rangle$ where $x\,R\,y \Leftrightarrow x \in f\{y\}$. Exercise: $f \dashv g$, between powersets, if and only if what?
  - Kan Extensions — for the categorically inclined, Hinze (2012).
- **Hoare Triples:** For relation R, take

$$R\!\rightarrow(A) = \{s \mid (\exists\, a \mid a \in A \bullet a\,R\,s)\} = \text{"the set of successors of } \textit{some} \text{ of } A\text{"}$$

cf $R\uparrow$, and, dually, define $R\!\leftarrow\, = (R\,\breve{}\,)\!\rightarrow$. Then it can be shown that

$$R\!\rightarrow\, \dashv (R\!\leftarrow)^{*} \text{ where } f^{*}(x) = \neg\,(f\,(\neg\,x))$$

In particular, with respect to total correctness, we have that $\{P\}\ S\ \{Q\} \equiv S\!\rightarrow(P) \subseteq Q$ and, it is usually written that, $(S\!\leftarrow)^{*}(Q) = wp.S.Q$. Then the connection takes the particular shape,

$$\{P\}\ S\ \{Q\} \equiv P \subseteq wp.S.Q$$

Note that if we lift the target of S by adding a bottom element, representing non-termination, then the result is not wp but rather wlp, the weakest liberate predicate.
- **Free vs. Forgetful:** for a fixed mathematical structure X, let L (S) be the substructure of X generated by the set S, and let U (S) be the underlying set of structure S. Then, $L \dashv U$.

2.4.2 Definition

```
open PosetMeetI using (IsMeetI; module IsMeetI)
open PosetJoinI using (IsJoinI; module IsJoinI)
```

First, a local abbreviation for obtaining the carrier of a poset:

```
private
  _₀ : {i j k : Level} → (A : Poset i j k) → Set i
  A ₀ = Poset.Carrier A
```

A *Galois connection* between a poset $A = (A_0, \leq)$ and a poset $B = (B_0, \sqsubseteq)$ consists of a pair of mappings L, U between the carriers such that

$$\forall x, y \bullet L\,x \sqsubseteq y \equiv x \leq U\,y$$

This equivalence is formalized as a pair of implications — while in the internal setting it will become a single equation. The mappings L and U are referred to as the Lower and Upper 'adjoints', respectively, and the connection is denoted $L \dashv U$.

```
record IsGC {i j k i′ j′ k′ : Level} (A : Poset i j k) (B : Poset i′ j′ k′) (L : A ₀ → B ₀) (U : B ₀ → A ₀)
  : Set (i ⊍ j ⊍ k ⊍ i′ ⊍ j′ ⊍ k′) where
  A₁ = posetSetoid A; B₁ = posetSetoid B
  open SetoidA A₁ hiding (A₀); open SetoidB B₁ hiding (B₀)
  open Poset′ A renaming (Carrier to A₀); open Poset-square B renaming (⊑-Carrier to B₀)
  field
    gc : {x : A₀} {y : B₀} → L x ⊑ y → x ≤ U y
    gc˘ : {x : A₀} {y : B₀} → x ≤ U y → L x ⊑ y
```

The induced point-wise poset induces a Galois connection.

```
infix 5 ≤̇ ⊑̇

≤̇ : ∀ {l} {Q : Set l} → (f g : Q → A0) → Set (l ⊍ k)
≤̇ {Q = Q} = Poset′._≤_ (pointwisePoset A Q)

⊑̇ : ∀ {l} {Q : Set l} → (f g : Q → B0) → Set (l ⊍ k′)
⊑̇ {Q = Q} = Poset′._≤_ (pointwisePoset B Q)
isgc-functional : {l : Level} {Q : Set l}
  → IsGC (pointwisePoset A Q) (pointwisePoset B Q) (λ f → L ∘ f) (λ g → U ∘ g)
isgc-functional {l} {Q} = record
  {gc = λ {f} {g} Lf⊑g {x} → gc Lf⊑g; gc˘ = λ {f} {g} f≤Ug {x} → gc˘ f≤Ug}
```

Taking f, g both as the identity yields the converse result.

Before we move on, let us note that there is an equivalent reformulation: a Galois connection is precisely a monotonic pair of maps with one composition being increasing and the other composition being decreasing. For certain mappings, it may be easier to prove the pieces independently than it is to prove the universal characterization.

```
piecewise-to-gc : IsMonotone A B L → IsMonotone B A U → id ≤̇ U ∘ L → L ∘ U ⊑̇ id → IsGC A B L U
piecewise-to-gc L-mon U-mon id≤UL LU⊑id = record
  {gc = λ Lx⊑y → id≤UL ⟨≤≤⟩ U-mon Lx⊑y; gc˘ = λ x≤Uy → L-mon x≤Uy ⟨⊑⊑⟩ LU⊑id}
```

Where (f ≤̇ g) ≡ (∀ {x} → f x ≤ g x) and likewise for ⊑̇.

While the two variable quantification in the characterization can naively be checked by a quadratic-time algorithm, this piecewise definition would take linear-time.

Furthermore, this concept is also somewhat symmetric: ⟨L, ≤⟩ ⊣ ⟨U, ⊑⟩ ≡ ⟨U, ⊒⟩ ⊣ ⟨L, ≥⟩.

```
IsGC-dual : IsGC (dualPoset B) (dualPoset A) U L
IsGC-dual = record {gc = gc˘; gc˘ = gc}
```

2.4.3 Fundamental Properties

Let us recall those properties that are immediate from the connection and are some of the most used. The adjoints yield a pair of 'cancellation' laws, necessarily preserve equivalence and order, and are each other's 'semi-inverse'.

Let us denote the equality on A by _≈A_ and likewise for poset B. Then,

```
≤-can : {x : A0} → x ≤ U (L x)
≤-can = gc ⊑-refl

⊑-can : {y : B0} → L (U y) ⊑ y
⊑-can = gc˘ ≤-refl

U-cong : {a a′ : B0} → a ≈B a′ → U a ≈A U a′
U-cong a≈a′ = ≤-antisym (gc (⊑-can ⟨⊑≈⟩ a≈a′)) (gc (⊑-can ⟨⊑≈˘⟩ a≈a′))

L-cong : ∀ {a a′ : A0} → a ≈A a′ → L a ≈B L a′
L-cong a≈a′ = ⊑-antisym (gc˘ (a≈a′ ⟨≈≤⟩ ≤-can)) (gc˘ (a≈a′ ⟨≈˘≤⟩ ≤-can))

L-monotone : ∀ {x y} → x ≤ y → L x ⊑ L y
L-monotone {x} {y} x≤y = gc˘ (≤-trans x≤y ≤-can)

U-monotone : ∀ {x y} → x ⊑ y → U x ≤ U y
U-monotone {x} {y} x⊑y = ≤-indir→≤ (λ {z} z≤Ux → gc (gc˘ z≤Ux ⟨⊑⊑⟩ x⊑y))

L-semi-inverse : ∀ {x} → L (U (L x)) ≈B L x
L-semi-inverse {x} = indir-⊑→≈
  (λ {z} Lx⊑z → gc˘ (U-monotone Lx⊑z)) (λ {z} LULx⊑z → L-monotone ≤-can ⟨⊑⊑⟩ LULx⊑z)
U-semi-inverse : ∀ {x} → (U ∘ L ∘ U) x ≈A U x
U-semi-inverse {x} = ≤-indir→≈ (λ pf → gc (L-monotone pf)) (λ pf → pf ⟨≤≤⟩ U-monotone ⊑-can)
```

2.4.4 Properties of the Lower Adjoint

Let us turn to proving properties for the lower adjoint only. Then we dualize to obtain the properties for the upper adjoint.

```
module L-Props {i j k i′ j′ k′} {A : Poset i j k} {B : Poset i′ j′ k′}
  (let open Poset′ A renaming (Carrier to A0))
  (let open Poset-square B renaming (⊑-Carrier to B0))
  {L : A0 → B0} {U : B0 → A0} (isgc : IsGC A B L U)
  where
  open IsGC isgc; open SetoidA A1 hiding (A0); open SetoidB B1 hiding (B0)
```

It is well known that each adjoint determines the other uniquely, they satisfy an 'absorption law', elimination and interchange laws, and 'image isotonicty': each adjoint is isotonic on the image of the other adjoint.

```
adjoint-uniq-U→L : {L′ : A0 → B0} {U′ : B0 → A0} (isgc′ : IsGC A B L′ U′)
                  → (∀ {x} → U x ≈A U′ x) → (∀ {x} → L x ≈B L′ x)
adjoint-uniq-U→L {L′} {U′} isgc′ U≈U′ = λ {x} → indir-⊑→≈
  (λ {z} L′x⊑z → let x≤U′x = gc′ L′x⊑z in gc˘ (x≤U′x ⟨≤≈˘⟩ U≈U′))
  (λ {z} Lx⊑z → let x≤Uz = gc Lx⊑z in gc˘′ (x≤Uz ⟨≤≈⟩ U≈U′))
  where open IsGC isgc′ renaming (gc to gc′; gc˘ to gc˘′)
L-absorption : ∀ {x y} → U (L x) ≈A U (L y) → L x ≈B L y
L-absorption {x} {y} ULx≈ULy = indir-⊑→≈
  (λ {z} Ly⊑z → L-semi-inverse ⟨≈˘⊑⟩ gc˘ (ULx≈ULy ⟨≈≤⟩ U-monotone Ly⊑z))
  (λ {z} Lx⊑z → L-semi-inverse ⟨≈˘⊑⟩ gc˘ (ULx≈ULy ⟨≈˘≤⟩ U-monotone Lx⊑z))
L-elim : ∀ {x y} → L y ⊑ L (U x) → y ≤ U x
L-elim L⊑LU = gc L⊑LU ⟨≤≈⟩ U-semi-inverse
L-U-interchange : ∀ {x y} → L y ⊑ L (U x) → U (L y) ≤ U x
L-U-interchange L⊑LU = U-monotone L⊑LU ⟨≤≈⟩ U-semi-inverse
L-isotone-on-U : ∀ {x y} → L (U x) ⊑ L (U y) → U x ≤ U y
L-isotone-on-U = L-elim
```

Junctivity

Adjoints are existentially ⊔/⊓-junctive, i.e., extrema preserving, between the images of the adjoints — recall that extrema, namely IsJoinI, were discussed in Sect. 2.4.1.

```
L-⊔-junctive-on-U : {gℓ : Level} {I : Set gℓ} {g : I → B0} {m : A0}
  → IsJoinI A (U ∘ g) m → IsJoinI B (L ∘ U ∘ g) (L m)
L-⊔-junctive-on-U Ug-join = record
  {bound = L-monotone ∘ bound
  ; universal = λ {y} LUg⊑y → (gc˘ ∘ universal) (gc ∘ LUg⊑y)
  }
  where open IsJoinI A Ug-join
```

For the other junctivitiy result, let us formalize the subposets of the adjoint images; and construct LL as the restriction of the mapping L to the these image subposets.

```
L-poset : Poset (j′ ⊍ i ⊍ i′) j′ k′
L-poset = subPoset B (λ y → Σ x : A0 • L x ≈B y)
```

```
U-poset : Poset (i′ ⊍ j ⊍ i) j k
U-poset = subPoset A (λ y → Σ x : B₀ • U x ≈A y)

LL : U-poset ₀ → L-poset ₀
LL e, e∈U = L e, e, ≈B-refl where e = proj₁ e, e∈U
```

Then, $\mathsf{L}\langle \sqcap\ \mathsf{y} \bullet \mathsf{U}\ \mathsf{y}\rangle = \langle \sqcap\ \mathsf{y} \bullet \mathsf{L}\ (\mathsf{u}\ \mathsf{y})\ \rangle$ is proved by witnessing that LL is an order isomorphism and hence junctive. Formally,

```
L-⊓-junctive-on-U-poset : {gℓ : Level} {I : Set gℓ} {g : I → U-poset ₀} {m : U-poset ₀}
  → IsMeetI U-poset g m → IsMeetI L-poset (LL ∘ g) (LL m)
L-⊓-junctive-on-U-poset = ⊓-junctive
  where
    open order-isos-are-junctive {i′ ⊍ j ⊍ i} {j} {k} {j′ ⊍ i ⊍ i′} {j′} {k′} U-poset L-poset LL L-monotone
        -- Proving {e, e∈U d, d∈U : U-poset ₀} → L e ⊑ L d → e ≤ d
      (λ {e, e∈U} {d, d∈U} Le⊑Ld
      → let open PosetCalc A
            e   = proj₁ e, e∈U
            e₀  = proj₁ (proj₂ e, e∈U)
            Ue₀≈e = proj₂ (proj₂ e, e∈U)
            d   = proj₁ d, d∈U
            d₀  = proj₁ (proj₂ d, d∈U)
            Ud₀≈d = proj₂ (proj₂ d, d∈U)
        in
        ≤-begin
          e
        ≈˘⟨ Ue₀≈e ⟩
          U e₀
        ≈˘⟨ U-semi-inverse ⟩
          U (L (U e₀))
        ≈⟨ U-cong (L-cong Ue₀≈e) ⟩
          U (L e)
        ≤⟨ U-monotone Le⊑Ld ⟩
          U (L d)
        ≈˘⟨ U-cong (L-cong Ud₀≈d) ⟩
          (U ∘ L ∘ U) d₀
        ≈⟨ U-semi-inverse ⟩
          U d₀
        ≈⟨ Ud₀≈d ⟩
          d
        □)
      (  -- Proving {y : L-poset ₀} → Σ x : U-poset ₀ • (proj₁ y) ≈B L (proj₁ x)
      λ {e, e∈L} →
      let open PosetCalc B
            e = proj₁ e, e∈L; e₀ = proj₁ (proj₂ e, e∈L); Le₀≈e = proj₂ (proj₂ e, e∈L)
      in ((U e), (e, ≈A-refl)), (
            ≈-begin
              e
            ≈˘⟨ Le₀≈e ⟩
              L e₀
            ≈˘⟨ L-semi-inverse ⟩
              (L ∘ U ∘ L) e₀
            ≈⟨ L-cong (U-cong Le₀≈e) ⟩
              L (U e)
            □)
      )
```

More generally: L is existentially ⊔-junctive, and U is existentially ⊓-junctive.

```
L-⊔-junctive : {gℓ : Level} {I : Set gℓ} {g : I → A₀} {m : A₀}
  → IsJoinI A g m → IsJoinI B (L ∘ g) (L m)
L-⊔-junctive {m} {g} ⊔f = let open IsJoinI A ⊔f in record
  {bound = L-monotone ∘ bound
  ; universal = λ {y} Lg⊑y → (gc˘ ∘ universal) (gc ∘ Lg⊑y)
  }
```

Interdefiniability

The adjoints determine one another as extrema of the others image.

```
_≤U⁻¹ : (x : A₀) → Poset (k ⊍ i′) j′ k′
x ≤U⁻¹ = subPoset B (λ y → x ≤ U y)

L-as-⊓ : {x : A₀} → IsMeetI (x ≤U⁻¹) (λ e → e) (L x, ≤-can)
L-as-⊓ {x} = record {bound = gc˘ ∘ proj₂; universal = λ {y, x≤Uy} y⊑id → y⊑id (L x, ≤-can)}
```

That is, $\forall\ \mathsf{x} \bullet \mathsf{L}\ \mathsf{x} = \sqcap\ \{\mathsf{y} \mid \mathsf{x} \le \mathsf{U}\ \mathsf{y}\}$. Exercise, fill in the blanks: $\forall\ \mathsf{y} \bullet \mathsf{U}\ \mathsf{x} = \_\ \{\mathsf{x} \mid \_\}$.

Induced (Co)closure Operators

Every Galois connection gives rise to a (co)closure operator: an order preserving function that is (co)increasing and idempotent.

```
LU-idemp : ∀ {x} → (L ∘ U ∘ L ∘ U) x ≈B (L ∘ U) x
LU-idemp = L-cong U-semi-inverse

LU-interior : ∀ {x y} → (L ∘ U) x ⊑ (L ∘ U) y → (L ∘ U) x ⊑ y
LU-interior = gc˘ ∘ L-elim

LU-monotone : IsMonotone B B (L ∘ U)
LU-monotone = L-monotone ∘ U-monotone

LU-cong : ∀ {x y} → x ≈B y → (L ∘ U) x ≈B (L ∘ U) y
LU-cong = L-cong ∘ U-cong
```

Closed Elements

The image of the lower (resp. upper) adjoint is precicely the open (resp. closed) elements.

```
closure≈L-image : {e : B₀} → L (U e) ≈B e → Σ a : A₀ • L a ≈B e
closure≈L-image {e} LUe≈e = (U e), LUe≈e

closure≈L-image˘ : {e : B₀} → Σ a : A₀ • L a ≈B e → L (U e) ≈B e
closure≈L-image˘ {e} (a, La≈e) = let open PosetCalc B in
  ≈-begin
    L (U e)
  ≈⟨ L-cong (U-cong (≈B-sym La≈e)) ⟩
    L (U (L a))
  ≈⟨ L-semi-inverse ⟩
    L a
  ≈⟨ La≈e ⟩
    e
  □

  -- Weaker assertions
⊑-closure≈L-image : {e : B₀} → e ⊑ L (U e) → Σ a : A₀ • L a ≈B e
⊑-closure≈L-image {e} e⊑LUe = closure≈L-image (⊑-antisym ⊑-can e⊑LUe)

⊑-closure≈L-image˘ : {e : B₀} → Σ a : A₀ • L a ≈B e → e ⊑ L (U e)
⊑-closure≈L-image˘ pf = ⊑-refl ⟨⊑≈˘⟩ closure≈L-image˘ pf
```

Perfect Connections

The connection is said to be 'perfect' if all the elements are (co)closed; (Aarts et al., 1992). The notion of perfection has many an equivalent formulation.

```
perfect≈L-injective : ({x : A₀} → U (L x) ≈A x) → ({x y : A₀} → L x ≈B L y → x ≈A y)
perfect≈L-injective per {x} {y} Lx≈Ly = let open PosetCalc A in
  ≈-begin
    x
  ≈˘⟨ per ⟩
    U (L x)
  ≈⟨ U-cong Lx≈Ly ⟩
    U (L y)
  ≈⟨ per ⟩
    y
  □
perfect≈L-injective˘ : ({x y : A₀} → L x ≈B L y → x ≈A y) → ({x : A₀} → U (L x) ≈A x)
perfect≈L-injective˘ L-inj = L-inj L-semi-inverse
perfect≈L-isotonic : ({x : A₀} → U (L x) ≈A x) → ({x y : A₀} → L x ⊑ L y → x ≤ y)
perfect≈L-isotonic per {x} {y} Lx⊑Ly = gc Lx⊑Ly ⟨≤≈⟩ per
perfect≈L-isotonic˘ : ({x y : A₀} → L x ⊑ L y → x ≤ y) → ({x : A₀} → U (L x) ≈A x)
perfect≈L-isotonic˘ L-iso {x} = ≤-antisym (L-iso (L-semi-inverse ⟨≈⊑⟩ ⊑-refl)) ≤-can
perfect≈L-surjective : ({e : B₀} → Σ a : A₀ • L a ≈B e) → ({e : B₀} → L (U e) ≈B e)
perfect≈L-surjective L-surj = λ {e} → closure≈L-image˘ L-surj
perfect≈L-surjective˘ : ({e : B₀} → L (U e) ≈B e) → ({e : B₀} → Σ a : A₀ • L a ≈B e)
perfect≈L-surjective˘ per = λ {e} → U e, per
```

2.4.5 Properties of the Upper Adjoint, by Duality

We placed the simplest properties into the record, then focused on one adjoint and now we dualize to obtain the results for the other adjoint — annotating the relevant type information.

```
module U-Props {i j k i′ j′ k′} {A : Poset i j k} {B : Poset i′ j′ k′}
  (let open Poset′ A renaming (Carrier to A₀))
  (let open Poset-square B renaming (⊑-Carrier to B₀))
  {L : A₀ → B₀} {U : B₀ → A₀} (isgc : IsGC A B L U)
  where
  open L-Props (IsGC.IsGC-dual isgc) public using () renaming
    (    adjoint-uniq-U→L to adjoint-uniq-L→U
           -- : ∀ {L′ U′} → IsGC A B L′ U′ → (∀ {x} → L x ≈B L′ x) → (∀ {x} → U x ≈A U′ x)
      ; L-absorption      to U-absorption
           -- : ∀ {x y} → L (U x) ≈B L (U y) → U x ≈B U y
      ; L-elim            to U-elim
           -- : {x : A₀} {y : B₀} → U (L x) ≤ U y → L x ⊑ y
      ; L-U-interchange to U-L-interchange
           -- : {x : A₀} {y : B₀} → U (L x) ≤ U y → L x ⊑ L (U y)
      ; L-isotone-on-U   to U-isotone-on-L
           -- : {x y : A₀} → U (L y) ≤ U (L x) → L y ⊑ L x
      ; LL      to UU        -- : U-poset ₀ → L-poset ₀
      ; _≤U⁻¹ to L⁻¹⊑_    -- = λ y → subPoset A (λ x → L x ⊑ y)
      ; L-as-⊓ to U-as-⊔    -- : {x : B₀} → IsMeetI (x ≤U⁻¹) id (U x, ≤-can (IsGC.IsGC-dual isgc))
      ; L-⊔-junctive-on-U to U-⊓-junctive-on-L
           -- : ∀ {g m} → IsJoinI (dualPoset B) (L ∘ g) m → IsJoinI (dualPoset A) (U ∘ L ∘ g) (U m)
      ; L-⊓-junctive-on-U-poset to U-⊔-junctive-on-L-poset
           -- : ∀ {g m} → IsJoinI L-poset g m → IsJoinI U-poset (UU ∘ g) (UU m)
      ; L-⊔-junctive              to U-⊓-junctive
           -- : ∀ {g m} → IsJoinI (dualPoset B) g m → IsJoinI (dualPoset A) (U ∘ g) (U m)
      ; LU-idemp to UL-idemp              -- : {x : A₀} → U (L (U (L x))) ≈ U (L x)
      ; LU-interior to UL-closure         -- : {x y : A₀} → U (L y) ≤ U (L x) → y ≤ U (L x)
      ; LU-monotone to UL-monotone    -- : {x y : A₀} → y ≤ x → U (L y) ≤ U (L x)
      ; LU-cong   to UL-cong               -- : {x y : A₀} → x ≈ y → U (L x) ≈ U (L y)
      ; closure≈L-image      to closure≈U-image
           -- : {e : A₀} → U (L e) ≈ e → Σ a : B₀ • U a ≈A e
      ; closure≈L-image˘     to closure≈U-image˘
           -- : {e : A₀} → Σ a : B₀ • U a ≈ e → U (L e) ≈A e
      ; ⊑-closure≈L-image   to ≤-closure≈U-image
           -- : {e : A₀} → U (L e) ≤ e → Σ a : B₀ • U a ≈A e
      ; ⊑-closure≈L-image˘  to ≤-closure≈U-image˘
           -- : {e : A₀} → Σ a : B₀ • U a ≈A e → U (L e) ≤ e
      ; perfect≈L-injective   to perfect≈U-injective
           -- : ({x : B₀} → L (U x) ≈B x) → ({x y : B₀} → U x ≈A U y → x ≈B y)
      ; perfect≈L-injective˘  to perfect≈U-injective˘
           -- : ({x y : B₀} → U x ≈A U y → x ≈B y) → ({x : B₀} → L (U x) ≈B x)
      ; perfect≈L-isotonic    to perfect≈U-isotonic
           -- : ({y : B₀} → L (U y) ≈B y) → ({x y : B₀} → U x ≤ U y → x ⊑ y)
      ; perfect≈L-isotonic˘   to perfect≈U-isotonic˘
           -- : ({x y : B₀} → U x ≤ U y → x ⊑ y) → ({y : B₀} → L (U y) ≈B y)
      ; perfect≈L-surjective to perfect≈U-surjective
           -- : ({e : A₀} → Σ a : B₀ • U a ≈A e) → ({e : A₀} → U (L e) ≈A e)
      ; perfect≈L-surjective˘ to perfect≈U-surjective˘
           -- : ({e : A₀} → U (L e) ≈A e) → ({e : A₀} → Σ a : B₀ • U a ≈A e)
    )
```

2.4.6 Conclusion

The proofs in this module are straightforward, and the notion of Galois connections is rather ubiquitous. The theoretician will note that this is due to the fact that this concept is an instance of categorical adjunctions between poset categories.

We will use these proofs first of all as a guide, more or less, for our internal presentation. There, in the generality where 'elements' are a luxury not guaranteed, more care and abstraction will be needed. In addition, the material here well be instantiante with concrete Galois connections occuring in the remainder of this development. In particular, these notions will be used when discussing bounds in Categoric.OSGC.Preorder.Extrema (Sect. 5.2).

3 Residuals in OCCs

Residuals of composition only need the context of locally ordered semigroupoids for their definition and a number of their properties (Sect. 3.1). Some additional properties hold in ordered categories (Sect. 3.2). In the presence of converse, a right-residual operator can be derived from a left-residual operator, and vice versa (Sect. 3.3).

Symmetric quotients were originally studied by Berghammer et al. (1986, 1989) in relation algebras, and by Freyd and Scedrov (1990) in division allegories. In the spirit of the axiomatic definitions of the simple residuals, Furusawa and Kahl (1998) gave a general axiomatic definition in distributive allegories without assuming existence of the simple residuals; Kahl (2008) provided the definition in the context of OSGCs that is formalised in Categoric.OSGC.SyQ (Sect. 3.4). If the simple residuals are available, symmetric quotients are meets, and additional useful properties hold, collected in Categoric.OSGC.SyQ.WithResiduals (Sect. 3.5). Presence of identities brings a few more lemmas, in Categoric.OCC.SyQ (Sect. 3.6).

3.1 Categoric.OrderedSemigroupoid.Residuals

```
record LeftResOp {i j k₁ k₂ : Level} {Obj : Set i}
                 (base : OrderedSemigroupoid j k₁ k₂ Obj)
                 : Set (i ⊍ j ⊍ k₁ ⊍ k₂) where
  open OrderedSemigroupoid base
  infixl 9 _/_
  field
    _/_ : {A B C : Obj} → Mor A C → Mor B C → Mor A B
    /-cancel-outer : {A B C : Obj} {S : Mor A C} {R : Mor B C} → (S / R) ⨾ R ⊑ S
    /-universal :    {A B C : Obj} {S : Mor A C} {R : Mor B C} {Q : Mor A B}
                  → Q ⨾ R ⊑ S → Q ⊑ S / R
  /-couniversal : {A B C : Obj} {S : Mor A C} {R : Mor B C} {T : Mor A B}
                  → ({X : Mor A B} → X ⨾ R ⊑ S → X ⊑ T) → S / R ⊑ T
  /-couniversal = λ couni → couni /-cancel-outer
  /-universal′ :   {A B C : Obj} {S : Mor A C} {R : Mor B C} {Q : Mor A B}
                  → Q ⊑ S / R → Q ⨾ R ⊑ S
  /-universal′ Q⊑S/R = ⨾-monotone₁ Q⊑S/R ⟨⊑⊑⟩ /-cancel-outer
  /-cancel-inner : {A B C : Obj} {T : Mor A B} {S : Mor B C} → T ⊑ (T ⨾ S) / S
  /-cancel-inner = /-universal ⊑-refl
  /-monotone : {A B C : Obj} {S₁ S₂ : Mor A C} {R : Mor B C} → S₁ ⊑ S₂ → S₁ / R ⊑ S₂ / R
  /-monotone S₁⊑S₂ = /-universal (/-cancel-outer ⟨⊑⊑⟩ S₁⊑S₂)
  /-cong₁ : {A B C : Obj} {S₁ S₂ : Mor A C} {R : Mor B C} → S₁ ≈ S₂ → S₁ / R ≈ S₂ / R
  /-cong₁ S₁≈S₂ = ⊑-antisym (/-monotone (⊑-reflexive S₁≈S₂)) (/-monotone (⊑-reflexive′ S₁≈S₂))
  /-antitone : {A B C : Obj} {S : Mor A C} {R₁ R₂ : Mor B C} → R₂ ⊑ R₁ → S / R₁ ⊑ S / R₂
  /-antitone R₂⊑R₁ = /-universal (⨾-monotone₂ R₂⊑R₁ ⟨⊑⊑⟩ /-cancel-outer)
  /-cong₂ : {A B C : Obj} {S : Mor A C} {R₁ R₂ : Mor B C} → R₁ ≈ R₂ → S / R₁ ≈ S / R₂
  /-cong₂ R₁≈R₂ = ⊑-antisym (/-antitone (⊑-reflexive′ R₁≈R₂)) (/-antitone (⊑-reflexive R₁≈R₂))
  /-cong :    {A B C : Obj} {S₁ S₂ : Mor A C} {R₁ R₂ : Mor B C}
           →   S₁ ≈ S₂ → R₁ ≈ R₂ → S₁ / R₁ ≈ S₂ / R₂
  /-cong S₁≈S₂ R₁≈R₂ = /-cong₁ S₁≈S₂ ⟨≈≈⟩ /-cong₂ R₁≈R₂
  /-cancel-outer²    :   {A B C D : Obj} {S : Mor A D} {R : Mor B D} {T : Mor C D}
                       → (S / R) ⨾ (R / T) ⨾ T ⊑ S
  /-cancel-outer²    =   ⨾-monotone₂ /-cancel-outer ⟨⊑⊑⟩ /-cancel-outer
  /-cancel-middle    :   {A B C D : Obj} {S : Mor A D} {R : Mor B D} {T : Mor C D}
                       → (S / R) ⨾ (R / T) ⊑ S / T
  /-cancel-middle    =   /-universal (⨾-assoc ⟨≈⊑⟩ /-cancel-outer²)
  /-cancel-⨾  :   {A B C D : Obj} {S : Mor A C} {R : Mor B C} {T : Mor C D}
               → S / R ⊑ (S ⨾ T) / (R ⨾ T)
  /-cancel-⨾  =   /-universal (⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-outer)
  /-outer-⨾ :   {A B C D : Obj} {F : Mor A B} {S : Mor B D} {R : Mor C D}
             → F ⨾ (S / R) ⊑ (F ⨾ S) / R
  /-outer-⨾ =   /-universal (⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ /-cancel-outer)
  // : {A B C D : Obj} {Q : Mor B C} {R : Mor C D} {S : Mor A D}
       → (S / R) / Q ≈ S / (Q ⨾ R)
  // {Q = Q} {R} {S} = ⊑-antisym
    (/-universal ((⊑-begin
          ((S / R) / Q) ⨾ (Q ⨾ R)
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-outer ⟩
          (S / R) ⨾ R
        ⊑⟨ /-cancel-outer ⟩
          S
        □
    )))
    (/-universal (/-universal (⊑-begin
          ((S / (Q ⨾ R)) ⨾ Q) ⨾ R
        ⊑⟨ ⨾-assoc ⟨≈⊑⟩ /-cancel-outer ⟩
          S
        □
    )))
  /-cancel-⨾-inner       :  {A B C D : Obj} {Q : Mor B C} {R : Mor C D} {S : Mor A D}
                          → (S / (Q ⨾ R)) ⨾ Q ⊑ S / R
  /-cancel-⨾-inner {Q = Q} {R} {S} = ⊑-begin
      (S / (Q ⨾ R)) ⨾ Q
    ≈˘⟨ ⨾-cong₁ // ⟩
      ((S / R) / Q) ⨾ Q
    ⊑⟨ /-cancel-outer ⟩
      S / R
    □

record RightResOp {i j k₁ k₂ : Level} {Obj : Set i}
                  (base : OrderedSemigroupoid j k₁ k₂ Obj)
                  : Set (i ⊍ j ⊍ k₁ ⊍ k₂) where
  open OrderedSemigroupoid base
  infixr 9 _\_
  field
    _\_ : {A B C : Obj} → Mor A B → Mor A C → Mor B C
    \-cancel-outer : {A B C : Obj} {S : Mor A C} {Q : Mor A B} → Q ⨾ (Q \ S) ⊑ S
    \-universal :    {A B C : Obj} {S : Mor A C} {Q : Mor A B} {R : Mor B C}
                  → Q ⨾ R ⊑ S → R ⊑ Q \ S
  \-couniversal : {A B C : Obj} {S : Mor A B} {R : Mor A C} {T : Mor B C}
                  → ({X : Mor B C} → S ⨾ X ⊑ R → X ⊑ T) → S \ R ⊑ T
  \-couniversal = λ couni → couni \-cancel-outer
  \-universal′ :   {A B C : Obj} {S : Mor A C} {Q : Mor A B} {R : Mor B C}
                  → R ⊑ Q \ S → Q ⨾ R ⊑ S
  \-universal′ R⊑Q\S = ⨾-monotone₂ R⊑Q\S ⟨⊑⊑⟩ \-cancel-outer
```

```
\-cancel-inner : {A B C : Obj} {T : Mor B C} {S : Mor A B} → T ⊑ S \ (S ⨾ T)
\-cancel-inner = \-universal ⊑-refl

\-monotone : {A B C : Obj} {S₁ S₂ : Mor A C} {Q : Mor A B} → S₁ ⊑ S₂ → Q \ S₁ ⊑ Q \ S₂
\-monotone S₁⊑S₂ = \-universal (\-cancel-outer ⟨⊑⊑⟩ S₁⊑S₂)

\-cong₂ : {A B C : Obj} {S₁ S₂ : Mor A C} → {Q : Mor A B} → S₁ ≈ S₂ → Q \ S₁ ≈ Q \ S₂
\-cong₂ S₁≈S₂ = ⊑-antisym (\-monotone (⊑-reflexive S₁≈S₂)) (\-monotone (⊑-reflexive′ S₁≈S₂))

\-antitone : {A B C : Obj} {S : Mor A C} {Q₁ Q₂ : Mor A B} → Q₂ ⊑ Q₁ → Q₁ \ S ⊑ Q₂ \ S
\-antitone Q₂⊑Q₁ = \-universal (⨾-monotone₁ Q₂⊑Q₁ ⟨⊑⊑⟩ \-cancel-outer)

\-cong₁ : {A B C : Obj} {S : Mor A C} {Q₁ Q₂ : Mor A B} → Q₁ ≈ Q₂ → Q₁ \ S ≈ Q₂ \ S
\-cong₁ Q₁≈Q₂ = ⊑-antisym (\-antitone (⊑-reflexive′ Q₁≈Q₂)) (\-antitone (⊑-reflexive Q₁≈Q₂))

\-cong :   {A B C : Obj} {S₁ S₂ : Mor A C} {Q₁ Q₂ : Mor A B}
        → Q₁ ≈ Q₂ → S₁ ≈ S₂ → Q₁ \ S₁ ≈ Q₂ \ S₂
\-cong Q₁≈Q₂ S₁≈S₂ = \-cong₂ S₁≈S₂ ⟨≈≈⟩ \-cong₁ Q₁≈Q₂

\-cancel-outer²   :  {A B C D : Obj} {S : Mor A D} {Q : Mor A C} {T : Mor A B}
                  → T ⨾ (T \ Q) ⨾ (Q \ S) ⊑ S
\-cancel-outer²   =  ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ \-cancel-outer ⟨⊑⊑⟩ \-cancel-outer
\-cancel-middle   :  {A B C D : Obj} {S : Mor A D} {Q : Mor A C} {T : Mor A B}
                  → (T \ Q) ⨾ (Q \ S) ⊑ T \ S
\-cancel-middle   =  \-universal \-cancel-outer²

\-cancel-⨾  :  {A B C D : Obj} {S : Mor B D} {Q : Mor B C} {T : Mor A B}
            → Q \ S ⊑ (T ⨾ Q) \ (T ⨾ S)
\-cancel-⨾  =  \-universal (⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ \-cancel-outer)

\-outer-⨾ :  {A B C D : Obj} {F : Mor C D} {S : Mor A C} {Q : Mor A B}
          → (Q \ S) ⨾ F ⊑ Q \ (S ⨾ F)
\-outer-⨾ =  \-universal (⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ \-cancel-outer)

\\ : {A B C D : Obj} {Q : Mor A B} {R : Mor B C} {S : Mor A D}
     → R \ (Q \ S) ≈ (Q ⨾ R) \ S
\\ {Q = Q} {R} {S} = ⊑-antisym
   (\-universal ((⊑-begin
         (Q ⨾ R) ⨾ R \ Q \ S
      ⊑⟨ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ \-cancel-outer ⟩
         Q ⨾ (Q \ S)
      ⊑⟨ \-cancel-outer ⟩
         S
      □
   )))
   (\-universal (\-universal (⊑-begin
         Q ⨾ R ⨾ ((Q ⨾ R) \ S)
      ⊑⟨ ⨾-assocL ⟨≈⊑⟩ \-cancel-outer ⟩
         S
      □
   )))

\-cancel-⨾-inner        :  {A B C D : Obj} {Q : Mor A B} {R : Mor B C} {S : Mor A D}
                        → R ⨾ ((Q ⨾ R) \ S) ⊑ Q \ S
\-cancel-⨾-inner {Q = Q} {R} {S} = ⊑-begin
      R ⨾ ((Q ⨾ R) \ S)
   ≈˘⟨ ⨾-cong₂ \\ ⟩
      R ⨾ (R \ (Q \ S))
   ⊑⟨ \-cancel-outer ⟩
      Q \ S
   □
```

```
module ResidualOps {i j k₁ k₂ : Level} {Obj : Set i}
                   {base : OrderedSemigroupoid j k₁ k₂ Obj}
                   (leftResOp : LeftResOp base)
                   (rightResOp : RightResOp base) where
```

```
open OrderedSemigroupoid base
open LeftResOp   leftResOp   public
open RightResOp rightResOp public

\/-⊑ : {A B C D : Obj} {S : Mor A D} {Q : Mor A B} {R : Mor C D} → Q \ (S / R) ⊑ (Q \ S) / R
\/-⊑ {S = S} {Q} {R} = /-universal (\-outer-⨾ ⟨⊑⊑⟩ \-monotone /-cancel-outer)

\/-⊒ : {A B C D : Obj} {S : Mor A D} {Q : Mor A B} {R : Mor C D} → (Q \ S) / R ⊑ Q \ (S / R)
\/-⊒ {S = S} {Q} {R} = \-universal (/-outer-⨾ ⟨⊑⊑⟩ /-monotone \-cancel-outer)

\/-≈ : {A B C D : Obj} {S : Mor A D} {Q : Mor A B} {R : Mor C D} → Q \ (S / R) ≈ (Q \ S) / R
\/-≈ {S = S} {Q} {R} = ⊑-antisym \/-⊑ \/-⊒

/-twist : {A B C D : Obj} {S : Mor A C} {R : Mor B C} {T : Mor D C} → S / R ⊑ (T / S) \ (T / R)
/-twist = \-universal /-cancel-middle

\-twist : {A B C D : Obj} {S : Mor A C} {Q : Mor A B} {T : Mor A D} → Q \ S ⊑ (Q \ T) / (S \ T)
\-twist = /-universal \-cancel-middle

  -- (Furusawa and Kahl, 1998, Lemma 4.9.ii)
/-twist-down : {A B C : Obj} {S : Mor A C} {R : Mor B C} → S / R ⊑ (R / S) \ (R / R)
/-twist-down = \-universal /-cancel-middle

\-twist-down : {A B C : Obj} {S : Mor A C} {Q : Mor A B} → Q \ S ⊑ (Q \ Q) / (S \ Q)
\-twist-down = /-universal \-cancel-middle

/-twist-up : {A B C : Obj} {S : Mor A C} {R : Mor B C} → S / R ⊑ (S / S) \ (S / R)
/-twist-up = /-twist

\-twist-up : {A B C : Obj} {S : Mor A C} {Q : Mor A B} → Q \ S ⊑ (Q \ S) / (S \ S)
\-twist-up = \-twist
```

For `/-twist-up` in ordered categories, (Furusawa and Kahl, 1998, Lemma 4.9.i) showed ≈, using `Id {A} ⊑ S / S`, see Sect. 3.2. There is a two-element ordered semigroup that does not satisfy `(S / S) ⨾ S ≈ S`, and a three-element linearly ordered semigroup that does not satisfy ⊒.

```
⊑-S/∘\S : {A B C : Obj} {S : Mor A C} {Q : Mor A B} → Q ⊑ S / (Q \ S)
⊑-S/∘\S {A} {B} {C} {S} {Q} = /-universal (⊑-begin
      Q ⨾ (Q \ S)
   ⊑⟨ \-cancel-outer ⟩
      S
   □)

⊑-\S∘S/ : {A B C : Obj} {S : Mor A C} {R : Mor B C} → R ⊑ (S / R) \ S
⊑-\S∘S/ {A} {B} {C} {S} {R} = \-universal (⊑-begin
      (S / R) ⨾ R
   ⊑⟨ /-cancel-outer ⟩
      S
   □)

S/∘\S∘S/ : {A B C : Obj} {S : Mor A C} {R : Mor B C} → S / ((S / R) \ S) ≈ S / R
S/∘\S∘S/ {A} {B} {C} {S} {R} = ⊑-antisym (/-antitone ⊑-\S∘S/) ⊑-S/∘\S

\S∘S/∘\S : {A B C : Obj} {S : Mor A C} {Q : Mor A B} → (S / (Q \ S)) \ S ≈ Q \ S
\S∘S/∘\S {A} {B} {C} {S} {Q} = ⊑-antisym (\-antitone ⊑-S/∘\S) ⊑-\S∘S/

T/∘\S∘S/ : {A₁ A₂ B C : Obj} {T : Mor A₁ C} {S : Mor A₂ C} {R : Mor B C}
   → (S / R) \ S ⊑ (T / R) \ T → T / ((S / R) \ S) ≈ T / R
T/∘\S∘S/ {A₁} {A₂} {B} {C} {T} {S} {R} p = ⊑-antisym (/-antitone ⊑-\S∘S/)
   (⊑-S/∘\S ⟨⊑⊑⟩ /-antitone p)

\T∘S/∘\S : {A B C₁ C₂ : Obj} {T : Mor A C₁} {S : Mor A C₂} {Q : Mor A B}
   → S / (Q \ S) ⊑ T / (Q \ T) → (S / (Q \ S)) \ T ≈ Q \ T
\T∘S/∘\S {A} {B} {C₁} {C₂} {T} {S} {Q} p = ⊑-antisym (\-antitone ⊑-S/∘\S)
   (⊑-\S∘S/ ⟨⊑⊑⟩ \-antitone p)

retractLeftResOp :  {i₁ i₂ j k₁ k₂ : Level} {Obj₁ : Set i₁} {Obj₂ : Set i₂}
                 → (F : Obj₂ → Obj₁)
```

```
                      → {base : OrderedSemigroupoid j k₁ k₂ Obj₁}
                      → LeftResOp base → LeftResOp (retractOrderedSemigroupoid F base)
retractLeftResOp F leftResOp = let open LeftResOp leftResOp in record
  { _/_ = _/_
  ; /-cancel-outer = /-cancel-outer
  ; /-universal = /-universal
  }

retractRightResOp : {i₁ i₂ j k₁ k₂ : Level} {Obj₁ : Set i₁} {Obj₂ : Set i₂}
                      → (F : Obj₂ → Obj₁)
                      → {base : OrderedSemigroupoid j k₁ k₂ Obj₁}
                      → RightResOp base → RightResOp (retractOrderedSemigroupoid F base)
retractRightResOp F rightResOp = let open RightResOp rightResOp in record
  { _\_ = _\_
  ; \-cancel-outer = \-cancel-outer
  ; \-universal = \-universal
  }
```

3.2 Categoric.OrderedCategory.Residuals

```
module OrdCat-LeftRes-Props {i j k₁ k₂ : Level} {Obj : Set i}
  (base : OrderedCategory j k₁ k₂ Obj)
  (leftResOp : LeftResOp (OrderedCategory.orderedSemigroupoid base))
    where
  open OrderedCategory base
  open LeftResOp leftResOp

  /-isReflexive : {A B : Obj} {R : Mor A B} → Id ⊑ R / R
  /-isReflexive = /-universal (⊑-reflexive leftId)

  /-isSuperidentity : {A B : Obj} {R : Mor A B} → isSuperidentity (R / R)
  /-isSuperidentity = reflexiveIsSuperidentity /-isReflexive

  /-Id : {A B : Obj} {R : Mor A B} → R / Id ≈ R
  /-Id {_} {_} {R} = ⊑-antisym
    (⊑-begin
        R / Id
      ≈⟨ ≈-sym rightId ⟩
        (R / Id) ⨾ Id
      ⊑⟨ /-cancel-outer ⟩
        R
      □)
    (/-universal (⊑-reflexive rightId))

  preorder-/ : {A : Obj} {E : Mor A A} → IsReflexive E → IsTransitive E → E / E ≈ E
  preorder-/ refl trans = ⊑-antisym (/-antitone refl ⟨⊑≈⟩ /-Id) (/-universal trans)

module OrdCat-RightRes-Props {i j k₁ k₂ : Level} {Obj : Set i}
  (base : OrderedCategory j k₁ k₂ Obj)
  (rightResOp : RightResOp (OrderedCategory.orderedSemigroupoid base))
    where
  open OrderedCategory base
  open RightResOp rightResOp

  \-isReflexive : {A B : Obj} {R : Mor A B} → Id ⊑ R \ R
  \-isReflexive = \-universal (⊑-reflexive rightId)

  \-isSuperidentity : {A B : Obj} {R : Mor A B} → isSuperidentity (R \ R)
  \-isSuperidentity = reflexiveIsSuperidentity \-isReflexive

  Id-\ : {A B : Obj} {R : Mor A B} → Id \ R ≈ R
  Id-\ {_} {_} {R} = ⊑-antisym
    (⊑-begin
        Id \ R
      ≈⟨ ≈-sym leftId ⟩
        Id ⨾ (Id \ R)
      ⊑⟨ \-cancel-outer ⟩
        R
      □)
    (\-universal (⊑-reflexive leftId))

  preorder-\ : {A : Obj} {E : Mor A A} → IsReflexive E → IsTransitive E → E \ E ≈ E
  preorder-\ refl trans = ⊑-antisym (\-antitone refl ⟨⊑≈⟩ Id-\) (\-universal trans)

module OrdCat-Residual-Props {i j k₁ k₂ : Level} {Obj : Set i}
  (base : OrderedCategory j k₁ k₂ Obj)
  (leftResOp : LeftResOp (OrderedCategory.orderedSemigroupoid base))
  (rightResOp : RightResOp (OrderedCategory.orderedSemigroupoid base))
    where
  open OrderedCategory base
  open ResidualOps leftResOp rightResOp
  open OrdCat-LeftRes-Props  base leftResOp  public
  open OrdCat-RightRes-Props base rightResOp public

  -- (Furusawa and Kahl, 1998, Lemma 4.9.i)
  /-twist-up-≈ : {A B C : Obj} {S : Mor A C} {R : Mor B C} → S / R ≈ (S / S) \ (S / R)
  /-twist-up-≈ {S = S} {R} = ⊑-antisym /-twist-up
    (⊑-begin
        (S / S) \ (S / R)
      ⊑⟨ \-antitone /-isReflexive ⟩
        Id \ (S / R)
      ≈⟨ Id-\ ⟩
        S / R
      □)

  \-twist-up-≈ : {A B C : Obj} {S : Mor A C} {Q : Mor A B} → Q \ S ≈ (Q \ S) / (S \ S)
  \-twist-up-≈ {S = S} {Q} = ⊑-antisym \-twist-up
    (⊑-begin
        (Q \ S) / (S \ S)
      ⊑⟨ /-antitone \-isReflexive ⟩
        (Q \ S) / Id
      ≈⟨ /-Id ⟩
        Q \ S
      □)
```

3.3 Categoric.OSGC.Residuals

```
module OSGC-Residuals {i j k₁ k₂ : Level} {Obj : Set i}
  (base : OSGC j k₁ k₂ Obj)
  (leftResOp   : LeftResOp (OSGC.orderedSemigroupoid base))
  (rightResOp  : RightResOp (OSGC.orderedSemigroupoid base))
  where
  open OSGC base
  open LeftResOp leftResOp
  open RightResOp rightResOp

  ˘/˘-universal : {A B C : Obj} {S : Mor C A} {R : Mor C B} {Q : Mor A B}
                  → R ⨾ Q ˘ ⊑ S → Q ⊑ S ˘ / R ˘
  ˘/˘-universal R⨾Q˘⊑S = /-universal (˘-involutionRightConv ⟨≈˘⊑⟩ ˘-monotone R⨾Q˘⊑S)
```

```
˘\˘-universal : {A B C : Obj} {S : Mor C A} {Q : Mor B A} {R : Mor B C}
            → R ˘ ⨾ Q ⊑ S → R ⊑ Q ˘ \ S ˘
˘\˘-universal R˘⨾Q⊑S = \-universal (˘-involutionLeftConv ⟨≈˘⊑⟩ ˘-monotone R˘⨾Q⊑S)

\-inner-⨾-⊑ : {A B C D : Obj} {S : Mor A D} {Q : Mor A B} {F : Mor C B}
          → Q ⨾ F ˘ ⨾ F ⊑ Q → F ⨾ (Q \ S) ⊑ (Q ⨾ F ˘) \ S
\-inner-⨾-⊑ {S = S} {Q} {F} Q⨾F˘⨾F⊑Q = \-universal (⊑-begin
      (Q ⨾ F ˘) ⨾ F ⨾ (Q \ S)
    ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ (⨾-assoc ⟨≈⊑⟩ Q⨾F˘⨾F⊑Q) ⟩
      Q ⨾ (Q \ S)
    ⊑⟨ \-cancel-outer ⟩
      S
    □)
/-inner-⨾-⊑ : {A B C D : Obj} {S : Mor A D} {R : Mor C D} {F : Mor B C}
          → F ˘ ⨾ F ⨾ R ⊑ R → (S / R) ⨾ F ˘ ⊑ S / (F ⨾ R)
/-inner-⨾-⊑ {S = S} {R} {F} F˘⨾F⨾R⊑R = /-universal (⊑-begin
      ((S / R) ⨾ F ˘) ⨾ (F ⨾ R)
    ⊑⟨ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ F˘⨾F⨾R⊑R ⟩
      (S / R) ⨾ R
    ⊑⟨ /-cancel-outer ⟩
      S
    □)

\-inner-⨾-unival : {A B C D : Obj} {S : Mor A D} {Q : Mor A B} {F : Mor C B}
                → isUnivalent F → F ⨾ (Q \ S) ⊑ (Q ⨾ F ˘) \ S
\-inner-⨾-unival F-unival = \-inner-⨾-⊑ (proj₂ F-unival)
/-inner-⨾-unival : {A B C D : Obj} {S : Mor A D} {R : Mor C D} {F : Mor B C}
                → isUnivalent F → (S / R) ⨾ F ˘ ⊑ S / (F ⨾ R)
/-inner-⨾-unival F-unival = /-inner-⨾-⊑ (⨾-assocL ⟨≈⊑⟩ proj₁ F-unival)

\-inner-⨾-total : {A B C D : Obj} {S : Mor A D} {Q : Mor A B} {F : Mor C B}
                → isTotal F → (Q ⨾ F ˘) \ S ⊑ F ⨾ (Q \ S)
\-inner-⨾-total {S = S} {Q} {F} F-total = ⊑-begin
      (Q ⨾ F ˘) \ S
    ⊑⟨ proj₁ F-total ⟨⊑≈⟩ ⨾-assoc ⟩
      F ⨾ F ˘ ⨾ ((Q ⨾ F ˘) \ S)
    ⊑⟨ ⨾-monotone₂₁ \-cancel-inner ⟩
      F ⨾ (Q \ (Q ⨾ F ˘)) ⨾ ((Q ⨾ F ˘) \ S)
    ⊑⟨ ⨾-monotone₂ \-cancel-middle ⟩
      F ⨾ (Q \ S)
    □
/-inner-⨾-total : {A B C D : Obj} {S : Mor A D} {R : Mor C D} {F : Mor B C}
                → isTotal F → S / (F ⨾ R) ⊑ (S / R) ⨾ F ˘
/-inner-⨾-total {S = S} {R} {F} F-total = ⊑-begin
      S / (F ⨾ R)
    ⊑⟨ proj₂ F-total ⟩
      (S / (F ⨾ R)) ⨾ F ⨾ F ˘
    ⊑⟨ ⨾-monotone₂₁ /-cancel-inner ⟩
      (S / (F ⨾ R)) ⨾ ((F ⨾ R) / R) ⨾ F ˘
    ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-middle ⟩
      (S / R) ⨾ F ˘
    □

\-inner-⨾ : {A B C D : Obj} {S : Mor A D} {Q : Mor A B} {F : Mor C B}
          → isMapping F → F ⨾ (Q \ S) ≈ (Q ⨾ F ˘) \ S
\-inner-⨾ {S = S} {Q} {F} (F-unival, F-total) = ⊑-antisym
  (\-inner-⨾-unival F-unival) (\-inner-⨾-total F-total)
```

```
/-inner-⨾ : {A B C D : Obj} {S : Mor A D} {R : Mor C D} {F : Mor B C}
          → isMapping F → (S / R) ⨾ F ˘ ≈ S / (F ⨾ R)
/-inner-⨾ {S = S} {R} {F} (F-unival, F-total) = ⊑-antisym
  (/-inner-⨾-unival F-unival) (/-inner-⨾-total F-total)

/-outer-⨾-⊒ : {A B C D : Obj} {F : Mor A B} {S : Mor B D} {R : Mor C D}
            → isMapping F → (F ⨾ S) / R ⊑ F ⨾ (S / R)
/-outer-⨾-⊒ {F = F} {S} {R} (F-unival, F-total) = ⊑-begin
    (F ⨾ S) / R
  ⊑⟨ proj₁ F-total ⟨⊑≈⟩ ⨾-assoc ⟩
    F ⨾ F ˘ ⨾ ((F ⨾ S) / R)
  ⊑⟨ ⨾-monotone₂ /-outer-⨾ ⟩
    F ⨾ ((F ˘ ⨾ F ⨾ S) / R)
  ⊑⟨ ⨾-monotone₂ (/-monotone (⨾-assocL ⟨≈⊑⟩ proj₁ F-unival)) ⟩
    F ⨾ (S / R)
  □
/-outer-⨾-≈ : {A B C D : Obj} {F : Mor A B} {S : Mor B D} {R : Mor C D}
            → isMapping F → F ⨾ (S / R) ≈ (F ⨾ S) / R
/-outer-⨾-≈ F-mapping = ⊑-antisym /-outer-⨾ (/-outer-⨾-⊒ F-mapping)

\-outer-⨾-⊒ : {A B C D : Obj} {F : Mor D C} {S : Mor A C} {Q : Mor A B}
            → isMapping F → Q \ (S ⨾ F ˘) ⊑ (Q \ S) ⨾ F ˘
\-outer-⨾-⊒ {F = F} {S} {Q} (F-unival, F-total) = ⊑-begin
    Q \ (S ⨾ F ˘)
  ⊑⟨ proj₂ F-total ⟨⊑≈⟩ ⨾-assocL ⟩
    ((Q \ (S ⨾ F ˘)) ⨾ F) ⨾ F ˘
  ⊑⟨ ⨾-monotone₁ \-outer-⨾ ⟩
    (Q \ (S ⨾ F ˘) ⨾ F) ⨾ F ˘
  ⊑⟨ ⨾-monotone₁ (\-monotone (⨾-assoc ⟨≈⊑⟩ proj₂ F-unival)) ⟩
    (Q \ S) ⨾ F ˘
  □
\-outer-⨾-≈ : {A B C D : Obj} {F : Mor D C} {S : Mor A C} {Q : Mor A B}
            → isMapping F → (Q \ S) ⨾ F ˘ ≈ Q \ (S ⨾ F ˘)
\-outer-⨾-≈ F-mapping = ⊑-antisym \-outer-⨾ (\-outer-⨾-⊒ F-mapping)

/-flip-⊑ : {A B C D : Obj} {S : Mor A D} {R : Mor B C} {F : Mor C D}
         → isTotal F → S / (R ⨾ F) ⊑ (S ⨾ F ˘) / R
/-flip-⊑ {S = S} {R} {F} F-total = /-universal (⊑-begin
      (S / (R ⨾ F)) ⨾ R
    ⊑⟨ /-cancel-⨾-inner ⟩
      S / F
    ⊑⟨ proj₂ F-total ⟩
      (S / F) ⨾ F ⨾ F ˘
    ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-outer ⟩
      S ⨾ F ˘
    □)
/-flip-⊒ : {A B C D : Obj} {S : Mor A D} {R : Mor B C} {F : Mor C D}
         → isUnivalent F → (S ⨾ F ˘) / R ⊑ S / (R ⨾ F)
/-flip-⊒ {S = S} {R} {F} F-unival = /-universal (⊑-begin
      ((S ⨾ F ˘) / R) ⨾ (R ⨾ F)
    ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-outer ⟩
      (S ⨾ F ˘) ⨾ F
    ⊑⟨ ⨾-assoc ⟨≈⊑⟩ proj₂ F-unival ⟩
      S
    □)
/-flip : {A B C D : Obj} {S : Mor A D} {R : Mor B C} {F : Mor C D}
       → isMapping F → S / (R ⨾ F) ≈ (S ⨾ F ˘) / R
```

```
/-flip (F-unival, F-total) = ⊑-antisym (/-flip-⊑ F-total) (/-flip-⊒ F-unival)
/-flip-˘  :  {A B C D : Obj} {S : Mor A D} {R : Mor B C} {F : Mor D C}
        → isMapping (F ˘) → S / (R ⨾ F ˘) ≈ (S ⨾ F) / R
/-flip-˘ F˘-isMappping = /-flip F˘-isMappping ⟨≈≈⟩ /-cong₁ (⨾-cong₂ ˘˘)

\-flip-⊑  :  {A B C D : Obj} {S : Mor A D} {F : Mor A B} {Q : Mor B C}
          → isSurjective F → (F ⨾ Q) \ S ⊑ Q \ (F ˘ ⨾ S)
\-flip-⊑ {S = S} {F} {Q} F-surj = \-universal (⊑-begin
      Q ⨾ ((F ⨾ Q) \ S)
    ⊑⟨ \-cancel-⨾-inner ⟩
      F \ S
    ⊑⟨ proj₁ F-surj ⟨⊑≈⟩ ⨾-assoc ⟩
      F ˘ ⨾ F ⨾ (F \ S)
    ⊑⟨ ⨾-monotone₂ \-cancel-outer ⟩
      F ˘ ⨾ S
    □)
\-flip-⊒  :  {A B C D : Obj} {S : Mor A D} {F : Mor A B} {Q : Mor B C}
          → isInjective F → Q \ (F ˘ ⨾ S) ⊑ (F ⨾ Q) \ S
\-flip-⊒ {S = S} {F} {Q} F-inj = \-universal (⊑-begin
      (F ⨾ Q) ⨾ (Q \ (F ˘ ⨾ S))
    ⊑⟨ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ \-cancel-outer ⟩
      F ⨾ F ˘ ⨾ S
    ⊑⟨ ⨾-assocL ⟨≈⊑⟩ proj₁ F-inj ⟩
      S
    □)
\-flip     : {A B C D : Obj} {S : Mor A D} {F : Mor A B} {Q : Mor B C}
          → isBijective F → (F ⨾ Q) \ S ≈ Q \ (F ˘ ⨾ S)
\-flip (F-inj, F-surj) = ⊑-antisym (\-flip-⊑ F-surj) (\-flip-⊒ F-inj)
\-flip-˘  :  {A B C D : Obj} {S : Mor A D} {F : Mor B A} {Q : Mor B C}
        → isBijective (F ˘) → (F ˘ ⨾ Q) \ S ≈ Q \ (F ⨾ S)
\-flip-˘ F˘-isBij = \-flip F˘-isBij ⟨≈≈⟩ \-cong₂ (⨾-cong₁ ˘˘)
\-flip-M :  {A B C D : Obj} {S : Mor A D} {F : Mor B A} {Q : Mor B C}
        → isMapping F → (F ˘ ⨾ Q) \ S ≈ Q \ (F ⨾ S)
\-flip-M F-isMapping = \-flip (˘-isBijective F-isMapping) ⟨≈≈⟩ \-cong₂ (⨾-cong₁ ˘˘)

/-˘ : {A B C : Obj} {S : Mor A C} {R : Mor B C} → (S / R) ˘ ≈ R ˘ \ S ˘
/-˘ {A} {B} {C} {S} {R} = ⊑-antisym
  (\-universal (⊑-begin
      R ˘ ⨾ (S / R) ˘
    ≈˘⟨ ˘-involution ⟩
      ((S / R) ⨾ R) ˘
    ⊑⟨ ˘-monotone /-cancel-outer ⟩
      S ˘
    □))
  (⊑-˘-swap (/-universal (⊑-begin
      (R ˘ \ S ˘) ˘ ⨾ R
    ≈˘⟨ ˘-involutionLeftConv ⟩
      (R ˘ ⨾ (R ˘ \ S ˘)) ˘
    ⊑⟨ ˘-monotone \-cancel-outer ⟨⊑≈⟩ ˘˘ ⟩
      S
    □)))
/˘-˘ : {A B C : Obj} {S : Mor A C} {R : Mor C B} → (S / R ˘) ˘ ≈ R \ S ˘
/˘-˘ {A} {B} {C} {S} {R} = ≈-begin
        (S / R ˘) ˘
      ≈⟨ /-˘ ⟩
        R ˘ ˘ \ S ˘
      ≈⟨ \-cong₁ ˘˘ ⟩
        R \ S ˘
        □
˘/-˘ : {A B C : Obj} {S : Mor C A} {R : Mor B C} → (S ˘ / R) ˘ ≈ R ˘ \ S
˘/-˘ {A} {B} {C} {S} {R} = ≈-begin
        (S ˘ / R) ˘
      ≈⟨ /-˘ ⟩
        R ˘ \ S ˘ ˘
      ≈⟨ \-cong₂ ˘˘ ⟩
        R ˘ \ S
        □
˘/˘-˘ : {A B C : Obj} {S : Mor C A} {R : Mor C B} → (S ˘ / R ˘) ˘ ≈ R \ S
˘/˘-˘ {A} {B} {C} {S} {R} = ≈-begin
        (S ˘ / R ˘) ˘
      ≈⟨ /˘-˘ ⟩
        R \ S ˘ ˘
      ≈⟨ \-cong₂ ˘˘ ⟩
        R \ S
        □
\-˘ : {A B C : Obj} {Q : Mor A B} {S : Mor A C} → (Q \ S) ˘ ≈ S ˘ / Q ˘
\-˘ = ≈-sym (˘-≈-swap ˘/˘-˘)
˘\-˘ : {A B C : Obj} {Q : Mor B A} {S : Mor A C} → (Q ˘ \ S) ˘ ≈ S ˘ / Q
˘\-˘ = ≈-sym (˘-≈-swap ˘/-˘)
\˘-˘ : {A B C : Obj} {Q : Mor A B} {S : Mor C A} → (Q \ S ˘) ˘ ≈ S / Q ˘
\˘-˘ = ≈-sym (˘-≈-swap /˘-˘)
˘\˘-˘ : {A B C : Obj} {Q : Mor B A} {S : Mor C A} → (Q ˘ \ S ˘) ˘ ≈ S / Q
˘\˘-˘ = ≈-sym (˘-≈-swap /-˘)

\-cancel-inner-⊑ : {A B C : Obj} {T : Mor B C} {S : Mor A B}
                  → isLeftIdentity (S ˘ ⨾ S) → S \ (S ⨾ T) ⊑ T
\-cancel-inner-⊑ {T = T} {S} S-leftId = ⊑-begin
      S \ (S ⨾ T)
    ≈⟨ S-leftId ⟨≈˘≈⟩ ⨾-assoc ⟩
      S ˘ ⨾ S ⨾ (S \ (S ⨾ T))
    ⊑⟨ ⨾-monotone₂ \-cancel-outer ⟩
      S ˘ ⨾ S ⨾ T
    ≈⟨ ⨾-assocL ⟨≈≈⟩ S-leftId ⟩
      T
    □
\-cancel-inner-≈ : {A B C : Obj} {T : Mor B C} {S : Mor A B}
                  → isLeftIdentity (S ˘ ⨾ S) → S \ (S ⨾ T) ≈ T
\-cancel-inner-≈ S-leftId = ⊑-antisym (\-cancel-inner-⊑ S-leftId) \-cancel-inner
\-cancel-outer-⊒ : {A B C : Obj} {S : Mor A C} {Q : Mor A B}
                  → isLeftIdentity (Q ⨾ Q ˘) → S ⊑ Q ⨾ (Q \ S)
\-cancel-outer-⊒ {S = S} {Q} Q-leftId = ⊑-begin
      S
    ≈⟨ Q-leftId ⟨≈˘≈⟩ ⨾-assoc ⟩
      Q ⨾ Q ˘ ⨾ S
    ⊑⟨ ⨾-monotone₂ (\-universal (⊑-begin
        Q ⨾ Q ˘ ⨾ S
      ≈⟨ ⨾-assocL ⟨≈≈⟩ Q-leftId ⟩
        S
      □)) ⟩
      Q ⨾ (Q \ S)
    □
\-cancel-outer-≈ : {A B C : Obj} {S : Mor A C} {Q : Mor A B}
                  → isLeftIdentity (Q ⨾ Q ˘) → Q ⨾ (Q \ S) ≈ S
\-cancel-outer-≈ Q-leftId = ⊑-antisym \-cancel-outer (\-cancel-outer-⊒ Q-leftId)
```

```
/-cancel-inner-⊑  : {A B C : Obj} {S : Mor A B} {T : Mor B C}
                  → isRightIdentity (T ⨾ T ˘) → (S ⨾ T) / T ⊑ S
/-cancel-inner-⊑ {S = S} {T} T-rightId  =  ⊑-begin
        (S ⨾ T) / T
    ≈˘⟨ T-rightId ⟩
        ((S ⨾ T) / T) ⨾ T ⨾ T ˘
    ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-outer ⟩
        (S ⨾ T) ⨾ T ˘
    ≈⟨ ⨾-assoc ⟨≈≈⟩ T-rightId ⟩
        S
    □

/-cancel-inner-≈  : {A B C : Obj} {T : Mor B C} {S : Mor A B}
                  → isRightIdentity (T ⨾ T ˘) → (S ⨾ T) / T ≈ S
/-cancel-inner-≈ T-rightId  =  ⊑-antisym (/-cancel-inner-⊑ T-rightId) /-cancel-inner

/-cancel-outer-⊒  : {A B C : Obj} {S : Mor A C} {R : Mor B C}
                  → isRightIdentity (R ˘ ⨾ R) → S ⊑ (S / R) ⨾ R
/-cancel-outer-⊒ {S = S} {R} R-rightId  =  ⊑-begin
        S
    ≈⟨ R-rightId ⟨≈˘≈⟩ ⨾-assocL ⟩
        (S ⨾ R ˘) ⨾ R
    ⊑⟨ ⨾-monotone₁ (/-universal (⊑-begin
            (S ⨾ R ˘) ⨾ R
        ≈⟨ ⨾-assoc ⟨≈≈⟩ R-rightId ⟩
            S
        □)) ⟩
        (S / R) ⨾ R
    □

/-cancel-outer-≈  : {A B C : Obj} {S : Mor A C} {R : Mor B C}
                  → isRightIdentity (R ˘ ⨾ R) → (S / R) ⨾ R ≈ S
/-cancel-outer-≈ R-rightId  =  ⊑-antisym /-cancel-outer (/-cancel-outer-⊒ R-rightId)


module RightResOp-from-LeftResOp {i j k₁ k₂ : Level} {Obj : Set i}
  (base : OSGC j k₁ k₂ Obj)
  (leftResOp : LeftResOp (OSGC.orderedSemigroupoid base)) where
  open OSGC base
  open LeftResOp leftResOp

  rightResOp : RightResOp orderedSemigroupoid
  rightResOp  =  record
    {_\_  =  λ {A} {B} {C} Q S → (S ˘ / Q ˘) ˘
    ; \-cancel-outer  =  λ {_} {_} {_} {S} {Q} → ⊑-begin
          Q ⨾ (S ˘ / Q ˘) ˘
      ≈⟨ ≈-sym ˘-involutionRightConv ⟩
          ((S ˘ / Q ˘) ⨾ Q ˘) ˘
      ⊑⟨ ˘-monotone /-cancel-outer ⟩
          (S ˘) ˘
      ≈⟨ ˘˘ ⟩
          S
      □
    ; \-universal  =  λ {_} {_} {_} {S} {Q} {R} Q⨾R⊑S → ⊑-˘-swap (/-universal (⊑-begin
          R ˘ ⨾ Q ˘
      ≈⟨ ≈-sym ˘-involution ⟩
          (Q ⨾ R) ˘
      ⊑⟨ ˘-monotone Q⨾R⊑S ⟩
          S ˘
      □))
    }
```

3.4 Categoric.OSGC.SyQ

For background on **symmetric quotients**, see the papers by Berghammer et al. (1986, 1989), Zierer (1991), Schmidt and Ströhlein (1993, Sect. 4.4) and Furusawa and Kahl (1998).

```
record SyqOp {i j k₁ k₂ : Level} {Obj : Set i}
                (base : OSGC j k₁ k₂ Obj)
                : Set (i ⊍ j ⊍ k₁ ⊍ k₂) where
  open OSGC base
  infix 9 _╳_
  field
    _╳_ : {A B C : Obj} → Mor A B → Mor A C → Mor B C
    ╳-cong          :   {A B C : Obj} {Q₁ Q₂ : Mor A B} {S₁ S₂ : Mor A C}
                      → Q₁ ≈ Q₂ → S₁ ≈ S₂ → Q₁ ╳ S₁ ≈ Q₂ ╳ S₂
    ╳-cancel-left   :   {A B C : Obj} {Q : Mor A B} {S : Mor A C} → Q ⨾ (Q ╳ S)  ⊑ S
    ╳-cancel-right  :   {A B C : Obj} {Q : Mor A B} {S : Mor A C} → (Q ╳ S) ⨾ S ˘ ⊑ Q ˘
    ╳-universal     :   {A B C : Obj} {Q : Mor A B} {S : Mor A C} {R : Mor B C}
                      → Q ⨾ R ⊑ S → R ⨾ S ˘ ⊑ Q ˘ → R ⊑ Q ╳ S

  ╳-cong₁ : {A B C : Obj} {Q₁ Q₂ : Mor A B} {S : Mor A C} → Q₁ ≈ Q₂ → Q₁ ╳ S ≈ Q₂ ╳ S
  ╳-cong₁ Q₁≈Q₂  =  ╳-cong Q₁≈Q₂ ≈-refl
  ╳-cong₂ : {A B C : Obj} {Q : Mor A B} {S₁ S₂ : Mor A C} → S₁ ≈ S₂ → Q ╳ S₁ ≈ Q ╳ S₂
  ╳-cong₂  =  ╳-cong ≈-refl
```

The following variants of `╳-universal` save tedious involutions in applications:

```
  ˘╳-universal     :   {A B C : Obj} {Q : Mor B A} {S : Mor A C} {R : Mor B C}
                     → Q ˘ ⨾ R      ⊑ S → R ⨾ S ˘ ⊑ Q → R ⊑ Q ˘ ╳ S
  ˘╳-universal Q˘⨾R⊑S R⨾S˘⊑Q  =  ╳-universal Q˘⨾R⊑S (R⨾S˘⊑Q ⟨⊑≈˘⟩ ˘˘)
  ╳˘-universal     :   {A B C : Obj} {Q : Mor A B} {S : Mor C A} {R : Mor B C}
                     → Q ⨾ R ⊑ S ˘ → R ⨾ S ⊑ Q ˘ → R ⊑ Q ╳ S ˘
  ╳˘-universal Q⨾R⊑S˘ R⨾S⊑Q˘  =  ╳-universal Q⨾R⊑S˘ (⨾-cong₂ ˘˘ ⟨≈⊑⟩ R⨾S⊑Q˘)
  ˘╳˘-universal    :   {A B C : Obj} {Q : Mor B A} {S : Mor C A} {R : Mor B C}
                     → Q ˘ ⨾ R      ⊑ S ˘ → R ⨾ S ⊑ Q → R ⊑ Q ˘ ╳ S ˘
  ˘╳˘-universal Q˘⨾R⊑S˘ R⨾S⊑Q  =  ╳˘-universal Q˘⨾R⊑S˘ (R⨾S⊑Q ⟨⊑≈˘⟩ ˘˘)

  ╳-universal-right     :   {A B C : Obj} {Q : Mor A B} {S : Mor A C}
                          → {R : Mor B C} → R ⊑ Q ╳ S → Q ⨾ R ⊑ S
  ╳-universal-right R⊑Q╳S  =  ⨾-monotone₂ R⊑Q╳S ⟨⊑⊑⟩ ╳-cancel-left
  ╳-universal-left      :   {A B C : Obj} {Q : Mor A B} {S : Mor A C}
                          → {R : Mor B C} → R ⊑ Q ╳ S → R ⨾ S ˘ ⊑ Q ˘
  ╳-universal-left R⊑Q╳S  =  ⨾-monotone₁ R⊑Q╳S ⟨⊑⊑⟩ ╳-cancel-right
  ╳-universal-left˘     :   {A B C : Obj} {Q : Mor A B} {S : Mor A C} {R : Mor B C}
                          → R ⊑ Q ╳ S → S ⨾ R ˘ ⊑ Q
  ╳-universal-left˘ R⊑Q╳S  =  ˘-involutionRightConv ⟨≈˘⊑⟩ ˘-⊑-swap (╳-universal-left R⊑Q╳S)
  ╳-cancel-right˘   :   {A B C : Obj} {Q : Mor A B} {S : Mor C A} → (Q ╳ S ˘) ⨾ S ⊑ Q ˘
  ╳-cancel-right˘  =  ⨾-cong₂ ˘˘ ⟨≈˘⊑⟩ ╳-cancel-right
  ╳-˘-⊑ : {A B C : Obj} {Q : Mor A B} {S : Mor A C} → (Q ╳ S) ˘ ⊑ S ╳ Q
  ╳-˘-⊑ {A} {B} {C} {Q} {S}  =  ╳-universal
    (⊑-begin
          S ⨾ (Q ╳ S) ˘
      ≈˘⟨ ˘-involutionRightConv ⟩
          ((Q ╳ S) ⨾ S ˘) ˘
      ⊑⟨ ˘-monotone ╳-cancel-right ⟩
          Q ˘ ˘
      ≈⟨ ˘˘ ⟩
          Q
    □)
```

```
    (⊑-begin
          (Q ╲ S) ˘ ⨾ Q ˘
      ≈˘⟨ ˘-involution ⟩
          (Q ⨾ (Q ╲ S)) ˘
      ⊑⟨ ˘-monotone ╲-cancel-left ⟩
          S ˘
    □)
╲-˘ : {A B C : Obj} {Q : Mor A B} {S : Mor A C} → (Q ╲ S) ˘ ≈ S ╲ Q
╲-˘ = ⊑-antisym ╲-˘-⊑ (˘˘ ⟨≈˘⊑⟩ ˘-monotone ╲-˘-⊑)
╲-cancel-inner  :         {A B C Z : Obj} {Q : Mor A B} {S : Mor A C} {P : Mor Z A}
                        → Q ╲ S ⊑ (P ⨾ Q) ╲ (P ⨾ S)
╲-cancel-inner {_} {_} {_} {_} {Q} {S} {P} = ╲-universal
    (⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ ╲-cancel-left)
    (⊑-begin
          (Q ╲ S) ⨾ (P ⨾ S) ˘
      ≈⟨ ⨾-cong₂ ˘-involution ⟩
          (Q ╲ S) ⨾ S ˘ ⨾ P ˘
      ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ╲-cancel-right ⟩
          Q ˘ ⨾ P ˘
      ≈˘⟨ ˘-involution ⟩
          (P ⨾ Q) ˘
    □)
╲-cancel-middle  :    {A B C D : Obj} {Q : Mor A B} {S : Mor A C} {T : Mor A D}
                        → (Q ╲ S) ⨾ (S ╲ T) ⊑ Q ╲ T
╲-cancel-middle {_} {_} {_} {_} {Q} {S} {T} = ╲-universal
    (⊑-begin
          Q ⨾ (Q ╲ S) ⨾ (S ╲ T)
      ≈⟨ ⨾-assocL ⟩
          (Q ⨾ (Q ╲ S)) ⨾ (S ╲ T)
      ⊑⟨ ⨾-monotone₁ ╲-cancel-left ⟩
          S ⨾ (S ╲ T)
      ⊑⟨ ╲-cancel-left ⟩
          T
    □)
    (⊑-begin
          ((Q ╲ S) ⨾ (S ╲ T)) ⨾ T ˘
      ≈⟨ ⨾-assoc ⟩
          (Q ╲ S) ⨾ (S ╲ T) ⨾ T ˘
      ⊑⟨ ⨾-monotone₂ ╲-cancel-right ⟩
          (Q ╲ S) ⨾ S ˘
      ⊑⟨ ╲-cancel-right ⟩
          Q ˘
    □)
╲-isDifunctional : {A B C : Obj} {Q : Mor A B} {S : Mor A C} → isDifunctional (Q ╲ S)
╲-isDifunctional {A} {B} {C} {Q} {S} = ⊑-begin
          (Q ╲ S) ⨾ (Q ╲ S) ˘ ⨾ (Q ╲ S)
      ≈⟨ ⨾-cong₂₁ ╲-˘ ⟩
          (Q ╲ S) ⨾ (S ╲ Q) ⨾ (Q ╲ S)
      ⊑⟨ ⨾-monotone₂ ╲-cancel-middle ⟩
          (Q ╲ S) ⨾ (S ╲ S)
      ⊑⟨ ╲-cancel-middle ⟩
          Q ╲ S
    □
╲-surjective-cancel-left          :    {A B C : Obj} {Q : Mor A B} {S : Mor A C}
                                     → isSurjective (Q ╲ S) → Q ⨾ (Q ╲ S) ≈ S
╲-surjective-cancel-left {_} {_} {_} {Q} {S} isSurj = ⊑-antisym ╲-cancel-left
    (⊑-begin
          S
      ⊑⟨ proj₂ isSurj ⟩
          S ⨾ (Q ╲ S) ˘ ⨾ (Q ╲ S)
      ≈⟨ ⨾-cong₂₁ ╲-˘ ⟩
          S ⨾ (S ╲ Q) ⨾ (Q ╲ S)
      ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ╲-cancel-left ⟩
          Q ⨾ (Q ╲ S)
    □)
╲-total-cancel-right        :    {A B C : Obj} {Q : Mor A B} {S : Mor A C}
                               → isTotal (Q ╲ S) → (Q ╲ S) ⨾ S ˘ ≈ Q ˘
╲-total-cancel-right {_} {_} {_} {Q} {S} isTot = ⊑-antisym ╲-cancel-right
    (⊑-begin
          Q ˘
      ⊑⟨ proj₁ isTot ⟨⊑≈⟩ ⨾-assoc ⟩
          (Q ╲ S) ⨾ (Q ╲ S) ˘ ⨾ Q ˘
      ≈⟨ ⨾-cong₂₁ ╲-˘ ⟩
          (Q ╲ S) ⨾ (S ╲ Q) ⨾ Q ˘
      ⊑⟨ ⨾-monotone₂ ╲-cancel-right ⟩
          (Q ╲ S) ⨾ S ˘
    □)
╲-total-cancel-middle        :    {A B C D : Obj} {Q : Mor A B} {S : Mor A C} {T : Mor A D}
                                → isTotal (Q ╲ S) → (Q ╲ S) ⨾ (S ╲ T) ≈ Q ╲ T
╲-total-cancel-middle {_} {_} {_} {_} {Q} {S} {T} isTot = ⊑-antisym ╲-cancel-middle
    (⊑-begin
          Q ╲ T
      ⊑⟨ proj₁ isTot ⟨⊑≈⟩ ⨾-assoc ⟩
          (Q ╲ S) ⨾ (Q ╲ S) ˘ ⨾ (Q ╲ T)
      ≈⟨ ⨾-cong₂₁ ╲-˘ ⟩
          (Q ╲ S) ⨾ (S ╲ Q) ⨾ (Q ╲ T)
      ⊑⟨ ⨾-monotone₂ ╲-cancel-middle ⟩
          (Q ╲ S) ⨾ (S ╲ T)
    □)
╲-surjective-cancel-middle    :    {A B C D : Obj} {Q : Mor A B} {S : Mor A C} {T : Mor A D}
                                 → isSurjective (S ╲ T) → (Q ╲ S) ⨾ (S ╲ T) ≈ Q ╲ T
╲-surjective-cancel-middle {_} {_} {_} {_} {Q} {S} {T} isSurj = ⊑-antisym ╲-cancel-middle
    (⊑-begin
          Q ╲ T
      ⊑⟨ proj₂ isSurj ⟩
          (Q ╲ T) ⨾ (S ╲ T) ˘ ⨾ (S ╲ T)
      ≈⟨ ⨾-cong₂₁ ╲-˘ ⟩
          (Q ╲ T) ⨾ (T ╲ S) ⨾ (S ╲ T)
      ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ╲-cancel-middle ⟩
          (Q ╲ S) ⨾ (S ╲ T)
    □)

╲-iso-shift-left : {A B C D : Obj} {Q : Mor A C} {S : Mor B D} {T : Mor A B}
                 → isBijective T → isMapping T → Q ╲ (T ⨾ S) ≈ (T ˘ ⨾ Q) ╲ S
╲-iso-shift-left {A} {B} {C} {D} {Q} {S} {T} isBij isMap = let
      idPair : isIdentity (T ⨾ T ˘) × isIdentity (T ˘ ⨾ T)   -- pattern binding impossible?
      idPair = bijMapping-identities isBij isMap
      isIdA : isIdentity (T ⨾ T ˘)
      isIdA = proj₁ idPair
      isIdB : isIdentity (T ˘ ⨾ T)
      isIdB = proj₂ idPair
    in ⊑-antisym (╲-cancel-inner ⟨⊑≈⟩ ╲-cong₂ (⨾-assocL ⟨≈≈⟩ proj₁ isIdB))
                 (╲-cancel-inner ⟨⊑≈⟩ ╲-cong₁ (⨾-assocL ⟨≈≈⟩ proj₁ isIdA))
╲-iso-shift-right       :    {A B C D : Obj} {Q : Mor A C} {S : Mor B D} {T : Mor B A}
                           → isBijective T → isMapping T → (T ⨾ Q) ╲ S ≈ Q ╲ (T ˘ ⨾ S)
╲-iso-shift-right isBij isMap = ╲-cong₁ (⨾-cong₁ ˘˘)
```

```
  ⟨≈˘≈˘⟩ (╳-iso-shift-left (˘-isBijective isMap) (˘-isMapping isBij))

╳-unival-in-left      :    {A B C D : Obj} {Q : Mor C B} {S : Mor C D} {F : Mor A B}
                          → isUnivalent F → (F ⨾ (Q ╳ S)) ⊑ ((Q ⨾ F ˘) ╳ S)
╳-unival-in-left {_} {_} {_} {_} {Q} {S} {F} isUnival = ╳-universal
  (⊑-begin
      (Q ⨾ F ˘) ⨾ F ⨾ (Q ╳ S)
    ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ⨾-assocL ⟩
      Q ⨾ (F ˘ ⨾ F) ⨾ (Q ╳ S)
    ⊑⟨ ⨾-monotone₂ (proj₁ isUnival) ⟩
      Q ⨾ (Q ╳ S)
    ⊑⟨ ╳-cancel-left ⟩
      S
  □)
  (⊑-begin
      (F ⨾ (Q ╳ S)) ⨾ S ˘
    ⊑⟨ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ ╳-cancel-right ⟩
      F ⨾ Q ˘
    ≈˘⟨ ˘-involutionRightConv ⟩
      (Q ⨾ F ˘) ˘
  □)
╳-unival-in-right       :    {A B C D : Obj} {Q : Mor C B} {S : Mor C D} {F : Mor A B}
                          → isUnivalent F → F ˘ ⨾ ((Q ⨾ F ˘) ╳ S) ⊑ (Q ╳ S)
╳-unival-in-right {_} {_} {_} {_} {Q} {S} {F} isUnival = ╳-universal
  (⊑-begin
      Q ⨾ F ˘ ⨾ ((Q ⨾ F ˘) ╳ S)
    ⊑⟨ ⨾-assoc ⟨≈˘⊑⟩ ╳-cancel-left ⟩
      S
    □)
  (⊑-begin
      (F ˘ ⨾ ((Q ⨾ F ˘) ╳ S)) ⨾ S ˘
    ⊑⟨ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ ╳-cancel-right ⟩
      F ˘ ⨾ (Q ⨾ F ˘) ˘
    ≈⟨ ⨾-cong₂ ˘-involutionRightConv ⟩
      F ˘ ⨾ F ⨾ Q ˘
    ⊑⟨ ⨾-assoc ⟨≈˘⊑⟩ proj₁ isUnival ⟩
      Q ˘
    □)
╳-in-left   : {A B C D : Obj} {Q : Mor C B} {S : Mor C D} {F : Mor A B}
              → isMapping F → F ⨾ (Q ╳ S) ≈ (Q ⨾ F ˘) ╳ S
╳-in-left (uni, tot) =
  ⊑-antisym (╳-unival-in-left uni) (swap-⨾-⊑-total˘ tot (╳-unival-in-right uni))


╳-inj-in-right    :        {A B C D : Obj} {Q : Mor A B} {S : Mor A C} {F : Mor C D}
                  →        isInjective F → ((Q ╳ S) ⨾ F) ⊑ (Q ╳ (S ⨾ F))
╳-inj-in-right {_} {_} {_} {_} {Q} {S} {F} isInj = ╳-universal
  (⊑-begin
      Q ⨾ (Q ╳ S) ⨾ F
    ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ╳-cancel-left ⟩
      S ⨾ F
  □)
  (⊑-begin
      ((Q ╳ S) ⨾ F) ⨾ (S ⨾ F) ˘
    ≈⟨ ⨾-cong₂ ˘-involution   ⟨≈≈⟩ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ⨾-assocL ⟩
      (Q ╳ S) ⨾ (F ⨾ F ˘) ⨾ S ˘
    ⊑⟨ ⨾-monotone₂ (proj₁ isInj) ⟩
      (Q ╳ S) ⨾ S ˘
    ⊑⟨ ╳-cancel-right ⟩
      Q ˘
  □)
╳-inj-in-left     :      {A B C D : Obj} {Q : Mor A B} {S : Mor A C} {F : Mor C D}
                         → isInjective F → (Q ╳ (S ⨾ F)) ⨾ F ˘ ⊑ (Q ╳ S)
╳-inj-in-left {_} {_} {_} {_} {Q} {S} {F} isInj = ╳-universal
  (⊑-begin
      Q ⨾ ((Q ╳ (S ⨾ F)) ⨾ F ˘)
    ⊑⟨ ⨾-assoc ⟨≈˘⊑⟩ ⨾-monotone₁ ╳-cancel-left ⟩
      (S ⨾ F) ⨾ F ˘
    ⊑⟨ ⨾-assoc ⟨≈⊑⟩ proj₂ isInj ⟩
      S
    □)
  (⊑-begin
      ((Q ╳ (S ⨾ F)) ⨾ F ˘) ⨾ S ˘
    ≈⟨ ⨾-assoc ⟨≈≈˘⟩ ⨾-cong₂ ˘-involution ⟩
      (Q ╳ (S ⨾ F)) ⨾ (S ⨾ F) ˘
    ⊑⟨ ╳-cancel-right ⟩
      Q ˘
    □)
╳-in-right     : {A B C D : Obj} {Q : Mor A B} {S : Mor A C} {F : Mor C D}
                 → isBijective F → (Q ╳ S) ⨾ F ≈ Q ╳ (S ⨾ F)
╳-in-right (inj, surj) =
  ⊑-antisym (╳-inj-in-right inj) (swap-⨾-⊑-surj˘ surj (╳-inj-in-left inj))
╳-unival-cancel-in-right       :    {A B C D : Obj} {Q : Mor A B} {S : Mor A C} {F : Mor D C}
                                  → isUnivalent F → (Q ╳ (S ⨾ F ˘)) ⨾ F ⊑ (Q ╳ S)
╳-unival-cancel-in-right F-unival = ⨾-cong₂ ˘˘ ⟨≈˘⊑⟩ ╳-inj-in-left (isUnivalentToInjective F-unival)
╳-M-in-right : {A B C D : Obj} {Q : Mor A B} {S : Mor A C} {F : Mor D C}
               → isMapping F → (Q ╳ S) ⨾ F ˘ ≈ Q ╳ (S ⨾ F ˘)
╳-M-in-right F-isMapping = ╳-in-right (˘-isBijective F-isMapping)

noy-⊒-subidentity : {A B : Obj} {Q : Mor A B} {p : Mor B B} → isSubidentity p → p ⊑ (Q ╳ Q)
noy-⊒-subidentity {A} {B} {Q} {p} (left, right) = ╳-universal right left
noy-isSubidentity : {A B : Obj} {Q : Mor A B} → isUnivalent Q → isSurjective Q → isSubidentity (Q ╳ Q)
noy-isSubidentity {A} {B} {Q} isUnival isSurj = ⊑-isSubidentity
  (⊑-begin
      Q ╳ Q
    ⊑⟨ proj₁ isSurj ⟨⊑≈⟩ ⨾-assoc ⟩
      Q ˘ ⨾ Q ⨾ (Q ╳ Q)
    ⊑⟨ ⨾-monotone₂ ╳-cancel-left ⟩
      Q ˘ ⨾ Q
  □)
  isUnival
symTrans╳ : {A : Obj} {Q : Mor A A} → IsSymmetric Q → IsTransitive Q → Q ⊑ Q ˘ ╳ Q
symTrans╳ {A} {Q} isSym isTrans = ╳-universal
  (⊑-begin
      Q ˘ ⨾ Q
    ≈⟨ ⨾-cong₁ isSym ⟩
      Q ⨾ Q
    ⊑⟨ isTrans ⟩
      Q
  □)
  (⊑-begin
      Q ⨾ Q ˘
    ≈⟨ ⨾-cong₂ isSym ⟩
      Q ⨾ Q
    ⊑⟨ isTrans ⟩
      Q
    ≈˘⟨ ˘˘ ⟩
```

```
          Q ˘ ˘
    □)
  ╳symTrans :     {A : Obj} {Q : Mor A A}
                → isCodifunctional Q → Q ⊑ Q ˘ ╳ Q → IsSymmetric Q × IsTransitive Q
  ╳symTrans {_} {Q} isCodifun Q⊑Q˘╳Q = let
      left : Q ˘ ⨾ Q ⊑ Q
      left = ╳-universal-right Q⊑Q˘╳Q
      left˘ : Q ˘ ⨾ Q ⊑ Q ˘
      left˘ = ˘-involutionLeftConv ⟨≈˘⊑⟩ ˘-monotone left
      right : Q ⨾ Q ˘ ⊑ Q
      right = ╳-universal-left Q⊑Q˘╳Q ⟨⊑≈⟩ ˘˘
      right˘ : Q ⨾ Q ˘ ⊑ Q ˘
      right˘ = ˘-involutionRightConv ⟨≈˘⊑⟩ ˘-monotone right
      sym˘ : Q ⊑ Q ˘
      sym˘ = ⊑-begin
                Q
        ⊑⟨ isCodifun ⟩
                Q ⨾ Q ˘ ⨾ Q
        ⊑⟨ ⨾-monotone₂ left˘ ⟩
                Q ⨾ Q ˘
        ⊑⟨ right˘ ⟩
                Q ˘
        □
      sym : Q ˘ ⊑ Q
      sym = ˘-⊑-swap sym˘
      trans = ⊑-begin
                Q ⨾ Q
        ⊑⟨ ⨾-monotone₂ sym˘ ⟩
                Q ⨾ Q ˘
        ⊑⟨ right ⟩
                Q
        □
    in isSymmetric⊑ sym, trans
  inj-╳-inj : {A B C : Obj} {Q : Mor A B} {S : Mor A C} → isInjective Q → isInjective S → Q ˘ ⨾ S ⊑ Q ╳ S
  inj-╳-inj {A} {B} {C} {Q} {S} isInjQ isInjS = ╳-universal
    (⊑-begin
        Q ⨾ Q ˘ ⨾ S
      ⊑⟨ ⨾-assocL  ⟨≈⊑⟩ proj₁ isInjQ ⟩
        S
    □)
    (⊑-begin
        (Q ˘ ⨾ S) ⨾ S ˘
      ⊑⟨ ⨾-assoc ⟨≈⊑⟩ proj₂ isInjS ⟩
        Q ˘
    □)


retractSyqOp :  {i₁ i₂ j k₁ k₂ : Level} {Obj₁ : Set i₁} {Obj₂ : Set i₂}
             → (F : Obj₂ → Obj₁)
             → {base : OSGC j k₁ k₂ Obj₁}
             → SyqOp base → SyqOp (retractOSGC F base)
retractSyqOp F syqOp = let open SyqOp syqOp in record
  { _╳_ = _╳_
  ; ╳-cong = ╳-cong
  ; ╳-cancel-left = ╳-cancel-left
  ; ╳-cancel-right = ╳-cancel-right
  ; ╳-universal = ╳-universal
  }
```

3.5 Categoric.OSGC.SyQ.WithResiduals

```
module SyQ-ResidualProps {i j k₁ k₂ : Level} {Obj : Set i} (osgc : OSGC j k₁ k₂ Obj)
                         (let open OSGC osgc)
                         (leftResOp  : LeftResOp orderedSemigroupoid)
                         (rightResOp : RightResOp orderedSemigroupoid)
                         (syqOp : SyqOp osgc)
  where
  open SyqOp syqOp
  open ResidualOps leftResOp rightResOp
  open OSGC-Residuals osgc leftResOp rightResOp
```

Where both symmetric quotients as directly characterised in Categoric.OSGC.SyQ (Sect. 3.4) and residuals are available, the symmetric quotient Q ╳ S actually is the meet of the two residuals Q \ S and Q ˘ / S ˘, even though not all meets may exist in Mor B C.

Due to the two converses in the "left-residual side" of the symmetric quotient definition, it is useful to have specialied variants of the corresponding inclusion.

```
  ╳-⊑-\ : {A B C : Obj} {Q : Mor A B} {S : Mor A C} → Q ╳ S ⊑ Q \ S
  ╳-⊑-\ = \-universal ╳-cancel-left
  ╳-⊑-/ : {A B C : Obj} {Q : Mor A B} {S : Mor A C} → Q ╳ S ⊑ Q ˘ / S ˘
  ╳-⊑-/ = /-universal ╳-cancel-right
  ╳-⊑-\˘ : {A B C : Obj} {Q : Mor A B} {S : Mor A C} → Q ╳ S ⊑ (S \ Q) ˘
  ╳-⊑-\˘ = ╳-⊑-/ ⟨⊑≈˘⟩ \-˘
  ˘╳-⊑-/ : {A B C : Obj} {Q : Mor B A} {S : Mor A C} → Q ˘ ╳ S ⊑ Q / S ˘
  ˘╳-⊑-/ = ╳-⊑-/ ⟨⊑≈⟩ /-cong₁ ˘˘
  ╳˘-⊑-/ : {A B C : Obj} {Q : Mor A B} {S : Mor C A} → Q ╳ S ˘ ⊑ Q ˘ / S
  ╳˘-⊑-/ = ╳-⊑-/ ⟨⊑≈⟩ /-cong₂ ˘˘
  ˘╳˘-⊑-/ : {A B C : Obj} {Q : Mor B A} {S : Mor C A} → Q ˘ ╳ S ˘ ⊑ Q / S
  ˘╳˘-⊑-/ = ╳-⊑-/ ⟨⊑≈⟩ /-cong ˘˘ ˘˘

  ⊑╳-from-\, / : {A B C : Obj} {Q : Mor A B} {S : Mor A C} {R : Mor B C}
                → R ⊑ Q \ S → R ⊑ Q ˘ / S ˘ → R ⊑ Q ╳ S
  ⊑╳-from-\, / R⊑Q\S R⊑Q˘/S˘ = ╳-universal (⨾-monotone₂ R⊑Q\S ⟨⊑⊑⟩ \-cancel-outer)
                                            (⨾-monotone₁ R⊑Q˘/S˘ ⟨⊑⊑⟩ /-cancel-outer)
```

Together, these show that the symmetric quotient is a meet:

```
  open LocOrdMeet Hom
  ╳-isMeet : {A B C : Obj} {Q : Mor A B} {S : Mor A C}
           → IsMeet (Q \ S) (Q ˘ / S ˘) (Q ╳ S)
  ╳-isMeet = record {bound₁ = ╳-⊑-\; bound₂ = ╳-⊑-/; universal = ⊑╳-from-\, /}
```

The following, ╳⊑\˘╳\˘, is (Furusawa and Kahl, 1998, Lemma 6.9).

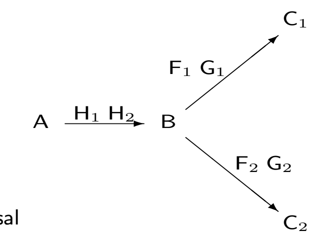
```
  ╳⊑\˘╳\˘ : {A B C : Obj} {Q : Mor A B} {S : Mor A C} → Q ╳ S ⊑ (Q \ S) ˘ ╳ (S \ S) ˘
  ╳⊑\˘╳\˘ {A} {B} {C} {Q} {S} = ˘╳˘-universal
    (⊑-begin
      (Q \ S) ˘ ⨾ (Q ╳ S)
    ⊑⟨ ⨾-monotone (⊑-reflexive \-˘) ╳-⊑-/ ⟩
      (S ˘ / Q ˘) ⨾ (Q ˘ / S ˘)
    ⊑⟨ /-cancel-middle ⟨⊑≈˘⟩ \-˘ ⟩
      (S \ S) ˘
    □)
    (⊑-begin
      (Q ╳ S) ⨾ (S \ S)
    ⊑⟨ ⨾-monotone₁ ╳-⊑-\ ⟩
```

```
      (Q \ S) ⨾ (S \ S)
    ⊑⟨ \-cancel-middle ⟩
      Q \ S
    □)

╳-cancel-inner-⊒ : {A B C Z : Obj} {Q : Mor A B} {S : Mor A C} {P : Mor Z A}
                  → isLeftIdentity (P ˘ ⨾ P) → (P ⨾ Q) ╳ (P ⨾ S) ⊑ Q ╳ S
╳-cancel-inner-⊒ {A} {B} {C} {Z} {Q} {S} {P} P-leftId = ╳-universal
  (⊑-begin
      Q ⨾ ((P ⨾ Q) ╳ (P ⨾ S))
    ⊑⟨ ⨾-monotone₂ ╳-⊑-\ ⟩
      Q ⨾ ((P ⨾ Q) \ (P ⨾ S))
    ⊑⟨ \-cancel-⨾-inner ⟩
      P \ (P ⨾ S)
    ⊑⟨ \-cancel-inner-⊑ P-leftId ⟩
      S
    □)
  (˘-involutionRightConv ⟨≈˘⊑⟩ ˘-monotone (⊑-begin
      S ⨾ ((P ⨾ Q) ╳ (P ⨾ S)) ˘
    ⊑⟨ ⨾-monotone₂ (╳-˘ ⟨≈⊑⟩ ╳-⊑-\) ⟩
      S ⨾ ((P ⨾ S) \ (P ⨾ Q))
    ⊑⟨ \-cancel-⨾-inner ⟩
      P \ (P ⨾ Q)
    ⊑⟨ \-cancel-inner-⊑ P-leftId ⟩
      Q
    □))

\-cancel-inner-⊑-precise : {A B C : Obj} {T : Mor B C} {S : Mor A B}
  → S \ (S ⨾ T) ⊑ (S ˘ ⨾ S) ⨾ (S \ (S ⨾ T))
  → (S ˘ ⨾ S) ⨾ T ⊑ T
  → S \ (S ⨾ T) ⊑ T
\-cancel-inner-⊑-precise {T = T} {S} incl1 incl2 = ⊑-begin
    S \ (S ⨾ T)
  ⊑⟨ incl1 ⟨⊑≈⟩ ⨾-assoc ⟩
    S ˘ ⨾ S ⨾ (S \ (S ⨾ T))
  ⊑⟨ ⨾-monotone₂ \-cancel-outer ⟩
    S ˘ ⨾ S ⨾ T
  ⊑⟨ ⨾-assocL ⟨≈⊑⟩ incl2 ⟩
    T
  □

╳-cancel-inner-⊒-precise : {A B C Z : Obj} {Q : Mor A B} {S : Mor A C} {P : Mor Z A}
  → P \ (P ⨾ S) ⊑ (P ˘ ⨾ P) ⨾ (P \ (P ⨾ S))
  → (P ˘ ⨾ P) ⨾ S ⊑ S
  → P \ (P ⨾ Q) ⊑ (P ˘ ⨾ P) ⨾ (P \ (P ⨾ Q))
  → (P ˘ ⨾ P) ⨾ Q ⊑ Q
    → (P ⨾ Q) ╳ (P ⨾ S) ⊑ Q ╳ S
╳-cancel-inner-⊒-precise {A} {B} {C} {Z} {Q} {S} {P} incl1 incl2 incl3 incl4 = ╳-universal
  (⊑-begin
      Q ⨾ ((P ⨾ Q) ╳ (P ⨾ S))
    ⊑⟨ ⨾-monotone₂ ╳-⊑-\ ⟩
      Q ⨾ ((P ⨾ Q) \ (P ⨾ S))
    ⊑⟨ \-cancel-⨾-inner ⟩
      P \ (P ⨾ S)
    ⊑⟨ \-cancel-inner-⊑-precise incl1 incl2 ⟩
      S
    □)
  (˘-involutionRightConv ⟨≈˘⊑⟩ ˘-monotone (⊑-begin
      S ⨾ ((P ⨾ Q) ╳ (P ⨾ S)) ˘
    ⊑⟨ ⨾-monotone₂ (╳-˘ ⟨≈⊑⟩ ╳-⊑-\) ⟩
      S ⨾ ((P ⨾ S) \ (P ⨾ Q))
    ⊑⟨ \-cancel-⨾-inner ⟩
      P \ (P ⨾ Q)
    ⊑⟨ \-cancel-inner-⊑-precise incl3 incl4 ⟩
      Q
    □))
```




```
retract╳ : {A B C₁ C₂ : Obj}
           {F₁ G₁ : Mor B C₁} {F₂ G₂ : Mor B C₂}
           {H₁ H₂ : Mor A B}
       → F₁ ⊑ G₁
       → F₂ ⊑ G₂
       → H₁ ⨾ G₂ ⨾ F₂ ˘ ⊑ H₂
       → F₁ ⨾ G₁ ˘ ⨾ H₂ ˘ ⊑ H₁ ˘
       → F₁ ⨾ (G₁ ╳ G₂) ⨾ F₂ ˘ ⊑ H₁ ╳ H₂
retract╳ {A} {B} {C₁} {C₂} {F₁} {G₁} {F₂} {G₂} {H₁} {H₂}
       F₁⊑G₁ F₂⊑G₂ H₁⨾G₂⨾F₂˘⊑H₂ F₁⨾G₁˘⨾H₂˘⊑H₁˘ = ╳-universal
         (⊑-begin
           H₁ ⨾ F₁ ⨾ (G₁ ╳ G₂) ⨾ F₂ ˘
         ⊑⟨ ⨾-monotone₂₁ F₁⊑G₁ ⟩
           H₁ ⨾ G₁ ⨾ (G₁ ╳ G₂) ⨾ F₂ ˘
         ⊑⟨ ⨾-monotone₂ (⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ╳-cancel-left) ⟩
           H₁ ⨾ G₂ ⨾ F₂ ˘
         ⊑⟨ H₁⨾G₂⨾F₂˘⊑H₂ ⟩
           H₂
         □)
         (⊑-begin
           (F₁ ⨾ (G₁ ╳ G₂) ⨾ F₂ ˘) ⨾ H₂ ˘
         ⊑⟨ ⨾-monotone₁ (⨾-monotone₂₂ (˘-monotone F₂⊑G₂)) ⟩
           (F₁ ⨾ (G₁ ╳ G₂) ⨾ G₂ ˘) ⨾ H₂ ˘
         ⊑⟨ ⨾-monotone₁₂ ╳-cancel-right ⟨⊑≈⟩ ⨾-assoc ⟩
           F₁ ⨾ G₁ ˘ ⨾ H₂ ˘
         ⊑⟨ F₁⨾G₁˘⨾H₂˘⊑H₁˘ ⟩
           H₁ ˘
         □)
```

3.6 Categoric.OCC.SyQ

```
module OCC-SyQ-Props {i j k₁ k₂ : Level} {Obj : Set i}
                     (occ : OCC j k₁ k₂ Obj)
                     (let open OCC occ)
                     (syqOp : SyqOp osgc)
  where
  open SyqOp syqOp
  noy-isReflexive : {A B : Obj} {R : Mor A B} → Id ⊑ R ╳ R
  noy-isReflexive = ╳-universal (⊑-reflexive rightId) (⊑-reflexive leftId)
  noy-isCoreflexive : {A B : Obj} {R : Mor A B} → isUnivalentI R → isSurjectiveI R → R ╳ R ⊑ Id
  noy-isCoreflexive {A} {B} {R} R-unival R-surj = ⊑-begin
      R ╳ R
    ⊑⟨ leftId ⟨≈˘⊑⟩ ⨾-monotone₁ R-surj ⟨⊑≈⟩ ⨾-assoc ⟩
      R ˘ ⨾ R ⨾ (R ╳ R)
    ⊑⟨ ⨾-monotone₂ ╳-cancel-left ⟩
      R ˘ ⨾ R
    ⊑⟨ R-unival ⟩
```

```
      Id
    □

noy-unival-surj-≈Id : {A B : Obj} {R : Mor A B} → isUnivalentI R → isSurjectiveI R → R ⨾ R ≈ Id
noy-unival-surj-≈Id R-unival R-surj = ⊑-antisym (noy-isCoreflexive R-unival R-surj) noy-isReflexive

noy-Id : {A : Obj} → Id ⨾ Id ≈ Id {A}
noy-Id = noy-unival-surj-≈Id (⊑-reflexive rightId ⟨⊑≈⟩ Id˘) (Id˘ ⟨≈˘⊑⟩ ⊑-reflexive′ rightId)
```

4 Power Operators

One way to abstractly deal with element relations is that using "power transpose" as presented for example by Bird and de Moor (1997). We formalise this here directly in the setting of OSGCs. Adding also residuals to that setting is sufficient for the formalisation of the *polarities* (Sect. 4.4) needed for formal concept analysis. This chapter (together with Chapter 7) underlies the publication (Kahl, 2014a).

4.1 Categoric.OSGC.PowerOp

We assume a base OSGC, and make the standard names available for all basic OSGC material:

```
module Categoric.OSGC.PowerOp {i j k₁ k₂ : Level} {Obj : Set i} (osgc : OSGC j k₁ k₂ Obj) where
open OSGC osgc
```

For the induced semigroupoid of mappings in osgc, we make names with subscript "$_1$" available:

```
open Semigroupoid₁ (MapSG osgc)
```

Before defining power allegories as a special kind of division allegories, Freyd and Scedrov (1990, Sect. 2.4) give an alternative definition that (if recast in a typed setting) enriches allegories with a type operator ℙ : Obj → Obj, a transformation ∋ : {B : Obj} → Mor (ℙ B) B, and an operator Λ : {A B : Obj} → Mor A B → Mapping A (ℙ B). That definition can be completely expressed in OSGCs — we start by giving a version that considers only a single power object ℙB for B, and uses ∊ for the element relation, that is, the converse of the ∋ used by Freyd and Scedrov (1990); it appears that we need to additionally assume Λ-cong:

```
record IsPowerFS {B ℙB : Obj} (∊ : Mor B ℙB)
                 {A : Obj}     (Λ : Mor A B → Mapping A ℙB) : Set (i ⊍ j ⊍ k₁ ⊍ k₂) where
  field
    Λ⨾∊˘ : {R : Mor A B} → Mapping.mor (Λ R) ⨾ ∊ ˘ ≈ R
    Λ-⨾∊˘ : {f : Mapping A ℙB} → Λ (Mapping.mor f ⨾ ∊ ˘) ≈₁ f
    Λ-cong : {R₁ R₂ : Mor A B} → R₁ ≈ R₂ → Λ R₁ ≈₁ Λ R₂
  ∊⨾Λ˘ : {R : Mor A B} → ∊ ⨾ (Mapping.mor (Λ R)) ˘ ≈ R ˘
  ∊⨾Λ˘ = ˘-involutionRightConv ⟨≈˘≈⟩ ˘-cong Λ⨾∊˘
```

Bird and de Moor (1997, Sect. 4.6) choose a different presentation of essentially the same definition, namely as a single equivalence:

$$\{R : \mathsf{Mor}\ A\ B\}\ \{f : \mathsf{Mapping}\ A\ \mathbb{P}B\} \rightarrow (f \approx_1 \Lambda\ R \leftrightarrow \mathsf{Mapping.mor}\ f \fatsemi \in\ \breve{}\ \approx R)$$

We follow Bird and de Moor in naming Λ "power transpose"; we present the equivalence as two implications in the **field**s below — the initial four definitions are just the explicit components of the *mapping* constraint on Λ R, for any R : Mor A B.

```
record IsPowerTranspose {B ℙB : Obj} (∊ : Mor B ℙB)
                        {A : Obj}     (Λ : Mor A B → Mapping A ℙB) : Set (i ⊍ j ⊍ k₁ ⊍ k₂) where
  Λ₀ : Mor A B → Mor A ℙB
  Λ₀ R = Mapping.mor (Λ R)
  Λ-unival : {R : Mor A B} → isUnivalent (Λ₀ R)
  Λ-unival {R} = Mapping.unival (Λ R)
  Λ-total : {R : Mor A B} → isTotal (Λ₀ R)
  Λ-total {R} = Mapping.total (Λ R)
  Λ-mapping : {R : Mor A B} → isMapping (Λ₀ R)
  Λ-mapping {R} = Mapping.prf (Λ R)
  field
    Λ⇒∊ : {R : Mor A B} {f : Mapping A ℙB} → f ≈₁ Λ R → Mapping.mor f ⨾ ∊ ˘ ≈ R
    ∊⇒Λ : {R : Mor A B} {f : Mapping A ℙB} → Mapping.mor f ⨾ ∊ ˘ ≈ R → f ≈₁ Λ R
```

This definition is in fact equivalent to the version of Freyd and Scedrov (1990):

```
  isPowerFS : IsPowerFS ∊ Λ
  isPowerFS = record
    {Λ⨾∊˘ = λ {R} → Λ⇒∊ {R} {Λ R} ≈-refl
    ; Λ-⨾∊˘ = λ {f} → ≈-sym (∊⇒Λ {Mapping.mor f ⨾ ∊ ˘} {f} ≈-refl)
    ; Λ-cong = λ {R₁} {R₂} R₁≈R₂ → ∊⇒Λ {R₂} {Λ R₁} (≈-begin
          Λ₀ R₁ ⨾ ∊ ˘
        ≈⟨ Λ⇒∊ {R₁} {Λ R₁} ≈-refl ⟩
          R₁
        ≈⟨ R₁≈R₂ ⟩
          R₂
        □)
    }
  open IsPowerFS isPowerFS public
  ˘⨾Λ : {R : Mor A B} → R ˘ ⨾ Λ₀ R ⊑ ∊
  ˘⨾Λ {R} = ⊑-begin
        R ˘ ⨾ Λ₀ R
      ≈⟨ ⨾-cong₁ (˘-cong Λ⨾∊˘ ⟨≈˘≈⟩ ˘-involutionRightConv) ⟨≈≈⟩ ⨾-assoc ⟩
        ∊ ⨾ Λ₀ R ˘ ⨾ Λ₀ R
      ⊑⟨ proj₂ Λ-unival ⟩
        ∊
      □
  ⨾Λ-˘ : {Q : Mor B A} → Q ⨾ Λ₀ (Q ˘) ⊑ ∊
  ⨾Λ-˘ = ⨾-cong₁ ˘˘ ⟨≈˘⊑⟩ ˘⨾Λ
```

Conversely:

```
fromPowerFS : {A B ℙB : Obj} {∊ : Mor B ℙB} {Λ : Mor A B → Mapping A ℙB}
            → (isPowerFS : IsPowerFS ∊ Λ) → IsPowerTranspose ∊ Λ
fromPowerFS isPowerFS = record
  {Λ⇒∊ = λ {R} {f} f≈ΛR → ⨾-cong₁ f≈ΛR ⟨≈≈⟩ Λ⨾∊˘
  ; ∊⇒Λ = λ {R} {f} f⨾∊˘≈R → Λ-⨾∊˘ {f} ⟨≈˘≈⟩ Λ-cong f⨾∊˘≈R
  }
  where open IsPowerFS isPowerFS
```

A power object has power transposes for morphisms starting at any object:

```
record IsPower {B ℙB : Obj} (∊ : Mor B ℙB) : Set (i ⊍ j ⊍ k₁ ⊍ k₂) where
  field
    Λ : {A : Obj} → Mor A B → Mapping A ℙB
    isPowerTranspose : (A : Obj) → IsPowerTranspose ∊ {A} Λ
  open module isPowerTranspose {A : Obj} = IsPowerTranspose (isPowerTranspose A) public
  Idℙ : Mapping ℙB ℙB
  Idℙ = Λ (∊ ˘)
  Idℙ₀ : Mor ℙB ℙB
  Idℙ₀ = Mapping.mor Idℙ
```

If there is an identity on ℙB, then $\mathsf{Id}\mathbb{P}_0$ is that identity:

```
IdℙId : {I : Mor ℙB ℙB} → isIdentity I → Idℙ₀ ≈ I
IdℙId {I} I-isId = ≈-begin
    Λ₀ (∈ ˘)
  ≈˘⟨ Λ-cong (proj₁ I-isId) ⟩
    Λ₀ (I ⨾ ∈ ˘)
  ≈⟨ Λ-⨾∈˘ {f = isIdentity-Mapping I-isId} ⟩
    I
  □
```

In any case, Idℙ is a right-identity for mappings:

```
rightIdℙ : {A : Obj} {f : Mapping A ℙB} → f ⨾₁ Idℙ ≈₁ f
rightIdℙ {A} {f} = ≈₁-begin
    f ⨾₁ Idℙ
  ≈₁⟨ Λ-⨾∈˘ {f = f ⨾₁ Idℙ} ⟨≈˘≈⟩ Λ-cong ⨾-assoc ⟩
    Λ (Mapping.mor f ⨾ Λ₀ (∈ ˘) ⨾ ∈ ˘)
  ≈₁⟨ Λ-cong (⨾-cong₂ Λ⨾∈˘) ⟩
    Λ (Mapping.mor f ⨾ ∈ ˘)
  ≈₁⟨ Λ-⨾∈˘ {f = f} ⟩
    f
  □₁
```

```
map-Λ : {A C : Obj} {R : Mor A B} {f : Mapping C A} → f ⨾₁ Λ R ≈₁ Λ (Mapping.mor f ⨾ R)
map-Λ {A} {C} {R} {f} = ∈⇒Λ {C} {Mapping.mor f ⨾ R} {f ⨾₁ Λ R} (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘)
```

Power objects are unique up to isomorphism:

```
module IsPower-iso (B : Obj) {ℙB₁ ℙB₂ : Obj} (∈₁ : Mor B ℙB₁) (∈₂ : Mor B ℙB₂)
    (𝒫₁ : IsPower ∈₁)
    (𝒫₂ : IsPower ∈₂)
  where
  private
    module 𝒫₁ = IsPower 𝒫₁
    module 𝒫₂ = IsPower 𝒫₂

  to : Mapping ℙB₁ ℙB₂
  to = 𝒫₂.Λ (∈₁ ˘)

  from : Mapping ℙB₂ ℙB₁
  from = 𝒫₁.Λ (∈₂ ˘)

  to⨾from : to ⨾₁ from ≈₁ 𝒫₁.Idℙ
  to⨾from = 𝒫₁.∈⇒Λ {f = 𝒫₂.Λ (∈₁ ˘) ⨾₁ 𝒫₁.Λ (∈₂ ˘)} (≈-begin
      (𝒫₂.Λ₀ (∈₁ ˘) ⨾ 𝒫₁.Λ₀ (∈₂ ˘)) ⨾ ∈₁ ˘
    ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ 𝒫₁.Λ⨾∈˘ ⟩
      𝒫₂.Λ₀ (∈₁ ˘) ⨾ ∈₂ ˘
    ≈⟨ 𝒫₂.Λ⨾∈˘ ⟩
      ∈₁ ˘
    □)
```

A power operator provides for any object A a pair $(\mathbb{P}A, \in_A)$ with a proof that this pair is a power object of A:

```
record PowerOp : Set (i ⊍ j ⊍ k₁ ⊍ k₂) where
  field
    ℙ : Obj → Obj
    ∈ : {A : Obj} → Mor A (ℙ A)
    isPower : {A : Obj} → IsPower (∈ {A})
  open module Power {A : Obj} = IsPower (isPower {A}) public
```

In the context of a PowerOp, a "power order" is an indexed relation on power objects satisfying conditions appropriate for a "subset relation":

```
record IsPowerOrder (Ω : {A : Obj} → Mor (ℙ A) (ℙ A)) : Set (i ⊍ j ⊍ k₁ ⊍ k₂) where
  field
    ∈⨾Ω : {A : Obj} → ∈ ⨾ Ω {A} ⊑ ∈
    Ω-universal : {A : Obj} {R : Mor (ℙ A) (ℙ A)} → ∈ ⨾ R ⊑ ∈ → R ⊑ Ω

  Ω˘-universal : {A : Obj} {R : Mor (ℙ A) (ℙ A)} → R ⨾ ∈ ˘ ⊑ ∈ ˘ → R ⊑ Ω ˘
  Ω˘-universal {A} {R} R⨾∈˘⊑∈˘ = ⊑-˘-swap (Ω-universal (⊑-begin
      ∈ ⨾ R ˘
    ≈˘⟨ ˘-involutionRightConv ⟩
      (R ⨾ ∈ ˘) ˘
    ⊑⟨ ˘-⊑-swap R⨾∈˘⊑∈˘ ⟩
      ∈
    □))
```

4.2 Categoric.OSGC.PowerOrder

```
module Categoric.OSGC.PowerOrder {i j k₁ k₂} {Obj : Set i} (osgc : OSGC j k₁ k₂ Obj)
  (leftResOp  : LeftResOp (OSGC.orderedSemigroupoid osgc))
  (rightResOp : RightResOp (OSGC.orderedSemigroupoid osgc))
  (powerOp : PowerOp osgc) where
open OSGC osgc
open ResidualOps leftResOp rightResOp
open OSGC-Residuals osgc leftResOp rightResOp
open PowerOp osgc powerOp
```

In the presence of residuals, a power order is easily defined:

```
Ω : {A : Obj} → Mor (ℙ A) (ℙ A)
Ω = ∈ \ ∈

isPowerOrder : IsPowerOrder Ω
isPowerOrder = record
  {∈⨾Ω = \-cancel-outer
  ;Ω-universal = λ {A} {R} ∈⨾R⊑∈ → \-universal ∈⨾R⊑∈
  }
open IsPowerOrder isPowerOrder
```

This is transitive and "as reflexive as can be defined" in the context of OSGCs with power operator:

```
Ω-trans : {A : Obj} → Ω ⨾ Ω ⊑ Ω {A}
Ω-trans = \-cancel-middle

Idℙ⊑Ω : {A : Obj} → Idℙ₀ ⊑ Ω {A}
Idℙ⊑Ω = \-universal (⊑-begin
    ∈ ⨾ Λ₀ (∈ ˘)
  ⊑⟨ ⨾Λ-˘ ⟩
    ∈
  □)

Λ₀⨾Ω˘ : {A B : Obj} {R : Mor A B} → Λ₀ R ⨾ Ω ˘ ≈ R / ∈ ˘
Λ₀⨾Ω˘ {R = R} = ≈-begin
    Λ₀ R ⨾ (∈ \ ∈) ˘
  ≈⟨ ⨾-cong₂ \-˘ ⟩
    Λ₀ R ⨾ (∈ ˘ / ∈ ˘)
```

```
  ≈⟨ /-outer-⨾-≈ Λ-mapping ⟩
    (Λ₀ R ⨾ ∈ ˘) / ∈ ˘
  ≈⟨ /-cong₁ Λ⨾∈˘ ⟩
    R / ∈ ˘
  □
```

```
open import Categoric.MapSG
open Semigroupoid₁ (MapSG osgc)

Lub : {X A : Obj} (R : Mor X (ℙ A)) → Mapping X (ℙ A)
Lub R = Λ (R ⨾ ∈ ˘)

Lub₀ : {X A : Obj} (R : Mor X (ℙ A)) → Mor X (ℙ A)
Lub₀ R = Mapping.mor (Lub R)

Lub-cong : {X A : Obj} {R₁ R₂ : Mor X (ℙ A)} → R₁ ≈ R₂ → Lub R₁ ≈₁ Lub R₂
Lub-cong R₁≈R₂ = Λ-cong (⨾-cong₁ R₁≈R₂)

Glb : {X A : Obj} (R : Mor X (ℙ A)) → Mapping X (ℙ A)
Glb R = Λ (R ˘ \ ∈ ˘)

Glb-cong : {X A : Obj} {R₁ R₂ : Mor X (ℙ A)} → R₁ ≈ R₂ → Glb R₁ ≈₁ Glb R₂
Glb-cong R₁≈R₂ = Λ-cong (\-cong₁ (˘-cong R₁≈R₂))

Lub-cocontinuous Glb-cocontinuous : {A B : Obj} (f : Mapping (ℙ B) (ℙ A)) → Set (i ⊍ j ⊍ k₁)
Lub-cocontinuous {A} {B} f = ∀ {X} (Q : Mor X (ℙ B)) → Lub Q ⨾₁ f ≈₁ Glb (Q ⨾ Mapping.mor f)
Glb-cocontinuous {A} {B} f = {X : Obj} (Q : Mor X (ℙ B)) → Glb Q ⨾₁ f ≈₁ Lub (Q ⨾ Mapping.mor f)

ℙ-antitone : {A B : Obj} → Mapping (ℙ B) (ℙ A) → Set k₂
ℙ-antitone f = Ω ⨾ Mapping.mor f ⊑ Mapping.mor f ⨾ Ω ˘
```

4.3 Categoric.OSGC.PowerRes

We prove that a power operator together with a power order gives rise to residuals.

```
module Categoric.OSGC.PowerRes {i j k₁ k₂ : Level} {Obj : Set i}
  (osgc : OSGC j k₁ k₂ Obj)
  (powerOp : PowerOp osgc) where
open OSGC osgc
open PowerOp osgc powerOp

module _ (Ω : {A : Obj} → Mor (ℙ A) (ℙ A)) (isPowerOrder : IsPowerOrder Ω) where
  open IsPowerOrder isPowerOrder
```

As motivation, recall that in sets, Ω = ⊆ and ΛR = x ↦ {y | x R y}; then we have:

$$a\ (S\ /\ R)\ b \Leftrightarrow \Lambda S\ a \supseteq \Lambda R\ b \Leftrightarrow a\ (\Lambda S \fatsemi \subseteq \fatsemi \Lambda R\ \breve{})\ b \Leftrightarrow a\ (\Lambda S \fatsemi \Omega\breve{} \fatsemi \Lambda R\ \breve{})\ b$$

```
  leftResOp : LeftResOp orderedSemigroupoid
  leftResOp = record
    {_/_ = λ {A} {B} {C} S R → Λ₀ S ⨾ Ω ˘ ⨾ (Λ₀ R) ˘
    ; /-cancel-outer = λ {A} {B} {C} {S} {R} → ⊑-begin
        (Λ₀ S ⨾ Ω ˘ ⨾ Λ₀ R ˘) ⨾ R
      ≈⟨ ⨾-assoc₃₊₁ ⟨≈≈˘⟩ ⨾-cong₂₂ ˘-involutionLeftConv ⟩
        Λ₀ S ⨾ Ω ˘ ⨾ (R ˘ ⨾ Λ₀ R) ˘
      ⊑⟨ ⨾-monotone₂₂ (˘-monotone ˘⨾Λ) ⟩
        Λ₀ S ⨾ Ω ˘ ⨾ ∈ ˘
      ⊑⟨ ⨾-monotone₂ (˘-involution ⟨≈˘⊑⟩ ˘-monotone ∈⨾Ω) ⟩
        Λ₀ S ⨾ ∈ ˘
      ≈⟨ Λ⨾∈˘ ⟩
        S
      □
    ; /-universal = λ {A} {B} {C} {S} {R} {Q} Q⨾R⊑S → ⊑-begin
        Q
      ⊑⟨ proj₁ Λ-total ⟨⊑≈⟩ ⨾-assoc ⟩
        Λ₀ S ⨾ (Λ₀ S) ˘ ⨾ Q
      ⊑⟨   ⨾-monotone₂₂ (proj₂ Λ-total) ⟩
        Λ₀ S ⨾ (Λ₀ S) ˘ ⨾ Q ⨾ Λ₀ R ⨾ Λ₀ R ˘
      ⊑⟨ ⨾-monotone₂ (⨾-assocL₃₊₁ ⟨≈⊑⟩ ⨾-monotone₁ (Ω˘-universal (⊑-begin
            ((Λ₀ S) ˘ ⨾ Q ⨾ Λ₀ R) ⨾ ∈ ˘
          ≈⟨ ⨾-assoc₃₊₁ ⟨≈≈⟩ ⨾-cong₂₂ Λ⨾∈˘ ⟩
            (Λ₀ S) ˘ ⨾ Q ⨾ R
          ⊑⟨ ⨾-monotone₂ Q⨾R⊑S ⟩
            (Λ₀ S) ˘ ⨾ S
          ⊑⟨ ˘-involutionLeftConv ⟨≈˘⊑⟩ ˘-monotone ˘⨾Λ ⟩
            ∈ ˘
          □))) ⟩
        Λ₀ S ⨾ Ω ˘ ⨾ Λ₀ R ˘
      □
    }

  rightResOp : RightResOp orderedSemigroupoid
  rightResOp = record
    {_\_ = λ {A} {B} {C} Q S → Λ₀ (Q ˘) ⨾ Ω ⨾ (Λ₀ (S ˘)) ˘
    ; \-cancel-outer = λ {A} {B} {C} {S} {Q} → ⊑-begin
        Q ⨾ Λ₀ (Q ˘) ⨾ Ω ⨾ (Λ₀ (S ˘)) ˘
      ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ⨾Λ-˘ ⟩
        ∈ ⨾ Ω ⨾ (Λ₀ (S ˘)) ˘
      ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ∈⨾Ω ⟩
        ∈ ⨾ (Λ₀ (S ˘)) ˘
      ≈⟨ ∈⨾Λ˘ ⟨≈≈⟩ ˘˘ ⟩
        S
      □
    ; \-universal = λ {A} {B} {C} {S} {Q} {R} Q⨾R⊑S → ⊑-begin
        R
      ⊑⟨ proj₁ Λ-total ⟨⊑≈⟩ ⨾-assoc ⟩
        Λ₀ (Q ˘) ⨾ (Λ₀ (Q ˘)) ˘ ⨾ R
      ⊑⟨ ⨾-monotone₂₂ (proj₂ Λ-total) ⟩
        Λ₀ (Q ˘) ⨾ (Λ₀ (Q ˘)) ˘ ⨾ R ⨾ Λ₀ (S ˘) ⨾ (Λ₀ (S ˘)) ˘
      ⊑⟨ ⨾-monotone₂ (⨾-assocL₃₊₁ ⟨≈⊑⟩ ⨾-monotone₁ (Ω-universal (⊑-begin
            ∈ ⨾ (Λ₀ (Q ˘)) ˘ ⨾ R ⨾ Λ₀ (S ˘)
          ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ (∈⨾Λ˘ ⟨≈≈⟩ ˘˘) ⟩
            Q ⨾ R ⨾ Λ₀ (S ˘)
          ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ Q⨾R⊑S ⟨⊑⊑⟩ ⨾Λ-˘ ⟩
            ∈
          □))) ⟩
        Λ₀ (Q ˘) ⨾ Ω ⨾ (Λ₀ (S ˘)) ˘
      □
    }
```

The standard definition of the power order via this right residual returns the given power order Ω:

```
  open RightResOp rightResOp
  Ω-via-\-⊑ : {A : Obj} → ∈ \ ∈ ⊑ Ω {A}
  Ω-via-\-⊑ = Ω-universal (⊑-begin
          ∈ ⨾ Λ₀ (∈ ˘) ⨾ Ω ⨾ (Λ₀ (∈ ˘)) ˘
```

```
      ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ⨾Λ-˘ ⟩
        ∈ ⨾ Ω ⨾ (Λ₀ (∈ ˘)) ˘
      ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ∈⨾Ω ⟩
        ∈ ⨾ (Λ₀ (∈ ˘)) ˘
      ≈⟨ ∈⨾Λ˘ ⟨≈≈⟩ ˘˘ ⟩
        ∈
      □)
  Ω-via-\ : {A : Obj} → ∈ \ ∈ ≈ Ω {A}
  Ω-via-\ = ⊑-antisym Ω-via-\-⊑ (\-universal ∈⨾Ω)
```

4.4 Categoric.OSGC.Power.Polarities

```
module Categoric.OSGC.Power.Polarities {i j k₁ k₂} {Obj : Set i} (osgc : OSGC j k₁ k₂ Obj)
  (leftResOp   : LeftResOp (OSGC.orderedSemigroupoid osgc))
  (rightResOp : RightResOp (OSGC.orderedSemigroupoid osgc))
  (powerOp : PowerOp osgc) where
open OSGC osgc
open ResidualOps leftResOp rightResOp
open OSGC-Residuals osgc leftResOp rightResOp
open PowerOp osgc powerOp
open import Categoric.OSGC.PowerOrder osgc leftResOp rightResOp powerOp
  using (Ω; Λ₀⨾Ω˘; Lub; Lub-cocontinuous; Glb; Glb-cong)
private
  module MapSG = Semigroupoid (MapSG osgc)
open Semigroupoid₁ (MapSG osgc)
```

We define the operators _↑ and _↓ (as postfix operators, these need to be separated from their argument by a space), which in set theory are defined as follows, for $p : \mathbb{P}\,\mathsf{A}$ and $q : \mathbb{P}\,\mathsf{B}$:

$$\mathsf{R}\uparrow p = \{b : \mathsf{B} \mid \forall a \in p\ .\ a\mathsf{R}b\} \qquad \text{and} \qquad \mathsf{R}\downarrow q = \{a : \mathsf{A} \mid \forall b \in q\ .\ a\mathsf{R}b\}$$

```
_↑ : {A B : Obj} → Mor A B → Mapping (ℙ A) (ℙ B)
R ↑ = Λ (∈ \ R)
_↓ : {A B : Obj} → Mor A B → Mapping (ℙ B) (ℙ A)
R ↓ = Λ (∈ \ (R ˘))
↑≈˘↓ : {A B : Obj} {R : Mor A B} → R ↑ ≈₁ R ˘ ↓
↑≈˘↓ = Λ-cong (\-cong₂ (≈-sym ˘˘))
↑-cong : {A B : Obj} {R S : Mor A B} → R ≈ S → R ↑ ≈₁ S ↑
↑-cong R≈S = Λ-cong (\-cong₂ R≈S)
↓-cong : {A B : Obj} {R S : Mor A B} → R ≈ S → R ↓ ≈₁ S ↓
↓-cong R≈S = Λ-cong (\-cong₂ (˘-cong R≈S))

_↑₀ : {A B : Obj} → Mor A B → Mor (ℙ A) (ℙ B)
R ↑₀ = Mapping.mor (R ↑)
_↓₀ : {A B : Obj} → Mor A B → Mor (ℙ B) (ℙ A)
R ↓₀ = Mapping.mor (R ↓)
```

The fact that the operators _↑ and _↓ produce Galois connections, that is, that for each R, the two mappings R ↓ and R ↑ form a Galois connection between the orders Ω {A} and Ω {B}, set-theoretically

$$p \subseteq \mathsf{R}\downarrow q \qquad \Leftrightarrow \qquad q \subseteq \mathsf{R}\uparrow p \qquad\qquad \text{for all } p : \mathbb{P}\,\mathsf{A} \text{ and } q : \mathbb{P}\,\mathsf{B},$$

can now be stated as a simple morphism equality and shown by algebraic calculation using residual and power properties:

```
Galois-↓-↑ : {A B : Obj} {R : Mor A B} → Ω ⨾ (R ↓₀) ˘ ≈ R ↑₀ ⨾ Ω ˘
Galois-↓-↑ {A} {B} {R} = ≈-begin
    Ω ⨾ Λ₀ (∈ \ R ˘) ˘
  ≈˘⟨ ˘-involutionRightConv ⟩
    (Λ₀ (∈ \ R ˘) ⨾ Ω ˘) ˘
  ≈⟨ ˘-cong Λ₀⨾Ω˘ ⟩
    ((∈ \ R ˘) / ∈ ˘) ˘
  ≈⟨ /˘-˘ ⟩
    ∈ \ (∈ \ R ˘) ˘
  ≈⟨ \-cong₂ \˘-˘ ⟩
    ∈ \ (R / ∈ ˘)
  ≈⟨ \/-≈ ⟩
    (∈ \ R) / ∈ ˘
  ≈˘⟨ Λ₀⨾Ω˘ ⟩
    Λ₀ (∈ \ R) ⨾ Ω ˘
  □
```

The operators _↑ and _↓ are both Lub-cocontinuous:

```
↓-Lub-cocontinuous : {A B : Obj} (R : Mor A B) → Lub-cocontinuous (R ↓)
↓-Lub-cocontinuous R {X} Q = ≈-begin
      Λ₀ (Q ⨾ ∈ ˘) ⨾ Λ₀ (∈ \ (R ˘))
  ≈⟨ ∈⇒Λ {f = Λ (Q ⨾ ∈ ˘) ⨾₁ Λ (∈ \ (R ˘))} (≈-begin
      (Λ₀ (Q ⨾ ∈ ˘) ⨾ Λ₀ (∈ \ (R ˘))) ⨾ ∈ ˘
    ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘ ⟩
      Λ₀ (Q ⨾ ∈ ˘) ⨾ (∈ \ (R ˘))
    ≈⟨ \-inner-⨾ Λ-mapping ⟩
      (∈ ⨾ Λ₀ (Q ⨾ ∈ ˘) ˘) \ (R ˘)
    ≈⟨ \-cong₁ (˘-involutionRightConv ⟨≈˘≈⟩ ˘-cong Λ⨾∈˘) ⟨≈≈˘⟩ /-˘ ⟩
      (R / (Q ⨾ ∈ ˘)) ˘
    ≈⟨ ˘-cong (⊑-antisym
      (/-universal (⊑-begin
          (R / (Q ⨾ ∈ ˘)) ⨾ Q ⨾ Λ₀ (∈ \ R ˘)
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-⨾-inner ⟩
          (R / ∈ ˘) ⨾ Λ₀ (∈ \ R ˘)
        ⊑⟨ ⨾-cong₁ \˘-˘ ⟨≈˘⊑⟩ ˘⨾Λ ⟩
          ∈
        □))
      (/-universal (⊑-begin
          (∈ / (Q ⨾ Λ₀ (∈ \ R ˘))) ⨾ (Q ⨾ ∈ ˘)
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-⨾-inner ⟩
          (∈ / Λ₀ (∈ \ R ˘)) ⨾ ∈ ˘
        ⊑⟨ ⨾-monotone₂ (proj₁ Λ-total ⟨⊑≈⟩ ⨾-assoc) ⟩
          (∈ / Λ₀ (∈ \ R ˘)) ⨾ Λ₀ (∈ \ R ˘) ⨾ Λ₀ (∈ \ R ˘) ˘ ⨾ ∈ ˘
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-outer ⟩
          ∈ ⨾ Λ₀ (∈ \ R ˘) ˘ ⨾ ∈ ˘
        ≈⟨ ⨾-assocL ⟨≈≈˘⟩ ⨾-cong₁ ˘-involutionRightConv ⟩
          (Λ₀ (∈ \ R ˘) ⨾ ∈ ˘) ˘ ⨾ ∈ ˘
        ≈⟨ ⨾-cong₁ (˘-cong Λ⨾∈˘) ⟩
          (∈ \ R ˘) ˘ ⨾ ∈ ˘
        ⊑⟨ ⨾-cong₁ \˘-˘ ⟨≈⊑⟩ /-cancel-outer ⟩
          R
        □))) ⟩
      (∈ / (Q ⨾ Λ₀ (∈ \ R ˘))) ˘
    ≈⟨ /-˘ ⟩
      (Q ⨾ Λ₀ (∈ \ (R ˘))) ˘ \ ∈ ˘
```

```
      □) ⟩
      Λ₀ ((Q ⨾ Λ₀ (∈ \ (R ˘))) ˘ \ ∈ ˘)
  □
```

```
↑-Lub-cocontinuous : {A B : Obj} (R : Mor A B) → Lub-cocontinuous (R ↑)
↑-Lub-cocontinuous R {X} Q = ≈-begin
      Λ₀ (Q ⨾ ∈ ˘) ⨾ Λ₀ (∈ \ R)
  ≈⟨ ∈⇒Λ {f = Λ (Q ⨾ ∈ ˘) ⨾₁ Λ (∈ \ R)} (≈-begin
      (Λ₀ (Q ⨾ ∈ ˘) ⨾ Λ₀ (∈ \ R)) ⨾ ∈ ˘
    ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘ ⟩
      Λ₀ (Q ⨾ ∈ ˘) ⨾ (∈ \ R)
    ≈⟨ \-inner-⨾ Λ-mapping ⟩
      (∈ ⨾ Λ₀ (Q ⨾ ∈ ˘) ˘) \ R
    ≈⟨ \-cong₁ (˘-involutionRightConv ⟨≈˘≈⟩ ˘-cong Λ⨾∈˘) ⟨≈≈˘⟩ ˘/-˘ ⟩
      (R ˘ / (Q ⨾ ∈ ˘)) ˘
    ≈⟨ ˘-cong (⊑-antisym
      (/-universal (⊑-begin
          (R ˘ / (Q ⨾ ∈ ˘)) ⨾ Q ⨾ Λ₀ (∈ \ R)
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-⨾-inner ⟩
          (R ˘ / ∈ ˘) ⨾ Λ₀ (∈ \ R)
        ⊑⟨ ⨾-cong₁ \-˘ ⟨≈˘⊑⟩ ˘⨾Λ ⟩
          ∈
        □))
      (/-universal (⊑-begin
          (∈ / (Q ⨾ Λ₀ (∈ \ R))) ⨾ (Q ⨾ ∈ ˘)
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-⨾-inner ⟩
          (∈ / Λ₀ (∈ \ R)) ⨾ ∈ ˘
        ⊑⟨ ⨾-monotone₂ (proj₁ Λ-total ⟨⊑≈⟩ ⨾-assoc) ⟩
          (∈ / Λ₀ (∈ \ R)) ⨾ Λ₀ (∈ \ R) ⨾ Λ₀ (∈ \ R) ˘ ⨾ ∈ ˘
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-outer ⟩
          ∈ ⨾ Λ₀ (∈ \ R) ˘ ⨾ ∈ ˘
        ≈⟨ ⨾-assocL ⟨≈≈˘⟩ ⨾-cong₁ ˘-involutionRightConv ⟩
          (Λ₀ (∈ \ R) ⨾ ∈ ˘) ˘ ⨾ ∈ ˘
        ≈⟨ ⨾-cong₁ (˘-cong Λ⨾∈˘) ⟩
          (∈ \ R) ˘ ⨾ ∈ ˘
        ⊑⟨ ⨾-cong₁ \-˘ ⟨≈⊑⟩ /-cancel-outer ⟩
          R ˘
        □))) ⟩
      (∈ / (Q ⨾ Λ₀ (∈ \ R))) ˘
    ≈⟨ /-˘ ⟩
      (Q ⨾ Λ₀ (∈ \ R)) ˘ \ ∈ ˘
      □) ⟩
      Λ₀ ((Q ⨾ Λ₀ (∈ \ R)) ˘ \ ∈ ˘)
  □
```

In general, Glb-cocontinuous (R ↑) does not hold. To see this, consider its expansion:

∀ Q → Glb Q ⨾₁ (R ↑) ≈₁ Lub (Q ⨾ Mapping.mor (R ↑))

If Q is not total, the resulting empty intersections on the left-hand side may be mapped by R ↑ to arbitrary sets, but on the right-hand side, the resulting empty unions are always the empty set.

By assuming Glb-cocontinuous (R ↑), we can derive a necessary condition:

```
↑-Glb-cocontinuous˘ : {A B : Obj} (R : Mor A B) {X : Obj} (Q : Mor X (ℙ A))
  → Glb Q ⨾₁ (R ↑) ≈₁ Lub (Q ⨾ Mapping.mor (R ↑))
  → (∈ / Q) \ R ≈ (Q ⨾ (∈ \ R))
↑-Glb-cocontinuous˘ R {X} Q assumption = ≈-begin
      (∈ / Q) \ R
  ≈˘⟨ \-cong₁ (˘-involutionRightConv ⟨≈˘≈⟩ ˘-cong Λ⨾∈˘ ⟨≈≈⟩ ˘\˘-˘) ⟩
      (∈ ⨾ Λ₀ (Q ˘ \ ∈ ˘) ˘) \ R
  ≈˘⟨ \-inner-⨾ Λ-mapping ⟩
      Λ₀ (Q ˘ \ ∈ ˘) ⨾ (∈ \ R)
  ≈˘⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘ ⟩
      (Λ₀ (Q ˘ \ ∈ ˘) ⨾ Λ₀ (∈ \ R)) ⨾ ∈ ˘
  ≈⟨ Λ⇒∈ {f = Λ (Q ˘ \ ∈ ˘) ⨾₁ Λ (∈ \ R)} (≈-begin
      Λ₀ (Q ˘ \ ∈ ˘) ⨾ Λ₀ (∈ \ R)
    ≈⟨ assumption ⟩
      Λ₀ ((Q ⨾ Λ₀ (∈ \ R)) ⨾ ∈ ˘)
    ≈⟨ Λ-cong (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘) ⟩
      Λ₀ (Q ⨾ (∈ \ R))
    □) ⟩
    (Q ⨾ (∈ \ R))
  □
```

This condition does not always hold: In the power-allegory of sets, if we set Q = ⊥ and R = ⊤, then we have:

```
    (∈ / Q) \ R
  ≈⟨ ≈-refl ⟩    -- Def. Q and R
    (∈ / ⊥) \ ⊤
  ≈⟨ ⊤ ⨾ ⊥ ⊑ ∈ ⟩
    ⊤ \ ⊤
  ≈⟨ ⊤ ⨾ ⊤ ⊑ ⊤ ⟩
    ⊤
  ≉⟨ {- For relations between non-empty sets, ⊤ ≠ ⊥ ! -} ⟩
    ⊥
  ≈˘⟨ ⊥-leftZero ⟩
    ⊥ ⨾ (∈ \ ⊤)
  ≈⟨ ≈-refl ⟩    -- Def. Q and R
    Q ⨾ (∈ \ R)
```

But this condition is in fact also sufficient: The closest we can have to Glb-cocontinuous (R ↑) is the following:

```
↑-Glb-cocontinuous : {A B : Obj} (R : Mor A B) {X : Obj} (Q : Mor X (ℙ A))
  → (∈ / Q) \ R ≈ (Q ⨾ (∈ \ R))
  → Glb Q ⨾₁ (R ↑) ≈₁ Lub (Q ⨾ Mapping.mor (R ↑))
↑-Glb-cocontinuous R {X} Q assumption = ≈-begin
      Λ₀ (Q ˘ \ ∈ ˘) ⨾ Λ₀ (∈ \ R)
  ≈⟨ ∈⇒Λ {f = Λ (Q ˘ \ ∈ ˘) ⨾₁ Λ (∈ \ R)} (≈-begin
      (Λ₀ (Q ˘ \ ∈ ˘) ⨾ Λ₀ (∈ \ R)) ⨾ ∈ ˘
    ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘ ⟩
      Λ₀ (Q ˘ \ ∈ ˘) ⨾ (∈ \ R)
    ≈⟨ \-inner-⨾ Λ-mapping ⟩
      (∈ ⨾ Λ₀ (Q ˘ \ ∈ ˘) ˘) \ R
    ≈⟨ \-cong₁ (˘-involutionRightConv ⟨≈˘≈⟩ ˘-cong Λ⨾∈˘ ⟨≈≈⟩ ˘\˘-˘) ⟩
      (∈ / Q) \ R
    ≈⟨ assumption ⟩
      (Q ⨾ (∈ \ R))
    □) ⟩
    Λ₀ (Q ⨾ (∈ \ R))
  ≈˘⟨ Λ-cong (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘) ⟩
    Λ₀ ((Q ⨾ Λ₀ (∈ \ R)) ⨾ ∈ ˘)
  □
```

We now define the composed operators _↑↓ and _↓↑, and derive their closure properties.

```
_↑↓ : {A B : Obj} → Mor A B → Mapping (ℙ A) (ℙ A)
R ↑↓ = R ↑ ⨾₁ R ↓
_↓↑ : {A B : Obj} → Mor A B → Mapping (ℙ B) (ℙ B)
R ↓↑ = R ↓ ⨾₁ R ↑

_↑↓₀ : {A B : Obj} → Mor A B → Mor (ℙ A) (ℙ A)
R ↑↓₀ = Mapping.mor (R ↑↓)
_↓↑₀ : {A B : Obj} → Mor A B → Mor (ℙ B) (ℙ B)
R ↓↑₀ = Mapping.mor (R ↓↑)
```

To prepare for the expansion properties ↑↓⊑Ω and ↓↑⊑Ω below, we first derive some simpler properties about ↑ and ↓:

```
↑˘⨾∈˘ : {A B : Obj} {R : Mor A B} → R ↑₀ ˘ ⨾ ∈ ˘ ⊑ ∈ \ (R ˘)
↑˘⨾∈˘ {A} {B} {R} = \-universal (⊑-begin
    ∈ ⨾ (Λ₀ (∈ \ R)) ˘ ⨾ ∈ ˘
  ≈⟨ ⨾-assocL ⟨≈≈˘⟩ ⨾-cong₁ ˘-involutionRightConv ⟩
    (Λ₀ (∈ \ R) ⨾ ∈ ˘) ˘ ⨾ ∈ ˘
  ≈⟨ ⨾-cong₁ (˘-cong Λ⨾∈˘) ⟩
    (∈ \ R) ˘ ⨾ ∈ ˘
  ≈˘⟨ ˘-involution ⟩
    (∈ ⨾ (∈ \ R)) ˘
  ⊑⟨ ˘-monotone \-cancel-outer ⟩
    R ˘
  □)
∈⨾↑ : {A B : Obj} {R : Mor A B} → ∈ ⨾ R ↑₀ ⊑ (∈ \ (R ˘)) ˘   -- ≈ ∈ ⨾ R ↓₀ ˘
∈⨾↑ {A} {B} {R} = ⊑-˘-swap (˘-involution ⟨≈⊑⟩ ↑˘⨾∈˘)

↑˘⨾∈˘⊑↓⨾∈˘ : {A B : Obj} {R : Mor A B} → (R ↑₀) ˘ ⨾ ∈ ˘ ⊑ R ↓₀ ⨾ ∈ ˘
↑˘⨾∈˘⊑↓⨾∈˘ = ↑˘⨾∈˘ ⟨⊑≈˘⟩ Λ⨾∈˘
∈⨾↑⊑∈⨾↓˘ : {A B : Obj} {R : Mor A B} → ∈ ⨾ R ↑₀ ⊑ ∈ ⨾ R ↓₀ ˘
∈⨾↑⊑∈⨾↓˘ = ∈⨾↑ ⟨⊑≈˘⟩ ∈⨾Λ˘
```

Now the expansion proof is essentially shunting applied to ↑˘⨾∈˘:

```
∈˘⊑↑↓⨾∈˘ : {A B : Obj} {R : Mor A B} → ∈ ˘ ⊑ R ↑↓₀ ⨾ ∈ ˘
∈˘⊑↑↓⨾∈˘ {R = R} = ⊑-begin
    ∈ ˘
  ⊑⟨ proj₁ Λ-total ⟨⊑≈⟩ ⨾-assoc ⟩
    Λ₀ (∈ \ R) ⨾ (Λ₀ (∈ \ R)) ˘ ⨾ ∈ ˘
  ⊑⟨ ⨾-monotone₂ ↑˘⨾∈˘ ⟩
    Λ₀ (∈ \ R) ⨾ (∈ \ (R ˘))
  ≈˘⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘ ⟩
    R ↑↓₀ ⨾ ∈ ˘
  □
```

To make the following calculation more readable, we use a dualised variant of the ⊑-begin ... ⊑⟨ ... ⟩ ... ≈⟨ ... ⟩ ... □ setup, which for the time being still requires additional primes in ≈⟨ ... ⟩′ ... □′ to avoid ambiguity.

```
∈⨾↑↓˘⊒∈ : {A B : Obj} {R : Mor A B} → (∈ ⨾ (R ↑↓₀) ˘) ⊒ ∈
∈⨾↑↓˘⊒∈ {R = R} = ⊒-begin
    ∈ ⨾ (R ↑↓₀) ˘
  ≈⟨ (⨾-cong₂ ˘-involution) ⟨≈≈⟩ ⨾-assocL ⟩′
    (∈ ⨾ Λ₀ (∈ \ R ˘) ˘) ⨾ Λ₀ (∈ \ R) ˘
  ≈⟨ ⨾-cong₁ ∈⨾Λ˘ ⟩′
    (∈ \ R ˘) ˘ ⨾ Λ₀ (∈ \ R) ˘
  ⊒⟨ ⨾-monotone₁ ∈⨾↑ ⟩
    (∈ ⨾ Λ₀ (∈ \ R)) ⨾ (Λ₀ (∈ \ R)) ˘
  ≈⟨ ⨾-assoc ⟩′
    ∈ ⨾ Λ₀ (∈ \ R) ⨾ (Λ₀ (∈ \ R)) ˘
  ⊒⟨ proj₂ Λ-total ⟩
    ∈
  □′
∈˘⊑↓↑⨾∈˘ : {A B : Obj} {R : Mor A B} → ∈ ˘ ⊑ R ↓↑₀ ⨾ ∈ ˘
∈˘⊑↓↑⨾∈˘ {R = R} = ⊑-begin
    ∈ ˘
  ⊑⟨ proj₁ Λ-total ⟨⊑≈⟩ ⨾-assoc ⟩
    Λ₀ (∈ \ R ˘) ⨾ (Λ₀ (∈ \ R ˘)) ˘ ⨾ ∈ ˘
  ⊑⟨ ⨾-monotone₂ ↑˘⨾∈˘ ⟩
    Λ₀ (∈ \ R ˘) ⨾ (∈ \ (R ˘ ˘))
  ≈˘⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ (Λ⨾∈˘ ⟨≈≈˘⟩ \-cong₂ ˘˘) ⟩
    R ↓↑₀ ⨾ ∈ ˘
  □
```

With this, we now can show that R ↑↓ and R ↓↑ are always expanding (in the subset ordering Ω):

```
∈⨾↑↓ : {A B : Obj} {R : Mor A B} → ∈ ⨾ R ↑↓₀ ⊑ ∈
∈⨾↑↓ {A} {B} {R} = ⊑-begin
    ∈ ⨾ R ↑↓₀
  ⊑⟨ ⨾-monotone₁ (⊑-˘-swap ∈˘⊑↑↓⨾∈˘ ⟨⊑≈⟩ ˘-involutionRightConv) ⟩
    (∈ ⨾ (R ↑↓₀) ˘) ⨾ R ↑↓₀
  ⊑⟨ ⨾-assoc ⟨≈⊑⟩ proj₂ (Mapping.unival (R ↑↓)) ⟩
    ∈
  □

↑↓⊑Ω : {A B : Obj} {R : Mor A B} → R ↑↓₀ ⊑ Ω
↑↓⊑Ω {A} {B} {R} = \-universal ∈⨾↑↓

∈⨾↓↑ : {A B : Obj} {R : Mor A B} → ∈ ⨾ R ↓↑₀ ⊑ ∈
∈⨾↓↑ {A} {B} {R} = ⊑-begin
    ∈ ⨾ R ↓↑₀
  ⊑⟨ ⨾-monotone₁ (⊑-˘-swap ∈˘⊑↓↑⨾∈˘ ⟨⊑≈⟩ ˘-involutionRightConv) ⟩
    (∈ ⨾ (R ↓↑₀) ˘) ⨾ R ↓↑₀
  ⊑⟨ ⨾-assoc ⟨≈⊑⟩ proj₂ (Mapping.unival (R ↓↑)) ⟩
    ∈
  □

↓↑⊑Ω : {A B : Obj} {R : Mor A B} → R ↓↑₀ ⊑ Ω
↓↑⊑Ω {A} {B} {R} = \-universal ∈⨾↓↑
```

Composition of our closure operators with ∈˘ translates into simple residual expressions:

```
↑↓⨾∈˘₀ : {A B : Obj} {R : Mor A B} → R ↑↓₀ ⨾ ∈ ˘ ≈ (∈ \ R) ˘ \ (R ˘)
↑↓⨾∈˘₀ {A} {B} {R} = ≈-begin
    R ↑↓₀ ⨾ ∈ ˘
  ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘ ⟩
    Λ₀ (∈ \ R) ⨾ (∈ \ (R ˘))
  ≈⟨ \-inner-⨾ Λ-mapping ⟩
    (∈ ⨾ Λ₀ (∈ \ R) ˘) \ (R ˘)
  ≈˘⟨ \-cong₁ ˘-involutionRightConv ⟩
    (Λ₀ (∈ \ R) ⨾ ∈ ˘) ˘ \ (R ˘)
  ≈⟨ \-cong₁ (˘-cong Λ⨾∈˘) ⟩
    (∈ \ R) ˘ \ (R ˘)
  □
```

```
↑↓⨾∈˘ : {A B : Obj} {R : Mor A B} → R ↑↓₀ ⨾ ∈ ˘ ≈ (R / (∈ \ R)) ˘
↑↓⨾∈˘ {A} {B} {R} = ≈-begin
    R ↑↓₀ ⨾ ∈ ˘
  ≈⟨ ↑↓⨾∈˘₀ ⟩
    (∈ \ R) ˘ \ (R ˘)
  ≈˘⟨ /-˘ ⟩
    (R / (∈ \ R)) ˘
  □
```

```
↑↓⨾∈˘′ : {A B : Obj} {R : Mor A B} → R ↑↓₀ ⨾ ∈ ˘ ≈ (R ˘ / ∈ ˘) \ (R ˘)
↑↓⨾∈˘′ {A} {B} {R} = ≈-begin
    R ↑↓₀ ⨾ ∈ ˘
  ≈⟨ ↑↓⨾∈˘₀ ⟩
    (∈ \ R) ˘ \ (R ˘)
  ≈⟨ \-cong₁ \-˘ ⟩
    (R ˘ / ∈ ˘) \ (R ˘)
  □
```

```
↓↑⨾∈˘ : {A B : Obj} {R : Mor A B} → R ↓↑₀ ⨾ ∈ ˘ ≈ (R / ∈ ˘) \ R
↓↑⨾∈˘ {A} {B} {R} = ≈-begin
    R ↓↑₀ ⨾ ∈ ˘
  ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘ ⟩
    Λ₀ (∈ \ R ˘) ⨾ (∈ \ R)
  ≈⟨ \-inner-⨾ Λ-mapping ⟩
    (∈ ⨾ Λ₀ (∈ \ R ˘) ˘) \ R
  ≈˘⟨ \-cong₁ ˘-involutionRightConv ⟩
    (Λ₀ (∈ \ R ˘) ⨾ ∈ ˘) ˘ \ R
  ≈⟨ \-cong₁ (˘-cong Λ⨾∈˘) ⟩
    (∈ \ R ˘) ˘ \ R
  ≈⟨ \-cong₁ \˘-˘ ⟩
    (R / ∈ ˘) \ R
  □
```

We also have some absorptive properties:

```
↓⨾↑↓ : {A B : Obj} {R : Mor A B} → R ↓ ⨾₁ R ↑↓ ≈₁ R ↓
↓⨾↑↓ {A} {B} {R} = ≈-begin
    R ↓₀ ⨾ R ↑↓₀
  ≈⟨ ∈⇒Λ {f = R ↓ ⨾₁ R ↑↓} (≈-begin
      (R ↓₀ ⨾ R ↑↓₀) ⨾ ∈ ˘
    ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ↑↓⨾∈˘ ⟩
      Λ₀ (∈ \ (R ˘)) ⨾ (R / (∈ \ R)) ˘
    ≈⟨ ⊑-antisym
      (\-universal (⊑-begin
          ∈ ⨾ Λ₀ (∈ \ (R ˘)) ⨾ (R / (∈ \ R)) ˘
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ /-universal′ (⊑-begin
          ∈ ⨾ Λ₀ (∈ \ (R ˘))
          ⊑⟨ /-universal (⊑-begin
              (∈ ⨾ Λ₀ (∈ \ (R ˘))) ⨾ ∈ ˘
            ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘ ⟩
              ∈ ⨾ (∈ \ (R ˘))
            ⊑⟨ \-cancel-outer ⟩
              R ˘
            □) ⟩
            R ˘ / ∈ ˘
          ≈˘⟨ \-˘ ⟩
            (∈ \ R) ˘
          ≈˘⟨ ˘-cong \S∘S/∘\S ⟩
            ((R / (∈ \ R)) \ R) ˘
          ≈⟨ \-˘ ⟩
            R ˘ / (R / (∈ \ R)) ˘
          □) ⟩
          R ˘
        □))
      (⊑-begin
          ∈ \ (R ˘)
        ≈˘⟨ Λ⨾∈˘ ⟩
          Λ₀ (∈ \ (R ˘)) ⨾ ∈ ˘
        ⊑⟨ ⨾-monotone₂ (˘-monotone ⊑-S/∘\S) ⟩
          Λ₀ (∈ \ (R ˘)) ⨾ (R / (∈ \ R)) ˘
        □) ⟩
      ∈ \ (R ˘)
    □) ⟩
    Λ₀ (∈ \ (R ˘))
  ≈⟨ ≈-refl ⟩
    R ↓₀
  □
```

```
↓↑⨾↓ : {A B : Obj} {R : Mor A B} → R ↓↑ ⨾₁ R ↓ ≈₁ R ↓
↓↑⨾↓ = ⨾-assoc ⟨≈≈⟩ ↓⨾↑↓
```

```
↑↓-idempotent : {A B : Obj} {R : Mor A B} → R ↑↓ ⨾₁ R ↑↓ ≈₁ R ↑↓
↑↓-idempotent = ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ↓⨾↑↓
```

```
↓↑-idempotent : {A B : Obj} {R : Mor A B} → R ↓↑ ⨾₁ R ↓↑ ≈₁ R ↓↑
↓↑-idempotent = ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ ↓↑⨾↓
```

This gives us convenient ways to express restrictions to closed elements:

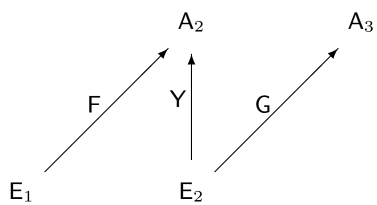
```
↓↑-ranClosed-→ : {A B C : Obj} {R : Mor A (ℙ B)} {S : Mor C B}
  →              R ⊑ R ⨾ (S ↓↑₀) ˘ ⨾ S ↓↑₀ → R ⨾ S ↓↑₀ ≈ R
↓↑-ranClosed-→ {S = S} = mapRanClosed-→ (Mapping.prf (S ↓↑)) ↓↑-idempotent
```

```
↓↑-ranClosed-← : {A B C : Obj} {R : Mor A (ℙ B)} {S : Mor C B}
  →              R ⨾ S ↓↑₀ ≈ R → R ⊑ R ⨾ (S ↓↑₀) ˘ ⨾ S ↓↑₀
↓↑-ranClosed-← {S = S} = mapRanClosed-← (Mapping.prf (S ↓↑)) ↓↑-idempotent
```

```
↓⨾↑↓′ : {A B₁ B₂ : Obj} {Q : Mor A B₁} {R : Mor A B₂}
    → R / (∈ \ R) ⊑ Q / (∈ \ Q) → Q ↓ ⨾₁ R ↑↓ ≈₁ Q ↓
↓⨾↑↓′ {A} {B₁} {B₂} {Q} {R} p = ≈-begin
    Q ↓₀ ⨾ R ↑↓₀
  ≈⟨ ∈⇒Λ {f = Q ↓ ⨾₁ R ↑↓} (≈-begin
      (Q ↓₀ ⨾ R ↑↓₀) ⨾ ∈ ˘
    ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ↑↓⨾∈˘ ⟩
      Λ₀ (∈ \ (Q ˘)) ⨾ (R / (∈ \ R)) ˘
    ≈⟨ ⊑-antisym
      (\-universal (⊑-begin
          ∈ ⨾ Λ₀ (∈ \ (Q ˘)) ⨾ (R / (∈ \ R)) ˘
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ /-universal′ (⊑-begin
          ∈ ⨾ Λ₀ (∈ \ (Q ˘))
          ⊑⟨ /-universal (⊑-begin
              (∈ ⨾ Λ₀ (∈ \ (Q ˘))) ⨾ ∈ ˘
            ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘ ⟩
              ∈ ⨾ (∈ \ (Q ˘))
            ⊑⟨ \-cancel-outer ⟩
              Q ˘
            □) ⟩
            Q ˘ / ∈ ˘
          ≈˘⟨ \-˘ ⟩
            (∈ \ Q) ˘
          ≈˘⟨ ˘-cong (\T∘S/∘\S p) ⟩
```

```
          ((R / (∈ \ R)) \ Q) ˘
        ≈⟨ \-˘ ⟩
          Q ˘ / (R / (∈ \ R)) ˘
        □) ⟩
        Q ˘
      □))
    (⊑-begin
        ∈ \ (Q ˘)
      ≈˘⟨ Λ⨾∈˘ ⟩
        Λ₀ (∈ \ (Q ˘)) ⨾ ∈ ˘
      ⊑⟨ ⨾-monotone₂ (˘-monotone ⊑-S/◦\S) ⟩
        Λ₀ (∈ \ (Q ˘)) ⨾ (R / (∈ \ R)) ˘
      □) ⟩
    ∈ \ (Q ˘)
  □) ⟩
  Λ₀ (∈ \ (Q ˘))
≈⟨ ≈-refl ⟩
  Q ↓₀
□
```

For composition of context homomorphisms, we will require Lub-cocontinuity of G ↓ ⨾₁ Y ↑ ⨾₁ F ↓ in the following situation:



The following calculation follows Moshier (2013):

```
↓↑↓-Lub-cocontinuous : {E₁ E₂ A₂ A₃ : Obj}
                     → (F : Mor E₁ A₂) (Y : Mor E₂ A₂) (G : Mor E₂ A₃)
                     → (F-trgCompat : Y ↓↑ ⨾₁ F ↓ ≈₁ F ↓)
                     → (G-srcCompat : G ↓ ⨾₁ Y ↑↓ ≈₁ G ↓)
                     → Lub-cocontinuous (G ↓ ⨾₁ Y ↑ ⨾₁ F ↓)
↓↑↓-Lub-cocontinuous F Y G F-trgCompat G-srcCompat Q = ≈₁-begin
      Lub Q ⨾₁ G ↓ ⨾₁ Y ↑ ⨾₁ F ↓
    ≈₁⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ (↓-Lub-cocontinuous G Q) ⟩
      Glb (Q ⨾ G ↓₀) ⨾₁ Y ↑ ⨾₁ F ↓
    ≈₁⟨ ⨾-cong₁ (Glb-cong (⨾-cong₂ G-srcCompat ⟨≈˘≈⟩ ⨾-assocL₃₊₁)) ⟩
      Glb ((Q ⨾ G ↓₀ ⨾ Y ↑₀) ⨾ Y ↓₀) ⨾₁ Y ↑ ⨾₁ F ↓
    ≈₁⟨ ⨾-cong₁ (↓-Lub-cocontinuous Y (Q ⨾ G ↓₀ ⨾ Y ↑₀)) ⟨≈˘≈⟩ ⨾-assoc ⟩
      Lub (Q ⨾ G ↓₀ ⨾ Y ↑₀) ⨾₁ Y ↓ ⨾₁ Y ↑ ⨾₁ F ↓
    ≈₁⟨ ⨾-cong₂ (⨾-assocL ⟨≈≈⟩ F-trgCompat) ⟩
      Lub (Q ⨾ G ↓₀ ⨾ Y ↑₀) ⨾₁ F ↓
    ≈₁⟨ ↓-Lub-cocontinuous F (Q ⨾ G ↓₀ ⨾ Y ↑₀) ⟨≈≈⟩ Glb-cong ⨾-assoc₃₊₁ ⟩
      Glb (Q ⨾ G ↓₀ ⨾ Y ↑₀ ⨾ F ↓₀)
    □₁
```

```
˘-isPreorder₀ : IsPreorder (E ˘)
˘-isPreorder₀ = record {supId = ˘-leftSupId, ˘-rightSupId; trans = ˘-trans}
```

5 Internal Order Theory and Direct Powers without Meet

5.1 Categoric.OSGC.Preorder

```
module _ {i j k₁ k₂} {Obj : Set i} (osgc : OSGC j k₁ k₂ Obj) where
  open OSGC osgc
```

Within an ordered semigroupoid with converse, a preorder is a morphism that is a superidentity and is transitive,

```
  record IsPreorder {A : Obj} (E : Mor A A) : Set (k₂ ⊍ j ⊍ i) where
    field
      supId : isSuperidentity E
      trans : IsTransitive E
```

For convenience, we define some useful combinators.

```
    leftSupId : isLeftSuperidentity E
    leftSupId = proj₁ supId

    rightSupId : isRightSuperidentity E
    rightSupId = proj₂ supId

    ˘-leftSupId : isLeftSuperidentity (E ˘)
    ˘-leftSupId = ˘-isLeftSuperidentity rightSupId

    ˘-rightSupId : isRightSuperidentity (E ˘)
    ˘-rightSupId = ˘-isRightSuperidentity leftSupId

    ˘-supId : isSuperidentity (E ˘)
    ˘-supId = ˘-leftSupId, ˘-rightSupId
```

5.1.1 The Dual Preorder

The converse of a preorder is again a preorder,

```
    ˘-trans : IsTransitive (E ˘)
    ˘-trans = ⊑-begin
            E ˘ ⨾ E ˘
          ≈˘⟨ ˘-involution ⟩
            (E ⨾ E) ˘
          ⊑⟨ ˘-monotone trans ⟩
            E ˘
          □

    idempot : IsIdempotent E
    idempot = ⊑-antisym trans rightSupId

    ˘-idempot : IsIdempotent (E ˘)
    ˘-idempot = ⊑-antisym ˘-trans ˘-rightSupId
```

More explicitly,

5.1.2 Indirect inclusion

The proof method of indirect inclusion, as presented in Relation.Binary.Poset.Renamed (Sect. 2.1), can also be moved over, though generalizing from 'points', functions to a terminal object, to arbitrary functions f, g as

$$(\forall\ x \bullet f\ x \preccurlyeq g\ x) \equiv (\forall\ x, y \bullet g\ x \preccurlyeq z \Rightarrow f\ x \preccurlyeq z)$$

In fact, a total and univalent pair suffice:

```
indirect-E : {B : Obj} {F : Mor B A} {G : Mor B A}
  → isTotal G → isUnivalent F → G ⨾ E ⊑ F ⨾ E → F ˘ ⊑ E ⨾ G ˘
indirect-E {B} {F} {G} g-tot f-univ GE-FE = ⊑-begin
    F ˘
  ⊑⟨ proj₂ g-tot ⟩
    F ˘ ⨾ G ⨾ G ˘
  ⊑⟨ ⨾-monotone₂₂ leftSupId ⟩
    F ˘ ⨾ G ⨾ E ⨾ G ˘
  ⊑⟨ ⨾-monotone₂ (⨾-assoc ⟨≈˘⊑⟩ (⨾-monotone₁ GE-FE ⟨⊑≈⟩ ⨾-assoc)) ⟩
    F ˘ ⨾ F ⨾ E ⨾ G ˘
  ≈⟨ ⨾-assoc₄ ⟨≈˘≈⟩ ⨾-assoc ⟩
    (F ˘ ⨾ F) ⨾ E ⨾ G ˘
  ⊑⟨ proj₁ f-univ ⟩
    E ⨾ G ˘
  □

indirect-∃ : {B : Obj} {F : Mor B A} {G : Mor B A}
  → isTotal G → isUnivalent F → G ⨾ E ⊑ F ⨾ E → F ⊑ G ⨾ E ˘
indirect-∃ tot univ indir = ⊑-˘-swap (indirect-E tot univ indir) ⟨⊑≈⟩ ˘-involutionRightConv
```

5.1.3 Bounds

If in addition we have access to residuation, then we may discuss bounds.

```
  module PreorderWithResiduals
      (leftResOp  : LeftResOp orderedSemigroupoid)
      (rightResOp : RightResOp orderedSemigroupoid) where
    open ResidualOps leftResOp rightResOp
    open OSGC-Residuals osgc leftResOp rightResOp
```

Majorants

Let us first discuss upper bounds, then dualize for lower bounds. We do so following Furusawa and Kahl (1998).

```
private
  module ubd-props {A : Obj} {E : Mor A A} (E-isPreorder : IsPreorder E) where
    open IsPreorder E-isPreorder
    ubd : {I : Obj} → Mor I A → Mor I A
    ubd Q = Q ˘ \ E
```

Useful combinators:

```
ubd-˘ : {I : Obj} {R : Mor A I} → ubd (R ˘) ≈ R \ E
ubd-˘ = \-cong₁ ˘˘

⨾-ubd-⊑ : {R : Mor A A} → R ⨾ ubd (R ˘) ⊑ E
⨾-ubd-⊑ {R} = ⊑-begin
      R ⨾ ubd (R ˘)
    ≈⟨  ⨾-cong₂ (\-cong₁ ˘˘) ⟩
      R ⨾ (R \ E)
    ⊑⟨ \-cancel-outer ⟩
      E
    □
```

The ‘cones’ and ‘closures’ of bounds:

```
ubd-downcone₀ : {I : Obj} {Q : Mor I A} → (E ⨾ Q ˘) \ E ≈ ubd Q
ubd-downcone₀ {I} {Q} = ⊑-antisym (⊑-begin
      (E ⨾ Q ˘) \ E
    ⊑⟨ \-antitone (proj₁ supId) ⟩
      ubd Q
    □) (⊑-begin
      ubd Q
    ⊑⟨ \-universal (⨾-assoc ⟨≈⊑⟩ (⨾-monotone₂ \-cancel-outer ⟨⊑⊑⟩ trans)) ⟩
      (E ⨾ Q ˘) \ E
    □)

ubd-downcone : {I : Obj} {Q : Mor I A} → ubd (Q ⨾ E ˘) ≈ ubd Q
ubd-downcone {I} {Q} = ≈-begin
      ubd (Q ⨾ E ˘)
    ≈⟨⟩
      (Q ⨾ E ˘) ˘ \ E
    ≈⟨ \-cong₁ ˘-involutionRightConv ⟩
      (E ⨾ Q ˘) \ E
    ≈⟨ ubd-downcone₀ ⟩
      ubd Q
    □

ubd-upclosed : {I : Obj} {Q : Mor I A} → ubd (Q) ⨾ E ≈ ubd Q
ubd-upclosed {I} {Q} = ⊑-antisym (⊑-begin
      ubd Q ⨾ E
    ⊑⟨ \-outer-⨾ ⟩
      Q ˘ \ (E ⨾ E)
    ⊑⟨ \-monotone trans ⟩
      ubd Q
    □) (proj₂ supId)
```

The relation between mappings and bounds:

```
Mapping-⨾-ubd : {I J : Obj} {F : Mor I J} {Q : Mor J A}
  → isMapping F → F ⨾ ubd Q ≈ ubd (F ⨾ Q)
Mapping-⨾-ubd {I} {J} {F} {Q} F-isMapping = ≈-begin
    F ⨾ (Q ˘ \ E)
  ≈⟨ \-inner-⨾ F-isMapping ⟩
    (Q ˘ ⨾ F ˘) \ E
  ≈˘⟨ \-cong₁ ˘-involution ⟩
    ubd (F ⨾ Q)
  □

ubd-mapping : {I : Obj} {R : Mor I A} → isMapping R → ubd R ≈ R ⨾ E
ubd-mapping {I} {R} (R-unival, R-total) = ⊑-antisym (⊑-begin
    R ˘ \ E
  ⊑⟨ proj₁ R-total ⟨⊑≈⟩ ⨾-assoc ⟩
    R ⨾ R ˘ ⨾ (R ˘ \ E)
  ⊑⟨ ⨾-monotone₂ \-cancel-outer ⟩
    R ⨾ E
  □) (⊑-begin
    R ⨾ E
  ⊑⟨ \-universal (⨾-assoc ⟨≈˘⊑⟩ proj₁ R-unival) ⟩
    R ˘ \ E
  □)

⨾order-⊑-ubd-→ : {I : Obj} {Q R : Mor I A} → R ⨾ E ⊑ ubd Q → R ⊑ ubd Q
⨾order-⊑-ubd-→ {I} {Q} {R} R⨾E⊑ubdQ
  = \-universal (⨾-monotone₂ rightSupId ⟨⊑⊑⟩ \-universal′ R⨾E⊑ubdQ)

⨾order-⊑-ubd-← : {I : Obj} {Q R : Mor I A} → R ⊑ ubd Q → R ⨾ E ⊑ ubd Q
⨾order-⊑-ubd-← {I} {Q} {R} R⊑ubdQ = ⨾-monotone₁ R⊑ubdQ ⟨⊑≈⟩ ubd-upclosed

order-\ : E \ E ≈ E
order-\ = ⊑-antisym (⊑-begin
    E \ E
  ⊑⟨ proj₁ supId ⟩
    E ⨾ (E \ E)
  ⊑⟨ \-cancel-outer ⟩
    E
  □) (\-universal trans)

order-/ : E / E ≈ E
order-/ = ⊑-antisym (proj₂ supId ⟨⊑⊑⟩ /-cancel-outer) (/-universal trans)
```

Compare these results with indirect-E above.

An immediate nifty consequence,

```
ubd-order˘ : ubd (E ˘) ≈ E
ubd-order˘ = ≈-begin
    ubd (E ˘)
  ≈⟨⟩
    (E ˘) ˘ \ E
  ≈⟨ \-cong₁ ˘˘ ⟩
    E \ E
  ≈⟨ order-\ ⟩
    E
  □
```

Minorants

Flipping the order around yields dual results:

```
private
  module lbd-props {A : Obj} {E : Mor A A} (E-isPreorder : IsPreorder E) where
    open IsPreorder E-isPreorder
    open ubd-props ˘-isPreorder₀ public hiding (ubd-downcone; ubd-order˘) renaming
        (ubd              to lbd               -- : {I : Obj} → Mor I A → Mor I A
        ; ubd-˘           to lbd-˘             -- : ∀ {R} → lbd (R ˘) ≈ R \ E ˘
        ; ⨾-ubd-⊑         to ⨾-lbd-⊑           -- : ∀ {R} → R ⨾ lbd (R ˘) ⊑ E ˘
        ; ubd-downcone₀   to lbd-downcone₀     -- : ∀ {Q} → (E ˘ ⨾ Q ˘) \ E ˘ ≈ lbd Q
        ; ubd-upclosed    to lbd-downclosed    -- : ∀ {Q} → lbd (Q) ⨾ E ˘ ≈ lbd Q
        ; Mapping-⨾-ubd   to Mapping-⨾-lbd     -- : ∀ {F Q} → isMapping F → F ⨾ lbd Q ≈ lbd (F ⨾ Q)
        ; ubd-mapping     to lbd-mapping       -- : ∀ {R} → isMapping R → lbd R ≈ R ⨾ E ˘
        ; ⨾order-⊑-ubd-→ to ⨾order˘-⊑-lbd-→  -- : ∀ {Q R} → R ⨾ E ˘ ⊑ lbd Q → R ⊑ lbd Q
        ; ⨾order-⊑-ubd-← to ⨾order˘-⊑-lbd-←  -- : ∀ {Q R} → R ⊑ lbd Q → R ⨾ E ˘ ⊑ lbd Q
        ; order-/         to order˘-/          -- : E ˘ / E ˘ ≈ E ˘
        ; order-\         to order˘-\          -- : E ˘ \ E ˘ ≈ E ˘
        )
```

```
open ubd-props ˘-isPreorder₀ using (ubd-downcone; ubd-order˘)
lbd-upcone : {I : Obj} {Q : Mor I A} → lbd (Q ⨾ E) ≈ lbd Q
lbd-upcone = \-cong₁ (˘-cong (⨾-cong₂ ˘˘)) ⟨≈˘≈⟩ ubd-downcone
lbd-order : lbd E ≈ E ˘
lbd-order = \-cong₁ (˘-cong ˘˘) ⟨≈˘≈⟩ ubd-order˘
```

Bound-functionals are Galois Connected

Consider an arbitrary preorder E,

```
module IsPreorder′ {A : Obj} {E : Mor A A} (E-isPreorder : IsPreorder E) where
  open IsPreorder E-isPreorder
```

That these bound operators are Galois connected yields many free results.

```
open ubd-props E-isPreorder public
open lbd-props E-isPreorder public
ubd-lbd-isGC : {I : Obj} → IsGC (Hom I A) (dualPoset (Hom I A)) ubd lbd
ubd-lbd-isGC = record {
    gc = λ {Q} {R} R⊑ubdQ → \-universal (⊑-begin
          R ˘ ⨾ Q
        ⊑⟨ ⨾-monotone₁ (˘-monotone R⊑ubdQ) ⟩
          (Q ˘ \ E) ˘ ⨾ Q
        ≈⟨ ⨾-cong₁ ˘\-˘ ⟩
          (E ˘ / Q) ⨾ Q
        ⊑⟨ /-cancel-outer ⟩
          E ˘
        □);
    gc˘ = λ {Q} {R} Q⊑lbdR → \-universal (⊑-begin
          Q ˘ ⨾ R
        ⊑⟨ ⨾-monotone₁ (˘-monotone Q⊑lbdR) ⟩
          (R ˘ \ E ˘) ˘ ⨾ R
        ≈⟨ ⨾-cong₁ ˘\˘-˘ ⟩
          (E / R) ⨾ R
        ⊑⟨ /-cancel-outer ⟩
          E
        □)
    }
module _ {I : Obj} where
    open IsGC (ubd-lbd-isGC {I}) public using () renaming
      (gc˘    to ubd-lbd-gc˘    -- : ∀ {Q R} → R ⊑ ubd Q → Q ⊑ lbd R
      ; gc      to ubd-lbd-gc     -- : ∀ {Q R} → Q ⊑ lbd R → R ⊑ ubd Q
      ; ≤-can to ⊑-lbd-ubd      -- : ∀ {R} → R ⊑ lbd (ubd R)
      ; ⊑-can to ⊑-ubd-lbd      -- : ∀ {R} → R ⊑ ubd (lbd R)
      ; L-cong to ubd-cong      -- : ∀ {R S} → R ≈ S → ubd R ≈ ubd S
      ; U-cong to lbd-cong      -- : ∀ {R S} → R ≈ S → lbd R ≈ lbd S
      ; L-monotone to ubd-antitone    -- : ∀ {R S} → R ⊑ S → ubd S ⊑ ubd R
      ; U-monotone to lbd-antitone    -- : ∀ {R S} → S ⊑ R → lbd R ⊑ lbd S
      ; L-semi-inverse to ubd-semi-inverse    -- : ∀ {R} → ubd (lbd (ubd R)) ≈ ubd R
      ; U-semi-inverse to lbd-semi-inverse    -- : ∀ {R} → lbd (ubd (lbd R)) ≈ lbd R
      )
```

Let us turn to the interaction between both bound operators and mappings.

```
Mapping-⨾-ubd-lbd : {I J : Obj} {F : Mor I J} {Q : Mor J A}
          → isMapping F → F ⨾ ubd (lbd Q) ≈ ubd (lbd (F ⨾ Q))
Mapping-⨾-ubd-lbd {I} {J} {F} {Q} F-isMapping = ≈-begin
      F ⨾ ubd (lbd Q)
    ≈⟨ Mapping-⨾-ubd F-isMapping ⟩
      ubd (F ⨾ lbd Q)
    ≈⟨ ubd-cong (Mapping-⨾-lbd F-isMapping) ⟩
      ubd (lbd (F ⨾ Q))
    □
Mapping-⨾-lbd-ubd : {I J : Obj} {F : Mor I J} {Q : Mor J A}
          → isMapping F → F ⨾ lbd (ubd Q) ≈ lbd (ubd (F ⨾ Q))
Mapping-⨾-lbd-ubd {I} {J} {F} {Q} F-isMapping = ≈-begin
      F ⨾ lbd (ubd Q)
    ≈⟨ Mapping-⨾-lbd F-isMapping ⟩
      lbd (F ⨾ ubd Q)
    ≈⟨ lbd-cong (Mapping-⨾-ubd F-isMapping) ⟩
      lbd (ubd (F ⨾ Q))
    □
```

Interaction of the Bound-Functionals

And a few more lemmas regarding the interaction of these two bound operators.

```
ubd-lbd : {I : Obj} {Q : Mor I A} → ubd (lbd Q) ≈ (E / Q) \ E
ubd-lbd {I} {Q} = ≈-begin
      (Q ˘ \ E ˘) ˘ \ E
    ≈⟨ \-cong₁ ˘\˘-˘ ⟩
      (E / Q) \ E
    □
ubd-lbd-˘ : {I : Obj} {Q : Mor I A} → ubd (lbd Q) ˘ ≈ E ˘ / lbd Q
ubd-lbd-˘ {I} {Q} = ≈-begin
      ubd (lbd Q) ˘
    ≈⟨ ˘-cong ubd-lbd ⟩
      ((E / Q) \ E) ˘
    ≈⟨ \-˘ ⟩
      E ˘ / (E / Q) ˘
    ≈⟨ /-cong₂ /-˘ ⟩
      E ˘ / (Q ˘ \ E ˘)
    ≈⟨⟩
      E ˘ / lbd Q
    □
ubd-lbd-⊒ : {I : Obj} {Q : Mor I A} → Q ⊑ ubd (lbd Q)
ubd-lbd-⊒ {I} {Q} = ⊑-ubd-lbd
lbd-ubd : {I : Obj} {Q : Mor I A} → lbd (ubd Q) ≈ (E ˘ / Q) \ E ˘
lbd-ubd {I} {Q} = ≈-begin
      (Q ˘ \ E) ˘ \ E ˘
    ≈⟨ \-cong₁ ˘\-˘ ⟩
      (E ˘ / Q) \ E ˘
    □
lbd-ubd-˘ : {I : Obj} {Q : Mor I A} → lbd (ubd Q) ˘ ≈ E / ubd Q
lbd-ubd-˘ {I} {Q} = ≈-begin
      lbd (ubd Q) ˘
    ≈⟨ ˘-cong lbd-ubd ⟩
      ((E ˘ / Q) \ E ˘) ˘
    ≈⟨ \˘-˘ ⟩
      E / (E ˘ / Q) ˘
    ≈⟨ /-cong₂ ˘/-˘ ⟩
      E / (Q ˘ \ E)
    ≈⟨⟩
      E / ubd Q
```

```
  □
lbd-ubd-⊒ : {I : Obj} {Q : Mor I A} → Q ⊑ lbd (ubd Q)
lbd-ubd-⊒ {I} {Q} = ⊑-lbd-ubd
```

Now we turn to the semi-inverse laws. As their direct proofs are not too difficult, we provide direct proofs and compare the sizes of the resulting normalised proof terms with those obtained from the Galois connection module. It seems that the direct proof of ubd-lbd-ubd, for example, is only 514 lines; whereas the derivation ubd-semi-inverse, though equivalent in content, is nearly three times larger at line count of 1573.

While lbd-ubd-lbd has proof term line count of 624, and lbd-semi-inverse has count of 1555.

```
ubd-lbd-ubd : {I : Obj} {Q : Mor I A} → ubd (lbd (ubd Q)) ≈ ubd Q   -- ≐ ubd-semi-inverse
ubd-lbd-ubd {I} {Q} = ≈-begin
    ubd (lbd (ubd Q))
  ≈⟨ ubd-lbd ⟩
    (E / ubd Q) \ E
  ≈⟨ ⊑-antisym (\-universal (⊑-begin
        Q ˘ ⨾ ((E / ubd Q) \ E)
      ⊑⟨ ⨾-monotone₂ (\-antitone ⊑-S/∘\S) ⟩
        Q ˘ ⨾ (Q ˘ \ E)
      ⊑⟨ \-cancel-outer ⟩
        E
      □)) ⊑-\S∘S/ ⟩
    ubd Q
  □

lbd-ubd-lbd : {I : Obj} {Q : Mor I A} → lbd (ubd (lbd Q)) ≈ lbd Q   -- ≐ lbd-semi-inverse
lbd-ubd-lbd {I} {Q} = ≈-begin
    lbd (ubd (lbd Q))
  ≈⟨ lbd-ubd ⟩
    (E ˘ / lbd Q) \ E ˘
  ≈⟨ ⊑-antisym (\-universal (⊑-begin
        Q ˘ ⨾ ((E ˘ / lbd Q) \ E ˘)
      ⊑⟨ ⨾-monotone₂ (\-antitone ⊑-S/∘\S) ⟩
        Q ˘ ⨾ (Q ˘ \ E ˘)
      ⊑⟨ \-cancel-outer ⟩
        E ˘
      □)) ⊑-\S∘S/ ⟩
    lbd Q
  □
```

5.2 Categoric.OSGC.Preorder.Extrema

With residuals and symmetric quotients, given an OSGC preorder we can discuss the notions of 'greatest elements' and least such 'elements'. Then go on to explore the notions of infima and suprema.

```
module Categoric.OSGC.Preorder.Extrema {i j k₁ k₂} {Obj : Set i} (osgc : OSGC j k₁ k₂ Obj)
  (let open OSGC osgc)
  (leftResOp  : LeftResOp orderedSemigroupoid)
  (rightResOp : RightResOp orderedSemigroupoid)
  (syqOp      : SyqOp osgc)
  where
open SyqOp                                                   syqOp
open SyQ-ResidualProps       osgc leftResOp rightResOp syqOp
open ResidualOps                  leftResOp rightResOp
open OSGC-Residuals          osgc leftResOp rightResOp
open PreorderWithResiduals osgc leftResOp rightResOp using (module IsPreorder′)
```

In conventional developments, as for example by Schmidt and Ströhlein (1993), gre and lea, the operators for greatest and least elements, are defined using meets, and an equivalent formulation using symmetric quotients is then derived. In our development, meets are not available, and we use the formulation based on symmetric quotients as our definitions.

```
module IsPreorder″ {A : Obj} {E : Mor A A} (E-isPreorder : IsPreorder osgc E) where
  open IsPreorder osgc E-isPreorder public
  open IsPreorder′ E-isPreorder public
```

5.2.1 gre, lea, and cones

```
gre : {I : Obj} → Mor I A → Mor I A
gre Q = (E ⨾ Q ˘) ╳ E

lea : {I : Obj} → Mor I A → Mor I A
lea Q = (E ˘ ⨾ Q ˘) ╳ E ˘

gre-cong : {I : Obj} {R S : Mor I A} → R ≈ S → gre R ≈ gre S
gre-cong R≈S = ╳-cong₁ (⨾-cong₂ (˘-cong R≈S))

lea-cong : {I : Obj} {R S : Mor I A} → R ≈ S → lea R ≈ lea S
lea-cong R≈S = ╳-cong₁ (⨾-cong₂ (˘-cong R≈S))
```

Following Furusawa and Kahl (1998), we prove certain cone properties.

```
gre-downcone : {I : Obj} {Q : Mor I A} → gre (Q ⨾ E ˘) ≈ gre Q
gre-downcone {I} {Q} = ≈-begin
    gre (Q ⨾ E ˘)
  ≈⟨ ╳-cong₁ ((⨾-cong₂ ˘-involutionRightConv)) ⟩
    (E ⨾ (E ⨾ Q ˘)) ╳ E
  ≈⟨ ╳-cong₁ (⨾-assocL ⟨≈≈⟩ ⨾-cong₁ idempot) ⟩
    gre Q
  □

lea-upcone : {I : Obj} {Q : Mor I A} → lea (Q ⨾ E) ≈ lea Q
lea-upcone {I} {Q} = ≈-begin
    lea (Q ⨾ E)
  ≈⟨ ╳-cong₁ (⨾-cong₂ ˘-involution) ⟩
    (E ˘ ⨾ (E ˘ ⨾ Q ˘)) ╳ E ˘
  ≈⟨ ╳-cong₁ (⨾-assocL ⟨≈≈⟩ ⨾-cong₁ ˘-idempot) ⟩
    lea Q
  □
```

5.2.2 lub and glb

Recall,

glb S = s

⇔ s is an upper bound of S and the least such upper bound
⇔ (∀ e | e ∈ S • e ≤ s) ∧ (∀ u | (∀ e | e ∈ S • e ≤ u) • s ≤ u)
⇔ ∀ u • u ≤ s ≡ (∀ e | e ∈ S • e ≤ u)
⇔ s (≤ ╳ (≥ / ∋) S

That is, S glb s ≡ s (≤ ╳ (≥ / ∋) S, i.e., glb = (≥ / ∋) ╳ ≤. Formally,

```
lub : {I : Obj} → Mor I A → Mor I A
lub Q = ubd Q ˘ ╳ E ˘
glb : {I : Obj} → Mor I A → Mor I A
glb Q = lbd Q ˘ ╳ E
lub-cong : {I : Obj} {R S : Mor I A} → R ≈ S → lub R ≈ lub S
lub-cong R≈S = ╳-cong₁ (˘-cong (ubd-cong R≈S))
glb-cong : {I : Obj} {R S : Mor I A} → R ≈ S → glb R ≈ glb S
glb-cong R≈S = ╳-cong₁ (˘-cong (lbd-cong R≈S))
```

The informal "least upper bound = least (upper bound) " takes a formal shape:

```
lea-ubd-≈-lub : {I : Obj} {Q : Mor I A} → lea (ubd Q) ≈ lub Q
lea-ubd-≈-lub {I} {Q} = ≈-begin
    (E ˘ ⨾ ubd Q ˘) ╳ E ˘
  ≈⟨ ╳-cong₁ (˘-involution ⟨≈˘≈⟩ ˘-cong ubd-upclosed) ⟩
    ubd Q ˘ ╳ E ˘
  □
```

and dually,

```
gre-lbd-≈-glb : {I : Obj} {Q : Mor I A} → gre (lbd Q) ≈ glb Q
gre-lbd-≈-glb {I} {Q} = ≈-begin
    (E ⨾ lbd Q ˘) ╳ E
  ≈⟨ ╳-cong₁ (˘-involutionRightConv ⟨≈˘≈⟩ ˘-cong lbd-downclosed) ⟩
    lbd Q ˘ ╳ E
  □
```

As is well known, infima and suprema are interdefinable. This still holds in our general setting.

```
lub-≈-glb-ubd : {I : Obj} {Q : Mor I A} → lub Q ≈ glb (ubd Q)
lub-≈-glb-ubd {I} {Q} = ≈-begin
    lub Q
  ≈⟨⟩
    ubd Q ˘ ╳ E ˘
  ≈⟨ ⊑-antisym
    (╳-universal
      (⊑-begin
          (E / ubd Q) ⨾ (ubd Q ˘ ╳ E ˘)
        ⊑⟨ ⨾-monotone₂ ˘╳˘-⊑-/ ⟩
          (E / ubd Q) ⨾ (ubd Q / E)
        ⊑⟨ /-cancel-middle ⟨⊑≈⟩ order-/ ⟩
          E
      □)
      (⊑-begin
          (ubd Q ˘ ╳ E ˘) ⨾ E ˘
        ⊑⟨ ⨾-monotone₁ ╳-⊑-\ ⟩
          (ubd Q ˘ \ E ˘) ⨾ E ˘
        ≈⟨ lbd-downclosed ⟩
          ubd Q ˘ \ E ˘
        ≈˘⟨ /-˘ ⟩
          (E / ubd Q) ˘
        □))
    (╳-universal
      (⊑-begin
          ubd Q ˘ ⨾ ((E / ubd Q) ╳ E)
        ⊑⟨ ⨾-monotone₂ (╳-⊑-/ ⟨⊑≈˘⟩ \-˘ ⟨⊑≈⟩ ˘-cong \/-≈) ⟩
          ubd Q ˘ ⨾ ((E \ E) / ubd Q) ˘
        ⊑⟨ ˘-involution ⟨≈˘⊑⟩ ˘-monotone (/-cancel-outer ⟨⊑≈⟩ order-\) ⟩
          E ˘
        □)
      ((⊑-begin
            ((E / ubd Q) ╳ E) ⨾ (E ˘) ˘
        ⊑⟨ ⨾-monotone ╳-⊑-\ (⊑-reflexive ˘˘) ⟩
            ((E / ubd Q) \ E) ⨾ E
        ≈⟨ ⨾-cong₁ \S∘S/∘\S ⟨≈≈⟩ ubd-upclosed ⟩
            ubd Q
        □) ⟨⊑≈˘⟩ ˘˘))
    ⟩
    (E / ubd Q) ╳ E
  ≈˘⟨ ╳-cong₁ lbd-ubd-˘ ⟩
    lbd (ubd Q) ˘ ╳ E
  ≈⟨⟩
    glb (ubd Q)
  □
```

Dually,

```
glb-≈-lub-lbd : {I : Obj} {Q : Mor I A} → glb Q ≈ lub (lbd Q)
glb-≈-lub-lbd {I} {Q} = ≈-begin
    glb Q
  ≈⟨⟩
    lbd Q ˘ ╳ E
  ≈⟨ ⊑-antisym
    (╳-universal
      (⊑-begin
          (E ˘ / lbd Q) ⨾ (lbd Q ˘ ╳ E)
        ⊑⟨ ⨾-monotone₂ ˘╳-⊑-/ ⟩
          (E ˘ / lbd Q) ⨾ (lbd Q / E ˘)
        ⊑⟨ /-cancel-middle ⟨⊑≈⟩ order˘-/ ⟩
          E ˘
      □)
      (⊑-begin
          (lbd Q ˘ ╳ E) ⨾ (E ˘) ˘
        ⊑⟨ ⨾-monotone ╳-⊑-\ (⊑-reflexive ˘˘) ⟩
          (lbd Q ˘ \ E) ⨾ E
        ≈⟨ ubd-upclosed ⟩
        lbd Q ˘ \ E
        ≈˘⟨ ˘/-˘ ⟩
          (E ˘ / lbd Q) ˘
        □))
    (╳-universal
      (⊑-begin
          lbd Q ˘ ⨾ ((E ˘ / lbd Q) ╳ E ˘)
        ⊑⟨ ⨾-monotone₂ (╳-⊑-/ ⟨⊑≈˘⟩ \-˘ ⟨⊑≈⟩ ˘-cong \/-≈) ⟩
          lbd Q ˘ ⨾ ((E ˘ \ E ˘) / lbd Q) ˘
        ⊑⟨ ˘-involution ⟨≈˘⊑⟩ ˘-monotone (/-cancel-outer ⟨⊑≈⟩ order˘-\) ⟨⊑≈⟩ ˘˘ ⟩
          E
        □)
      ((⊑-begin
          ((E ˘ / lbd Q) ╳ E ˘) ⨾ E ˘
        ⊑⟨ ⨾-monotone₁ ╳-⊑-\ ⟩
          ((E ˘ / lbd Q) \ E ˘) ⨾ E ˘
        ≈⟨ ⨾-cong₁ \S∘S/∘\S ⟨≈≈⟩ lbd-downclosed ⟩
          lbd Q
        □) ⟨⊑≈˘⟩ ˘˘))
    ⟩
    (E ˘ / lbd Q) ╳ E ˘
```

```
  ≈˘⟨ ╳-cong₁ ubd-lbd-˘ ⟩
    ubd (lbd Q) ˘ ╳ E ˘
  ≈⟨⟩
    lub (lbd Q)
  □
```

Whenever the least upper bound exists, above it (⨾ E) lie the upper bounds. Likewise for the greatest lower bound.

```
total-lub-⨾-order : {I : Obj} {Q : Mor I A} → isTotal (lub Q) → lub Q ⨾ E ≈ ubd Q
total-lub-⨾-order {I} {Q} lub-total = ≈-begin
    lub Q ⨾ E
  ≈˘⟨ ⨾-cong₂ ˘˘ ⟩
    (ubd Q ˘ ╳ E ˘) ⨾ (E ˘) ˘
  ≈⟨ ╳-total-cancel-right lub-total ⟨≈≈⟩ ˘˘ ⟩
    ubd Q
  □
total-glb-⨾-order˘ : {I : Obj} {Q : Mor I A} → isTotal (glb Q) → glb Q ⨾ E ˘ ≈ lbd Q
total-glb-⨾-order˘ {I} {Q} glb-total = ≈-begin
    glb Q ⨾ E ˘
  ≈⟨⟩
    (lbd Q ˘ ╳ E) ⨾ E ˘
  ≈⟨ ╳-total-cancel-right glb-total ⟨≈≈⟩ ˘˘ ⟩
    lbd Q
  □
```

The existence of one kind of suprema guarantees the existence of the other. That is, complete (internal) semi-lattices are precisely complete (internal) lattices.

```
total-glb→total-lub : {I : Obj} → ({Q : Mor I A} → isTotal (glb Q))
                                 → ({Q : Mor I A} → isTotal (lub Q))
total-glb→total-lub glb-total = ≈-isTotal lub-≈-glb-ubd glb-total
total-lub→total-glb : {I : Obj} → ({Q : Mor I A} → isTotal (lub Q))
                                 → ({Q : Mor I A} → isTotal (glb Q))
total-lub→total-glb lub-total = ≈-isTotal glb-≈-lub-lbd lub-total
```

5.2.3 A Risky Duality

Many of the proofs of this section appear very similar except for an odd converse here or there. In fact we can prove half of our theorems and obtain the others by duality, and so we only show this below. While such dualisation may be slick, it apparently occasionally involves significant overhead. The proof terms resulting from the dualisation are more than a constant overhead cost than those derived directly above.

Proof term line count:

name	direct	dual
lea-cong	19	19
lub-cong	1058	1058
gre-downcone	103	164
lea-ubd-≈-lub	105	325
glb-≈-lub-lbd	2569	2927
total-glb-⨾-order˘	347	370
total-lub→total-glb	10408	11868

For the examples listed, the "dualisation cost" as measured in normalised proof term size is below 20%, except for lea-ubd-≈-lub, where it is over 200%.

One possible reason for this is as follows. The proofs of monotonicity and cancellation are proved first, then the Galois characterization is proved from these. Then once the Galois module is opened with this connection, it generates its own proofs of monotonicity and cancellation which are then used to prove other results. Hence, the cost of, e.g., monotonicity and cancellation becomes much greater than their direct formulations.

Below is the approach proving half our results and applying duality to obtain the rest.

```
private
module gre-glb {A : Obj} {E : Mor A A} (E-isPreorder : IsPreorder osgc E) where
  open IsPreorder osgc E-isPreorder using (˘-idempot)
  open IsPreorder′ E-isPreorder public

  gre : {I : Obj} → Mor I A → Mor I A
  gre Q = (E ⨾ Q ˘) ╳ E
  lea : {I : Obj} → Mor I A → Mor I A
  lea Q = (E ˘ ⨾ Q ˘) ╳ E ˘
```

Recall,

glb S = s

⇔ s is an upper bound of S and the least such upper bound
⇔ (∀ e ∣ e ∈ S • e ≤ s) ∧ (∀ u ∣ (∀ e ∣ e ∈ S • e ≤ u) • s ≤ u)
⇔ ∀ u • u ≤ s ≡ (∀ e ∣ e ∈ S • e ≤ u)
⇔ s (≤ ╳ (≥ / ∋) S

That is, S glb s ≡ s (≤ ╳ (≥ / ∋) S, i.e., glb = (≥ / ∋) ╳ ≤. Formally,

```
  lub : {I : Obj} → Mor I A → Mor I A
  lub Q = ubd Q ˘ ╳ E ˘
  glb : {I : Obj} → Mor I A → Mor I A
  glb Q = lbd Q ˘ ╳ E
```

The expected congruence laws,

```
  gre-cong : {I : Obj} {R S : Mor I A} → R ≈ S → gre R ≈ gre S
  gre-cong R≈S = ╳-cong₁ (⨾-cong₂ (˘-cong R≈S))
  glb-cong : {I : Obj} {R S : Mor I A} → R ≈ S → glb R ≈ glb S
  glb-cong R≈S = ╳-cong₁ (˘-cong (lbd-cong R≈S))
```

Following Furusawa and Kahl (1998), we prove certain cone properties.

```
  lea-upcone : {I : Obj} {Q : Mor I A} → lea (Q ⨾ E) ≈ lea Q
  lea-upcone {I} {Q} = ≈-begin
      lea (Q ⨾ E)
    ≈⟨ ╳-cong₁ (⨾-cong₂ ˘-involution) ⟩
      (E ˘ ⨾ (E ˘ ⨾ Q ˘)) ╳ E ˘
    ≈⟨ ╳-cong₁ (⨾-assocL ⟨≈≈⟩ ⨾-cong₁ ˘-idempot) ⟩
      lea Q
    □
```

The informal "greatest lower bound = greatest (lower bound) " takes a formal shape:

```
  gre-lbd-≈-glb : {I : Obj} {Q : Mor I A} → gre (lbd Q) ≈ glb Q
  gre-lbd-≈-glb {I} {Q} = ≈-begin
      (E ⨾ lbd Q ˘) ╳ E
```

```
    ≈⟨ ⨟-cong₁ (˘-involutionRightConv ⟨≈˘≈⟩ ˘-cong lbd-downclosed) ⟩
      lbd Q ˘ ⨟ E
    □
```

As is well known, infima and suprema are interdefinable. This still holds in our general setting.

First we obtain some useful lemmas,

```
  /-⨾-˘⨟˘-⊑ : {I : Obj} {R : Mor I A} → (E / R) ⨾ (R ˘ ⨟ E ˘) ⊑ E
  /-⨾-˘⨟˘-⊑ {I} {R} = ⊑-begin
      (E / R) ⨾ (R ˘ ⨟ E ˘)
    ⊑⟨ ⨾-monotone₂ ˘⨟˘-⊑-/ ⟩
      (E / R) ⨾ (R / E)
    ⊑⟨ /-cancel-middle ⟨⊑≈⟩ order-/ ⟩
      E
    □
  ˘⨟˘-⨾-˘-⊑ : {I : Obj} {R : Mor I A} → (R ˘ ⨟ E ˘) ⨾ E ˘ ⊑ (E / R) ˘
  ˘⨟˘-⨾-˘-⊑ {I} {R} = ⊑-begin
      (R ˘ ⨟ E ˘) ⨾ E ˘
    ⊑⟨ ⨾-monotone₁ ⨟-⊑-\ ⟩
      (R ˘ \ E ˘) ⨾ E ˘
    ≈⟨ lbd-downclosed ⟩
      R ˘ \ E ˘
    ≈˘⟨ /-˘ ⟩
      (E / R) ˘
    □
  ˘⨟˘-⊑-/-⨟ : {I : Obj} {R : Mor I A} → R ˘ ⨟ E ˘ ⊑ (E / R) ⨟ E
  ˘⨟˘-⊑-/-⨟ = ⨟-universal /-⨾-˘⨟˘-⊑ ˘⨟˘-⨾-˘-⊑
  ˘-⨾-/-⨟-⊑ : {I : Obj} {R : Mor I A} → R ˘ ⨾ ((E / R) ⨟ E) ⊑ E ˘
  ˘-⨾-/-⨟-⊑ {I} {R} = ⊑-begin
      R ˘ ⨾ ((E / R) ⨟ E)
    ⊑⟨ ⨾-monotone₂ (⨟-⊑-/ ⟨⊑≈˘⟩ \-˘ ⟨⊑≈⟩ ˘-cong \/-≈) ⟩
      R ˘ ⨾ ((E \ E) / R) ˘
    ⊑⟨ ˘-involution ⟨≈˘⊑⟩ ˘-monotone (/-cancel-outer ⟨⊑≈⟩ order-\) ⟩
      E ˘
    □
```

Now we can obtain our desired result,

```
  lub-≈-glb-ubd : {I : Obj} {Q : Mor I A} → lub Q ≈ glb (ubd Q)
  lub-≈-glb-ubd {I} {Q} = ≈-begin
      lub Q
    ≈⟨⟩
      ubd Q ˘ ⨟ E ˘
    ≈⟨ ⊑-antisym ˘⨟˘-⊑-/-⨟
      (⨟-universal
        ˘-⨾-/-⨟-⊑
        ((⊑-begin
            ((E / ubd Q) ⨟ E) ⨾ (E ˘) ˘
          ⊑⟨ ⨾-monotone ⨟-⊑-\ (⊑-reflexive ˘˘) ⟩
            ((E / ubd Q) \ E) ⨾ E
          ≈⟨ ⨾-cong₁ \S∘S/∘\S ⟨≈≈⟩ ubd-upclosed ⟩
            ubd Q
          □) ⟨⊑≈˘⟩ ˘˘))
      ⟩
      (E / ubd Q) ⨟ E
    ≈˘⟨ ⨟-cong₁ lbd-ubd-˘ ⟩
      lbd (ubd Q) ˘ ⨟ E
    ≈⟨⟩
      glb (ubd Q)
    □
```

Under certain condition, the upper bounds are precisely that which the succesors of the least upper bound. Likewise for the greatest lower bound.

```
  total-lub-⨾-order : {I : Obj} {Q : Mor I A} → isTotal (lub Q) → lub Q ⨾ E ≈ ubd Q
  total-lub-⨾-order {I} {Q} lub-total = ≈-begin
      lub Q ⨾ E
    ≈˘⟨ ⨾-cong₂ ˘˘ ⟩
      (ubd Q ˘ ⨟ E ˘) ⨾ (E ˘) ˘
    ≈⟨ ⨟-total-cancel-right lub-total ⟨≈≈⟩ ˘˘ ⟩
      ubd Q
    □
```

The existence of one kind of suprema guarantees the existence of the other. That is, complete (internal) semi-lattices are precisely complete (internal) lattices.

```
  total-glb→total-lub : {I : Obj} → ({Q : Mor I A} → isTotal (glb Q))
                                  → ({Q : Mor I A} → isTotal (lub Q))
  total-glb→total-lub glb-total = ≈-isTotal lub-≈-glb-ubd glb-total

private
module lea-lub {A : Obj} {E : Mor A A} (E-isPreorder : IsPreorder osgc E) where
  open IsPreorder osgc E-isPreorder
  open gre-glb ˘-isPreorder₀ public renaming
    (ubd to lbd; lbd to ubd; lub to glb; glb to lub; lea to gre; gre to lea
    ; gre-cong to lea-cong   -- : ∀ {R S} → R ≈ S → lea R ≈ lea S
    ; glb-cong to lub-cong   -- : ∀ {R S} → R ≈ S → lub R ≈ lub S
    ; lea-upcone to gre-downcone   -- : ∀ {Q} → gre (Q ⨾ E ˘) ≈ gre Q
    ; gre-lbd-≈-glb to lea-ubd-≈-lub   -- : ∀ {Q} → lea (ubd Q) ≈ lub Q
    ; lub-≈-glb-ubd to glb-≈-lub-lbd   -- : ∀ {Q} → glb Q ≈ lub (lbd Q)
    ; total-lub-⨾-order to total-glb-⨾-order˘   -- : ∀ {Q} → isTotal (glb Q) → glb Q ⨾ E ˘ ≈ lbd Q
    ; total-glb→total-lub to total-lub→total-glb
        -- : ∀ {Q} → isTotal (lub Q)) → ({Q : Mor I A} → isTotal (glb Q))
    )
```

5.3 Categoric.OCC.Preorder

We now migrate to the setting of OCCs, thereby permitting ourselves the luxury of identities and witness how matters simplify.

```
module _ {i j k₁ k₂} {Obj : Set i} (occ : OCC j k₁ k₂ Obj) where
  open OCC occ

  record IsPreorder {A : Obj} (E : Mor A A) : Set k₂ where
    field
      refl : IsReflexive E
      trans : IsTransitive E
```

Needless to say, this is also a preorder in the underlying OSGC (ordered semigroupoid with converse).

```
isPreorder₀ : IsPreorder₀ osgc E
isPreorder₀ = record {supId = reflexiveIsSuperidentity refl; trans = trans}
open IsPreorder₀ osgc isPreorder₀ public hiding (trans)
```

Also, the converse of the preorder is again a preorder.

```
˘-refl : IsReflexive (E ˘)
˘-refl = ⊑-begin
    Id
  ≈˘⟨ Id˘ ⟩
    Id ˘
  ⊑⟨ ˘-monotone refl ⟩
    E ˘
  □

˘-isPreorder : IsPreorder (E ˘)
˘-isPreorder = record {refl = ˘-refl; trans = ˘-trans}
```

5.3.1 Retract Preorder and Preorder Invaraince

If $\_\preccurlyeq\_$ is a preorder then so is $x \preccurlyeq' y :\equiv f(x) \preccurlyeq f(y)$, for any mapping f. Formally,

```
module _ {A : Obj} {E : Mor A A} (E-isPreorder : IsPreorder E) {Z : Obj} (F : Mapping Z A)
  where
    open IsPreorder E-isPreorder
    F₀ = Mapping.mor F

    retractPreorder : IsPreorder (F₀ ⨾ E ⨾ F₀ ˘)
    retractPreorder = record
      {refl = isTotal-to-I (mappingTotal F) ⟨⊑⊑⟩ ⨾-monotone₂ (leftId ⟨≈˘⊑⟩ ⨾-monotone₁ refl)
      ;trans = ⊑-begin
            (F₀ ⨾ E ⨾ F₀ ˘) ⨾ (F₀ ⨾ E ⨾ F₀ ˘)
          ≈⟨ ⨾-assoc₃₊₁ ⟨≈≈⟩ ⨾-cong₂₂ ⨾-assocL ⟩
            F₀ ⨾ E ⨾ (F₀ ˘ ⨾ F₀) ⨾ E ⨾ F₀ ˘
          ⊑⟨ ⨾-monotone₂₂ (proj₁ (mappingUnivalent F)) ⟩
            F₀ ⨾ E ⨾ E ⨾ F₀ ˘
          ⊑⟨ ⨾-monotone₂ (⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ trans) ⟩
            F₀ ⨾ E ⨾ F₀ ˘
          □
      }
```

The notion of being a preorder is invariant under equivalence.

```
IsPreorder-subst : {A : Obj} {E₁ E₂ : Mor A A}
                 → E₁ ≈ E₂ → IsPreorder E₁ → IsPreorder E₂
IsPreorder-subst {A} {E₁} {E₂} E₁≈E₂ E₁-isPreorder = record
  {refl = refl ⟨⊑≈⟩ E₁≈E₂
  ;trans = ⨾-cong E₁≈E₂ E₁≈E₂ ⟨≈˘⊑⟩ trans ⟨⊑≈⟩ E₁≈E₂
  } where open IsPreorder E₁-isPreorder
```

Some useful subscripts.

```
module IsPreorder₁ {A : Obj} {E : Mor A A} (isPreorder : IsPreorder E) where
    open IsPreorder isPreorder public using () renaming
      (
        refl          to refl₁
      ; trans         to trans₁
      ; idempot       to idempot₁
      ; leftSupId     to leftSupId₁
      ; rightSupId    to rightSupId₁
      ; ˘-leftSupId   to ˘-leftSupId₁
      ; ˘-rightSupId  to ˘-rightSupId₁
      )
module IsPreorder₂ {A : Obj} {E : Mor A A} (isPreorder : IsPreorder E) where
    open IsPreorder isPreorder public using () renaming
      (
        refl          to refl₂
      ; trans         to trans₂
      ; idempot       to idempot₂
      ; leftSupId     to leftSupId₂
      ; rightSupId    to rightSupId₂
      ; ˘-leftSupId   to ˘-leftSupId₂
      ; ˘-rightSupId  to ˘-rightSupId₂
      )
```

5.3.2 Residual Induced Preorders

The power of residuals yields more opportunities,

```
module PreorderWithResiduals
      (leftResOp  : LeftResOp orderedSemigroupoid)
      (rightResOp : RightResOp orderedSemigroupoid) where
  open ResidualOps leftResOp rightResOp
  open OrdCat-Residual-Props orderedCategory leftResOp rightResOp
  open OSGC-Residuals osgc leftResOp rightResOp
  module IsPreorder′ {A : Obj} {E : Mor A A} (E-isPreorder : IsPreorder E) where
    open IsPreorder E-isPreorder
    open OSGC-PreorderWithResiduals.IsPreorder′ osgc leftResOp rightResOp isPreorder₀ public
```

namely every morphism induces a preorder:

```
  \-isPreorder : {A B : Obj} (R : Mor A B) → IsPreorder (R \ R)
  \-isPreorder R = record {refl = \-isReflexive; trans = \-cancel-middle}
  \-isPreorder₀ : {A B : Obj} (R : Mor A B) → IsPreorder₀ osgc (R \ R)
  \-isPreorder₀ R = record
    {supId = reflexiveIsSuperidentity \-isReflexive
    ;trans = \-cancel-middle
    }
  module \-Preorder {A B : Obj} (R : Mor A B) where
    E : Mor B B
    E = R \ R
    isPreorder : IsPreorder E
    isPreorder = \-isPreorder R
    open IsPreorder′ isPreorder public
    ubdE≈\ : {I : Obj} {Q : Mor I B} → ubd Q ≈ (R ⨾ Q ˘) \ R
    ubdE≈\ {I} {Q} = ≈-begin
        Q ˘ \ (R \ R)
      ≈⟨ \\ ⟩
        (R ⨾ Q ˘) \ R
      □
    ubd˘E≈/ : {I : Obj} {Q : Mor I B} → ubd Q ˘ ≈ R ˘ / (Q ⨾ R ˘)
    ubd˘E≈/ {I} {Q} = ≈-begin
        ubd Q ˘
      ≈⟨ ˘-cong ubdE≈\ ⟩
        ((R ⨾ Q ˘) \ R) ˘
      ≈⟨ \-˘ ⟩
        R ˘ / (R ⨾ Q ˘) ˘
      ≈⟨ /-cong₂ ˘-involutionRightConv ⟩
        R ˘ / (Q ⨾ R ˘)
      □
    lbd-ubd-≈-twist : {I : Obj} {Q : Mor I B} → lbd (ubd Q) ≈ (R ˘ / (Q ⨾ R ˘)) \ (R ˘ / R ˘)
```

```
lbd-ubd-≈-twist {I} {Q} = ≈-begin
      lbd (ubd Q)
   ≈⟨ \-cong₂ \-˘ ⟩
      (ubd Q) ˘ \ (R ˘ / R ˘)
   ≈⟨ \-cong₁ ubd˘E≈/ ⟩
      (R ˘ / (Q ⨾ R ˘)) \ (R ˘ / R ˘)
   □
```

It is to be noted that the reflexitivity proof here could not be expressed as a superidentity and as such these induced preorders could not be within an OSGC setting. It is interesting to observe that the concepts of super- and sub-identities are not as expressive as the notion of reflexitivity.

5.4 Categoric.OCC.Order

In an OCC with residuals and symmetric quotients, we can present the notion of antisymmetry and so may investigate internal partial orders.

```
module Categoric.OCC.Order {i j k₁ k₂} {Obj : Set i} (occ : OCC j k₁ k₂ Obj)
  (let open OCC occ) (leftResOp : LeftResOp orderedSemigroupoid)
  (rightResOp : RightResOp orderedSemigroupoid) (syqOp : SyqOp osgc)
  where
open SyqOp                                                    syqOp
open OCC-SyQ-Props         occ                                syqOp
open SyQ-ResidualProps     osgc            leftResOp rightResOp syqOp
open ResidualOps                           leftResOp rightResOp
open OrdCat-Residual-Props orderedCategory leftResOp rightResOp
open OSGC-Residuals        osgc            leftResOp rightResOp
open PreorderWithResiduals occ             leftResOp rightResOp
  using (module IsPreorder′; module \-Preorder)
open import Categoric.OSGC.Preorder.Extrema osgc leftResOp rightResOp public
```

5.4.1 IsOrder Definition

```
record IsOrder {A : Obj} (E : Mor A A) : Set k₂ where
  field
    refl    : IsReflexive E
    trans   : IsTransitive E
    antisym : E ╳ E ⊑ Id
```

As argued earlier, antisymmetry is expressed pointfree as a symmetric quotient and so that is what we shall employ.

Of course this merely extends the notion of a preorder, and its variants,

```
  isPreorder : IsPreorder occ E
  isPreorder = record {refl = refl; trans = trans}
  open IsPreorder occ isPreorder public using (˘-refl; isPreorder₀; ˘-isPreorder)
  open IsPreorder″ syqOp isPreorder₀ public hiding (trans)
```

With these tools in hand, we can rephrase antisymmetry as an equivalence:

```
  antisym≈ : E ╳ E ≈ Id
  antisym≈ = ⊑-antisym antisym noy-isReflexive
```

```
  ˘-antisym≈ : E ˘ ╳ E ˘ ≈ Id
  ˘-antisym≈ = ⊑-antisym (⊑-begin
      E ˘ ╳ E ˘
    ⊑⟨ ╳-universal
        (⊑-begin
            E ⨾ (E ˘ ╳ E ˘)
          ⊑⟨ ⨾-monotone₂ ˘╳˘-⊑-/ ⟩
            E ⨾ (E / E)
          ⊑⟨ ⨾-cong₂ order-/ ⟨≈⊑⟩ trans ⟩
            E
          □)
        (⊑-begin
            (E ˘ ╳ E ˘) ⨾ E ˘
          ⊑⟨ ⨾-monotone₁ ╳-⊑-\ ⟩
            (E ˘ \ E ˘) ⨾ E ˘
          ⊑⟨ ⨾-cong₁ order˘-\ ⟨≈⊑⟩ ˘-trans ⟩
            E ˘
          □)
        ⟩
        E ╳ E
    ⊑⟨ antisym ⟩
        Id
    □) noy-isReflexive
```

Then, as expected, the converse morphism is also an order.

```
  ˘-isOrder : IsOrder (E ˘)
  ˘-isOrder = record {refl = ˘-refl    ; trans = ˘-trans; antisym = ⊑-reflexive ˘-antisym≈}
```

5.4.2 Indirect Equality

As mentioned in the introduction, the notion of indirect equality is of great import to order theory. Without it, certain results can only be phrased as indirect equivalence and not true equalities. We can now rectify the situation,

By massaging the notion of function equality with the aim of introducing symmetric quotients, we may obtain a point-free formulation as follows:

```
   f ≈ g
 ≡⟨ extensionality ⟩
   ∀ x • f (x) ≈ g (x)
 ≡⟨ equality ⟩
   ∀ x • ∀ y • f (x) ≈ y ≡ g (x) ≈ y
 ≡⟨ indirect equality ⟩
   ∀ x • ∀ y • (∀ z • z ≼ f (x) ≡ z ≼ y) ≡ (∀ z • z ≼ g (x) ≡ z ≼ y)
 ≡⟨ symmetric quotients ⟩
   ∀ x • ∀ y • x (≼ ⨾ f ˘ ╳ ≼) y ≡ x (≼ ⨾ g ˘ ╳ ≼) y
 ≡⟨ extensionality ⟩
   ≼ ⨾ f ˘ ╳ ≼  =  ≼ ⨾ g ˘ ╳ ≼
```

Formally,

```
  indirect-≈₀ : {B : Obj} {F G : Mor B A}
    → isMapping F → isMapping G → (E ⨾ F ˘) ╳ E ≈ (E ⨾ G ˘) ╳ E → F ≈ G
  indirect-≈₀ {B} {F} {G} F-map G-map indir = ≈-begin
      F
    ≈˘⟨ rightId ⟩
      F ⨾ Id
    ≈˘⟨ ⨾-cong₂ antisym≈ ⟩
```

```
    F ⨾ (E ≬ E)
  ≈⟨ ≬-in-left F-map ⟩
    (E ⨾ F ˘) ≬ E
  ≈⟨ indir ⟩
    (E ⨾ G ˘) ≬ E
  ≈⟨ ≬-in-left G-map ⟨≈˘≈⟩ (⨾-cong2 antisym≈ ⟨≈≈⟩ rightId) ⟩
    G
  □
```

Had we considered different indirect inclusions in the above motivating derivation, say the first from the left and the second from the right,

$$(\forall\ z \bullet z \preccurlyeq f\ (x) \equiv z \preccurlyeq y) \equiv (\forall\ z \bullet g\ (x) \preccurlyeq z \equiv y \preccurlyeq z)\ ,$$

then the resulting formalization would be:

```
indirect-≈-˘ : {F G : Mor A A}
  → isMapping F → isMapping G → (E ⨾ F ˘) ≬ E ≈ E ≬ (E ⨾ G ˘) → F ≈ G ˘
indirect-≈-˘ {F} {G} F-map G-map indir = ≈-begin
    F
  ≈˘⟨ rightId ⟩
    F ⨾ Id
  ≈˘⟨ ⨾-cong2 antisym≈ ⟩
    F ⨾ (E ≬ E)
  ≈⟨ ≬-in-left F-map ⟩
    (E ⨾ F ˘) ≬ E
  ≈⟨ indir ⟩
    E ≬ (E ⨾ G ˘)
  ≈⟨ ≬-M-in-right G-map ⟨≈˘≈⟩ ⨾-cong1 antisym≈ ⟨≈≈⟩ leftId ⟩
    G ˘
  □
```

Of course, we can explore other variant with the converse order:

```
˘-indirect-≈0 : {B : Obj} {F G : Mor B A}
  → isMapping F → isMapping G → (E ˘ ⨾ F ˘) ≬ E ˘ ≈ (E ˘ ⨾ G ˘) ≬ E ˘ → F ≈ G
˘-indirect-≈0 {B} {F} {G} F-map G-map indir =
  rightId ⟨≈˘≈˘⟩ ⨾-cong2 ˘-antisym≈ ⟨≈≈⟩ ≬-in-left F-map ⟨≈≈⟩ indir
  ⟨≈≈˘⟩ ≬-in-left G-map ⟨≈≈⟩ ⨾-cong2 ˘-antisym≈ ⟨≈≈⟩ rightId
˘-indirect-≈-˘ : {F G : Mor A A}
  → isMapping F → isMapping G → (E ˘ ⨾ F ˘) ≬ E ˘ ≈ E ˘ ≬ (E ˘ ⨾ G ˘) → F ≈ G ˘
˘-indirect-≈-˘ {F} {G} F-map G-map indir =
    rightId ⟨≈˘≈˘⟩ ⨾-cong2 ˘-antisym≈ ⟨≈≈⟩ ≬-in-left F-map ⟨≈≈⟩ indir
    ⟨≈≈˘⟩ ≬-M-in-right G-map ⟨≈≈⟩ ⨾-cong1 ˘-antisym≈ ⟨≈≈⟩ leftId
```

However, the most straightforward approach would be

$$(\forall\ x \bullet f\ x \approx g\ x) \equiv (\forall\ x, z \bullet f\ (x) \preccurlyeq z \equiv g\ (x) \preccurlyeq z)$$

— compare with the point-level, rather than morphism level, presentation of Relation.Binary.Poset.Renamed (Sect. 2.1) — and to this end we formulate some lemmas and formalise the desideratum as indirect-≈.

```
˘-indirect-⊑-⨾ : {B : Obj} {F G : Mor B A}
  → isMapping F → isMapping G → Id ⊑ (E ˘ ⨾ F ˘) ≬ (E ˘ ⨾ G ˘) → Id ⊑ F ⨾ G ˘
˘-indirect-⊑-⨾ {B} {F} {G} map-F map-G indir = ⊑-begin
    Id
  ⊑⟨ indir ⟩
    (E ˘ ⨾ F ˘) ≬ (E ˘ ⨾ G ˘)
  ≈˘⟨ ≬-M-in-right map-G ⟩
    ((E ˘ ⨾ F ˘) ≬ E ˘) ⨾ G ˘
  ≈⟨ ⨾-cong1 (≬-in-left map-F ⟨≈˘≈⟩ (⨾-cong2 ˘-antisym≈ ⟨≈≈⟩ rightId)) ⟩
    F ⨾ G ˘
  □
˘-indirect-⊑ : {B : Obj} {F G : Mor B A}
  → isMapping F → isMapping G → Id ⊑ (E ˘ ⨾ F ˘) ≬ (E ˘ ⨾ G ˘) → G ⊑ F
˘-indirect-⊑ {B} {F} {G} map-F map-G indir = leftId ⟨≈˘⊑⟩
  swap-⊑-⨾-unival˘ (proj1 map-G) (˘-indirect-⊑-⨾ map-F map-G indir)
indirect-≈ : {B : Obj} {F G : Mor B A}
  → isMapping F → isMapping G → F ⨾ E ≈ G ⨾ E → F ≈ G
indirect-≈ {B} {F} {G} map-F map-G indir = ⊑-antisym
  (˘-indirect-⊑ map-G map-F (noy-isReflexive ⟨⊑≈⟩ ≬-cong1 indir˘))
  (˘-indirect-⊑ map-F map-G (noy-isReflexive ⟨⊑≈˘⟩ ≬-cong1 indir˘))
  where indir˘ : E ˘ ⨾ F ˘ ≈ E ˘ ⨾ G ˘
        indir˘ = ˘-involution ⟨≈˘≈⟩ ˘-cong indir ⟨≈≈⟩ ˘-involution
```

Compare this with the indirect inclusion of preorders, Sect. 5.1.2.

Of course, we can explore other variants with the converse order:

```
indirect-⊑-⨾ : {B : Obj} {F G : Mor B A}
  → isMapping F → isMapping G → Id ⊑ (E ⨾ F ˘) ≬ (E ⨾ G ˘) → Id ⊑ F ⨾ G ˘
indirect-⊑-⨾ {B} {F} {G} map-F map-G indir = ⊑-begin
    Id
  ⊑⟨ indir ⟩
    (E ⨾ F ˘) ≬ (E ⨾ G ˘)
  ≈˘⟨ ≬-M-in-right map-G ⟩
    ((E ⨾ F ˘) ≬ E) ⨾ G ˘
  ≈⟨ ⨾-cong1 (≬-in-left map-F ⟨≈˘≈⟩ (⨾-cong2 antisym≈ ⟨≈≈⟩ rightId)) ⟩
    F ⨾ G ˘
  □
indirect-⊑ : {B : Obj} {F G : Mor B A}
  → isMapping F → isMapping G → Id ⊑ (E ⨾ F ˘) ≬ (E ⨾ G ˘) → G ⊑ F
indirect-⊑ {B} {F} {G} map-F map-G indir = leftId ⟨≈˘⊑⟩
  swap-⊑-⨾-unival˘ (proj1 map-G) (indirect-⊑-⨾ map-F map-G indir)
indirect-˘-≈ : {B : Obj} {F G : Mor B A}
  → isMapping F → isMapping G → E ⨾ F ˘ ≈ E ⨾ G ˘ → F ≈ G
indirect-˘-≈ {B} {F} {G} map-F map-G indir = ⊑-antisym
  (indirect-⊑ map-G map-F (noy-isReflexive ⟨⊑≈⟩ ≬-cong1 indir))
  (indirect-⊑ map-F map-G (noy-isReflexive ⟨⊑≈˘⟩ ≬-cong1 indir))
˘-indirect-≈ : {B : Obj} {F G : Mor B A}
  → isMapping F → isMapping G → F ⨾ E ˘ ≈ G ⨾ E ˘ → F ≈ G
˘-indirect-≈ map-F map-G indir = indirect-˘-≈ map-F map-G
  (˘-involutionRightConv ⟨≈˘≈⟩ (˘-cong indir ⟨≈≈⟩ ˘-involutionRightConv))
```

5.4.3 Univalence

Since, in an order, the down-cones (respectively up-cones) of different elements are different, we obtain univalence for the following symmetric quotients:

```
≬-order-univalentI : {B : Obj} {R : Mor A B} → isUnivalentI (R ≬ E)
≬-order-univalentI {I} {R} = ⊑-begin
    (R ≬ E) ˘ ⨾ (R ≬ E)
  ≈⟨ ⨾-cong1 ≬-˘ ⟩
    (E ≬ R) ⨾ (R ≬ E)
  ⊑⟨ ≬-cancel-middle ⟩
    E ≬ E
```

```
    ≈⟨ antisym≈ ⟩
      Id
    □
  ⨾-order-univalent : {B : Obj} {R : Mor A B} → isUnivalent (R ⨾ E)
  ⨾-order-univalent = isUnivalent-from-I ⨾-order-univalentI
  ⨾-order˘-univalentI : {B : Obj} {R : Mor A B} → isUnivalentI (R ⨾ E ˘)
  ⨾-order˘-univalentI {I} {R} = ⊑-begin
      (R ⨾ E ˘) ˘ ⨾ (R ⨾ E ˘)
    ≈⟨ ⨾-cong₁ ⨾-˘ ⟩
      (E ˘ ⨾ R) ⨾ (R ⨾ E ˘)
    ⊑⟨ ⨾-cancel-middle ⟩
      E ˘ ⨾ E ˘
    ≈⟨ ˘-antisym≈ ⟩
      Id
    □
  ⨾-order˘-univalent : {B : Obj} {R : Mor A B} → isUnivalent (R ⨾ E ˘)
  ⨾-order˘-univalent = isUnivalent-from-I ⨾-order˘-univalentI
```

5.4.4 Extrema

With the added power of antisymmetry, we obtain a host of new results concerning extrema. For starters, certain extrema of the order are precisely the identity:

```
  lub-order : lub (E ˘) ≈ Id
  lub-order = ≈-begin
      ubd (E ˘) ˘ ⨾ E ˘
    ≈⟨ ⨾-cong₁ (˘-cong ubd-order˘) ⟩
      E ˘ ⨾ E ˘
    ≈⟨ ˘-antisym≈ ⟩
      Id
    □
  glb-order : glb E ≈ Id
  glb-order = ≈-begin
      lbd E ˘ ⨾ E
    ≈⟨ ⨾-cong₁ (˘-cong lbd-order ⟨≈≈⟩ ˘˘) ⟩
      E ⨾ E
    ≈⟨ antisym≈ ⟩
      Id
    □
```

Next, mappings are fixed-points of extrema.

```
  lub-mapping : {I : Obj} {R : Mor I A} → isMapping R → lub R ≈ R
  lub-mapping {I} {R} R-map = ≈-begin
      lub R
    ≈⟨⟩
      ubd R ˘ ⨾ E ˘
    ≈⟨ ⨾-cong₁ (˘-cong (ubd-mapping R-map) ⟨≈≈⟩ ˘-involution) ⟩
      (E ˘ ⨾ R ˘)     ⨾ E ˘
    ≈˘⟨ ⨾-in-left R-map ⟩
      R ⨾ (E ˘ ⨾ E ˘)
    ≈⟨ ⨾-cong₂ ˘-antisym≈ ⟨≈≈⟩ rightId ⟩
      R
    □
  glb-mapping : {I : Obj} {R : Mor I A} → isMapping R → glb R ≈ R
  glb-mapping {I} {R} R-map = ≈-begin
      glb R
    ≈⟨⟩
      lbd R ˘ ⨾ E
    ≈⟨ ⨾-cong₁ (˘-cong (lbd-mapping R-map) ⟨≈≈⟩ ˘-involutionRightConv) ⟩
      (E ⨾ R ˘) ⨾ E
    ≈˘⟨ ⨾-in-left R-map ⟩
      R ⨾ (E ⨾ E)
    ≈⟨ ⨾-cong₂ antisym≈ ⟨≈≈⟩ rightId ⟩
      R
    □
```

Additionally, extrema are always univalent.

```
  lub-isUnivalentI : {I : Obj} {R : Mor I A} → isUnivalentI (lub R)
  lub-isUnivalentI {I} {R} = ⨾-order˘-univalentI
  lub-isUnivalent : {I : Obj} {R : Mor I A} → isUnivalent (lub R)
  lub-isUnivalent = isUnivalent-from-I lub-isUnivalentI
  glb-isUnivalentI : {I : Obj} {R : Mor I A} → isUnivalentI (glb R)
  glb-isUnivalentI {I} {R} = ⨾-order-univalentI
  glb-isUnivalent : {I : Obj} {R : Mor I A} → isUnivalent (glb R)
  glb-isUnivalent = isUnivalent-from-I glb-isUnivalentI
```

5.4.5 Order Constructions

Let us turn to certain order constructions. Namely, promoting a preorder to an order, a congruence of the order property, and a suborder construction.

An antisymmetric preorder is an order:

```
fromPreorder :       {A : Obj} {E : Mor A A} → IsPreorder occ E → (E ⨾ E ⊑ Id) → IsOrder E
fromPreorder E-isPreorder E⨾E⊑Id = record {refl = refl; trans = trans; antisym = E⨾E⊑Id}
  where open IsPreorder occ E-isPreorder
```

If a morphism is an order and it is equivalent to another morphism, then that too is an order. That is, the property of being an order respects equivalence.

```
IsOrder-subst  : {A : Obj} {E₁ E₂ : Mor A A} → E₁ ≈ E₂ → IsOrder E₁ → IsOrder E₂
IsOrder-subst {A} {E₁} {E₂} E₁≈E₂ E₁-isOrder = record
    {refl = refl ⟨⊑≈⟩ E₁≈E₂
    ; trans = ⨾-cong E₁≈E₂ E₁≈E₂ ⟨≈˘⊑⟩ trans ⟨⊑≈⟩ E₁≈E₂
    ; antisym = ⨾-cong E₁≈E₂ E₁≈E₂ ⟨≈˘⊑⟩ antisym
    }
    where open IsOrder E₁-isOrder
```

For a injective mapping f and an order _≼_, we again have an order x ≼′ y :≡ f (x) ≼ f (y). Formally,

```
module SubOrder {A : Obj} {E : Mor A A} (E-isOrder : IsOrder E)
               {Z : Obj} (F : Mapping Z A) (F-inj : isInjective (Mapping.mor F)) where
  open IsOrder E-isOrder
  private
    F₀ = Mapping.mor F
    F-isM = Mapping.prf F
    F-unival = mappingUnivalent F
  open IsPreorder₂ occ (retractPreorder occ isPreorder F)
  subOrder : Mor Z Z
  subOrder = F₀ ⨾ E ⨾ F₀ ˘
  subOrder-isOrder     : IsOrder subOrder
  subOrder-isOrder = record
```

```
{refl = refl₂; trans = trans₂
; antisym = ⊑-begin
        (F₀ ⨾ E ⨾ F₀ ˘) ╳ (F₀ ⨾ E ⨾ F₀ ˘)
    ≈⟨ ╳-cong ⨾-assocL ⨾-assocL ⟩
        ((F₀ ⨾ E) ⨾ F₀ ˘) ╳ ((F₀ ⨾ E) ⨾ F₀ ˘)
    ≈⟨ ╳-in-left F-isM ⟨≈˘≈˘⟩ ⨾-cong₂ (╳-in-right (˘-isBijective F-isM)) ⟩
        F₀ ⨾ ((F₀ ⨾ E) ╳ (F₀ ⨾ E)) ⨾ F₀ ˘
    ⊑⟨ retract╳ rightSupId rightSupId
      (⊑-begin
          (E ⨾ F₀ ˘) ⨾ (F₀ ⨾ E) ⨾ F₀ ˘
        ⊑⟨ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ (⨾-₁₂₁assoc₂₂ ⟨≈⊑⟩ proj₁ F-unival) ⟩
          E ⨾ E ⨾ F₀ ˘
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ trans ⟩
          E ⨾ F₀ ˘
        □)
      (⊑-begin
          F₀ ⨾ (F₀ ⨾ E) ˘ ⨾ (E ⨾ F₀ ˘) ˘
        ≈⟨ ⨾-cong₂ (⨾-cong ˘-involution ˘-involutionRightConv ⟨≈≈⟩ ⨾-assoc) ⟩
          F₀ ⨾ E ˘ ⨾ F₀ ˘ ⨾ F₀ ⨾ E ˘
        ⊑⟨ ⨾-monotone₂₂ (⨾-assocL ⟨≈⊑⟩ proj₁ F-unival) ⟩
          F₀ ⨾ E ˘ ⨾ E ˘
        ⊑⟨ ⨾-monotone₂ ˘-trans ⟩
          F₀ ⨾ E ˘
        ≈˘⟨ ˘-involutionRightConv ⟩
          (E ⨾ F₀ ˘) ˘
        □)
      ⟩
        ((E ⨾ F₀ ˘) ╳ (E ⨾ F₀ ˘))
    ≈⟨   ╳-in-left F-isM ⟨≈˘≈˘⟩ ⨾-cong₂ (╳-in-right (˘-isBijective F-isM)) ⟩
        F₀ ⨾ (E ╳ E) ⨾ F₀ ˘
    ⊑⟨ ⨾-cong₂ (⨾-cong₁ antisym≈ ⟨≈≈⟩ leftId) ⟨≈⊑⟩ isInjective-to-I F-inj ⟩
        Id
    □
}
```

5.4.6 Preorders Induced By Residuals and Endowed with Syqs

Just as before, residuals induce a preorder.

```
module \-Preorder″ {A B : Obj} (R : Mor A B) where
  open \-Preorder R using (E; isPreorder)
  open IsPreorder occ isPreorder using (isPreorder₀)
  open IsPreorder″ syqOp isPreorder₀
  ╳-\preorder : E ╳ E ⊑ R ╳ R
  ╳-\preorder = ╳-universal
          (⊑-begin
              R ⨾ (E ╳ E)
            ⊑⟨ ⨾-monotone₂ (╳-⊑-\ ⟨⊑≈⟩ order-\) ⟩
              R ⨾ (R \ R)
            ⊑⟨ \-cancel-outer ⟩
              R
            □)
          (⊑-begin
              (E ╳ E) ⨾ R ˘
            ⊑⟨ ⨾-monotone₁ (╳-⊑-/ ⟨⊑≈⟩ order˘-/) ⟩
              (R \ R) ˘ ⨾ R ˘
            ⊑⟨ ˘-involution ⟨≈˘⊑⟩ ˘-monotone \-cancel-outer ⟩
              R ˘
            □)
```

If, for a moment, we think of R as a membership relation ∈, then we have

$$\mathsf{x\ wrap\ y} \Leftrightarrow (\forall\ \mathsf{z} \bullet \mathsf{z} \approx \mathsf{x} \equiv \mathsf{z} \in \mathsf{y}) \Leftrightarrow \{\mathsf{x}\} \approx \mathsf{y}$$

Formally, and generally,

```
wrap : Mor A B
wrap = Id ╳ R
wrap-injective : isInjectiveI wrap
wrap-injective =    ⊑-begin
        (Id ╳ R) ⨾ (Id ╳ R) ˘
    ≈⟨ ⨾-cong₂ ╳-˘ ⟩
        (Id ╳ R) ⨾ (R ╳ Id)
    ⊑⟨ ╳-cancel-middle ⟩
        Id ╳ Id
    ≈⟨ noy-Id ⟩
        Id
    □
wrap⨾R˘ : isTotal wrap → wrap ⨾ R ˘ ≈ Id
wrap⨾R˘ total = ≈-begin
        wrap ⨾ R ˘
      ≈⟨⟩
        (Id ╳ R) ⨾ R ˘
      ≈⟨ ╳-total-cancel-right total ⟩
        Id ˘
      ≈⟨ Id˘ ⟩
        Id
      □
R⨾wrap˘ : isTotal wrap → R ⨾ wrap ˘ ≈ Id
R⨾wrap˘ total = ˘-involutionRightConv ⟨≈˘≈⟩ ˘-cong (wrap⨾R˘ total) ⟨≈≈⟩ Id˘
wrap⨾E : isMapping wrap → wrap ⨾ (R \ R) ≈ R
wrap⨾E map = ≈-begin
        wrap ⨾ (R \ R)
    ≈⟨ \-inner-⨾ map ⟩
        (R ⨾ wrap ˘) \ R
    ≈⟨ \-cong₁ (R⨾wrap˘ (proj₂ map)) ⟩
        Id \ R
    ≈⟨ Id-\ ⟩
        R
    □
R⨾-R\ : {C : Obj} {Q : Mor A C} → isTotal wrap → R ⨾ (R \ Q) ≈ Q
R⨾-R\ {C} {Q} total = ⊑-antisym \-cancel-outer (⊑-begin
        Q
    ≈⟨ leftId ⟨≈˘≈⟩ (⨾-cong₁ (R⨾wrap˘ total) ⟨≈˘≈⟩ ⨾-assoc) ⟩
        R ⨾ wrap ˘ ⨾ Q
    ⊑⟨ ⨾-monotone₂ (\-universal (⊑-reflexive (⨾-assocL ⟨≈≈⟩ ⨾-cong₁ (R⨾wrap˘ total) ⟨≈≈⟩ leftId))) ⟩
        R ⨾ (R \ Q)
    □)
```

Note that this last result is a form of ‘exact division’.

5.4.7 Orders Induced by Residuation and Endowed with Comprehension

```
module \-OrderWithComprehension {A B : Obj} {R : Mor A B}
  (isOrder : IsOrder (R \ R)) (comprehensiveI : {C : Obj} {Q : Mor A C} → isTotalI (Q ╳ R))
```

```
where
open \-Preorder′′ R
open IsOrder isOrder
comprehensive : {C : Obj} {Q : Mor A C} → isTotal (Q ╳ R)
comprehensive = isTotal-from-I comprehensiveI
Ω : Mor B B
Ω = R \ R
Ω˘ : Ω ˘ ≈ R ˘ / R ˘
Ω˘ = \-˘
R⨾-R╳ : {C : Obj} {Q : Mor A C} → R ⨾ (R ╳ Q) ≈ Q
R⨾-R╳ {C} {Q} = ╳-surjective-cancel-left (isSurjectiveFromTotal (≈-isTotal ╳-˘ comprehensive))
```

lubΩ≈╳ is (Furusawa and Kahl, 1998, Prop. 9.8(i)).

```
lubΩ≈╳ : {I : Obj} {Q : Mor I B} → lub Q ≈ (R ⨾ Q ˘) ╳ R
lubΩ≈╳ {I} {Q} = ≈-sym (total⊑unival-≈ comprehensive lub-isUnivalent (⊑-begin
        (R ⨾ Q ˘) ╳ R
    ⊑⟨ ╳⊑\˘╳\˘ ⟩
        ((R ⨾ Q ˘) \ R) ˘ ╳ (R \ R) ˘
    ≈⟨      ╳-cong (\-˘ ⟨≈≈⟩ /-cong2 ˘-involutionRightConv) \-˘ ⟩
        (R ˘ / (Q ⨾ R ˘)) ╳ (R ˘ / R ˘)
    ≈˘⟨       ╳-cong1 // ⟩
        ((R ˘ / R ˘) / Q) ╳ (R ˘ / R ˘)
    ≈˘⟨       ╳-cong (/-cong1 Ω˘) Ω˘ ⟩
        (Ω ˘ / Q) ╳ Ω ˘
    ≈˘⟨ ╳-cong1 ˘\-˘ ⟩
        (Q ˘ \ Ω) ˘ ╳ Ω ˘
    ≈⟨⟩
        ubd Q ˘ ╳ Ω ˘
    ≈⟨⟩
        lub Q
    □))
lubΩ-totalI : {I : Obj} {Q : Mor I B} → isTotalI (lub Q)
lubΩ-totalI = ≈-isTotalI lubΩ≈╳    comprehensiveI
lubΩ-total : {I : Obj} {Q : Mor I B} → isTotal (lub Q)
lubΩ-total = ≈-isTotal lubΩ≈╳ comprehensive
lubΩ : {I : Obj} (Q : Mor I B) → Mapping I B
lubΩ Q = record {mor = lub Q; prf = lub-isUnivalent, lubΩ-total}
```

The statement glb Q ≈ (lbd Q) ˘ ╳ Ω of (Furusawa and Kahl, 1998, Prop. 9.8(ii)) here holds definitionally.

```
glbΩ-totalI : {I : Obj} {Q : Mor I B} → isTotalI (glb Q)
glbΩ-totalI = ≈-isTotalI glb-≈-lub-lbd lubΩ-totalI
glbΩ-total : {I : Obj} {Q : Mor I B} → isTotal (glb Q)
glbΩ-total = ≈-isTotal glb-≈-lub-lbd lubΩ-total
glbΩ : {I : Obj} (Q : Mor I B) → Mapping I B
glbΩ Q = record {mor = glb Q; prf = glb-isUnivalent, glbΩ-total}

lub-⨾wrap : {I : Obj} {Q : Mor I A} → isTotal wrap → lub (Q ⨾ wrap) ≈ Q ˘ ╳ R
lub-⨾wrap {I} {Q} total = ≈-begin
        lub (Q ⨾ wrap)
    ≈⟨ lubΩ≈╳ ⟨≈≈⟩ ╳-cong1 (⨾-cong2 ˘-involution) ⟩
        (R ⨾ wrap ˘ ⨾ Q ˘) ╳ R
    ≈⟨ ╳-cong1 (⨾-assocL ⟨≈≈⟩ ⨾-cong1 (R⨾wrap˘ total) ⟨≈≈⟩ leftId) ⟩
        Q ˘ ╳ R
    □
lub-/R˘ :          {I : Obj} {Q : Mor I A} → isTotal wrap → lub (Q / R ˘) ≈ Q ˘ ╳ R
lub-/R˘ {I} {Q} total = ≈-begin
        lub (Q / R ˘)
    ≈⟨ lubΩ≈╳ ⟨≈≈⟩ ╳-cong1 (⨾-cong2 /˘-˘) ⟩
      (R ⨾ (R \ Q ˘)) ╳ R
    ≈⟨ ╳-cong1 (R⨾-R\ total) ⟩
        Q ˘ ╳ R
    □
```

5.4.8 Power Transpose Λ

Again, if we momentarily think of membership relations, then we find

$$x\ (\Lambda_0\ Q)\ y \Leftrightarrow (\forall\ z \bullet x\ Q\ z \equiv z \in y) \Leftrightarrow y = \{z \mid x\ Q\ z\},$$

i.e., the set of Q-successors of x.

```
Λ0 : {I : Obj} → Mor I A → Mor I B
Λ0 Q = Q ˘ ╳ R
Λ-cong : {I : Obj} {Q S : Mor I A} → Q ≈ S → Λ0 Q ≈ Λ0 S
Λ-cong Q≈S = ╳-cong1 (˘-cong Q≈S)
```

This is indeed a power transpose, as it satisfies the characterization:

$$\forall\ \{Q\ f\} \to \mathsf{isMapping}\ f \to f_0 ⨾ R\ ˘ \approx Q \equiv f_0 \approx \Lambda_0\ Q$$

Indeed,

```
Λ⇒∈ : {I : Obj} {Q : Mor I A} {f : Mapping I B}
  → Mapping.mor f ≈ Λ0 Q → Mapping.mor f ⨾ R ˘ ≈ Q
Λ⇒∈ {I} {Q} {f} f≈ΛQ = ≈-begin
      Mapping.mor f ⨾ R ˘
    ≈⟨ ⨾-cong1 f≈ΛQ ⟩
      (Q ˘ ╳ R) ⨾ R ˘
    ≈⟨ ╳-total-cancel-right comprehensive ⟨≈≈⟩ ˘˘ ⟩
      Q
    □
∈⇒Λ : {I : Obj} {Q : Mor I A} {f : Mapping I B}
  → Mapping.mor f ⨾ R ˘ ≈ Q → Mapping.mor f ≈ Λ0 Q
∈⇒Λ {I} {Q} {f} f⨾R˘≈Q = ≈-sym (≈-begin
    Q ˘ ╳ R
  ≈⟨ ╳-cong1 (˘-cong f⨾R˘≈Q ⟨≈˘≈⟩ ˘-involutionRightConv) ⟩
    (R ⨾ f0 ˘) ╳ R
  ≈˘⟨ lubΩ≈╳ ⟩
    lub f0
  ≈⟨ lub-mapping (Mapping.prf f) ⟩
    f0
  □) where f0 = Mapping.mor f
```

5.5 Categoric.OCC.DirectPower

For the time being, we just follow the exposition in (Furusawa and Kahl, 1998, Sect. 9).

```
module Categoric.OCC.DirectPower {i j k1 k2} {Obj : Set i} (occ : OCC j k1 k2 Obj)
  (let open OCC occ)
  (leftResOp   : LeftResOp orderedSemigroupoid)
  (rightResOp : RightResOp orderedSemigroupoid)
```

```
  (syqOp        : SyqOp osgc)
  where
  open SyqOp                                                syqOp
  open OCC-SyQ-Props        occ                             syqOp
  open SyQ-ResidualProps    osgc          leftResOp rightResOp syqOp
  open ResidualOps                        leftResOp rightResOp
  open OrdCat-Residual-Props orderedCategory leftResOp rightResOp
  open OSGC-Residuals       osgc          leftResOp rightResOp
  open import Categoric.OCC.Order occ leftResOp rightResOp syqOp
  open PreorderWithResiduals occ leftResOp rightResOp using (\-isPreorder; module \-Preorder)
  open import Categoric.OSGC.PowerOp osgc

  record IsMembership {X ℙX : Obj} (∈ : Mor X ℙX) : Set (i ⊍ j ⊍ k₂) where
    field
        ∈-extensional : ∈ ╳ ∈ ⊑ Id
        ∈-comprehensiveI : {A : Obj} {Q : Mor X A} → isTotalI (Q ╳ ∈)
    ∈-comprehensive : {A : Obj} {Q : Mor X A} → isTotal (Q ╳ ∈)
    ∈-comprehensive = isTotal-from-I ∈-comprehensiveI
    Ω : Mor ℙX ℙX
    Ω = ∈ \ ∈
    open \-Preorder″ ∈ public using () renaming
      (╳-\preorder to Ω╳Ω⊑∈╳∈
      ; wrap to 𝒮₀   -- ≈ Id ╳ ∈ : Mor X ℙX − singleton
      ; wrap-injective to 𝒮-injective   -- : isInjectiveI 𝒮₀
      )
    open \-Preorder ∈ using (lbd-ubd-≈-twist) renaming
      (ubdE≈\ to ubdΩ≈\      -- : {I : Obj} {Q : Mor I ℙX} → ubd Q ≈ (∈ ⨾ Q ˘) \ ∈
      ; ubd˘E≈/ to ubd˘Ω≈/   -- : {I : Obj} {Q : Mor I ℙX} → ubd Q ˘ ≈ ∈ ˘ / (Q ⨾ ∈ ˘)
      )
    Ω-isOrder : IsOrder Ω
    Ω-isOrder = fromPreorder (\-isPreorder ∈) (⊑-begin
          Ω ╳ Ω
        ⊑⟨ Ω╳Ω⊑∈╳∈ ⟩
          ∈ ╳ ∈
        ⊑⟨ ∈-extensional ⟩
          Id
        □)
    open IsOrder Ω-isOrder public renaming
        (refl to Ω-refl; trans to Ω-trans; idempot to Ω-idempot
        ; ˘-refl to Ω˘-refl; ˘-trans to Ω˘-trans; ˘-isOrder to Ω˘-isOrder
        ; antisym≈ to Ω╳Ω; ˘-antisym≈ to Ω˘╳Ω˘
        ; order-\ to Ω\Ω≈Ω
        ; isPreorder to Ω-isPreorder; ˘-isPreorder to Ω˘-isPreorder
        ; isPreorder₀ to Ω-isPreorder₀; ˘-isPreorder₀ to Ω˘-isPreorder₀
        )
    open \-OrderWithComprehension Ω-isOrder ∈-comprehensiveI public renaming
        (R⨾-R╳ to ∈⨾-∈╳   -- {Y : Obj} {R : Mor X Y} → ∈ ⨾ (∈ ╳ R) ≈ R
        )
        hiding (Ω; lub-⨾wrap; lub-/R˘)
    ╳∈-univalent : {A : Obj} {Q : Mor X A} → isUnivalent (Q ╳ ∈)
    ╳∈-univalent {A} {Q} = isUnivalent-from-I (⊑-begin
          (Q ╳ ∈) ˘ ⨾ (Q ╳ ∈)
        ≈⟨ ⨾-cong₁ ╳-˘ ⟩
          (∈ ╳ Q) ⨾ (Q ╳ ∈)
        ⊑⟨ ╳-cancel-middle ⟩
          ∈ ╳ ∈
        ⊑⟨ ∈-extensional ⟩
          Id
        □)

    open \-Preorder″ ∈ using (wrap⨾R˘; R⨾wrap˘; wrap⨾E; R⨾-R\)
    𝒮 : Mapping X ℙX
    𝒮 = record {mor = 𝒮₀; prf = ╳∈-univalent, ∈-comprehensive}
    𝒮⨾∈˘ : 𝒮₀ ⨾ ∈ ˘ ≈ Id
    𝒮⨾∈˘ = wrap⨾R˘ (mappingTotal 𝒮)
    ∈⨾𝒮˘ : ∈ ⨾ 𝒮₀ ˘ ≈ Id
    ∈⨾𝒮˘ = R⨾wrap˘ (mappingTotal 𝒮)
    𝒮⨾Ω : 𝒮₀ ⨾ Ω ≈ ∈
    𝒮⨾Ω = wrap⨾E (Mapping.prf 𝒮)
    ∈⨾-∈\ : {Y : Obj} {R : Mor X Y} → ∈ ⨾ (∈ \ R) ≈ R
    ∈⨾-∈\ = R⨾-R\ (mappingTotal 𝒮)
    open \-OrderWithComprehension Ω-isOrder ∈-comprehensiveI using
      (lub-⨾wrap; lub-/R˘)
    lub-⨾𝒮 : {I : Obj} {R : Mor I X} → lub (R ⨾ 𝒮₀) ≈ R ˘ ╳ ∈
    lub-⨾𝒮 {I} {R} = lub-⨾wrap (mappingTotal 𝒮)
    lub-/∈˘ : {I : Obj} {R : Mor I X} → lub (R / ∈ ˘) ≈ R ˘ ╳ ∈
    lub-/∈˘ = lub-/R˘ (mappingTotal 𝒮)

    Λ-isMapping : {I : Obj} {R : Mor I X} → isMapping (Λ₀ R)
    Λ-isMapping = ╳∈-univalent, ∈-comprehensive
    Λ^-isMapping : {I : Obj} {Q : Mor X I} → isMapping (Q ╳ ∈)
    Λ^-isMapping = ╳∈-univalent, ∈-comprehensive
    Λ : {I : Obj} → Mor I X → Mapping I ℙX
    Λ    R = record {mor = Λ₀ R; prf = Λ-isMapping}
    isPower : IsPower ∈
    isPower = record
        {Λ = Λ
        ; isPowerTranspose = λ I → record
          {Λ⇒∈ = λ {R} {f} → Λ⇒∈ {I} {R} {f}
          ; ∈⇒Λ = λ {R} {f} → ∈⇒Λ {I} {R} {f}
          }
        }

    Λ⨾Ω˘ : {I : Obj} {R : Mor I X} → Λ₀ R ⨾ Ω ˘ ≈ R / ∈ ˘
    Λ⨾Ω˘ {I} {R} = ≈-begin
          (R ˘ ╳ ∈) ⨾ Ω ˘
      ≈⟨ ⨾-cong₂ Ω˘ ⟩
          (R ˘ ╳ ∈) ⨾ (∈ ˘ / ∈ ˘)
      ≈⟨ /-outer-⨾-≈ Λ-isMapping ⟩
          ((R ˘ ╳ ∈) ⨾ ∈ ˘) / ∈ ˘
      ≈⟨ /-cong₁ (╳-total-cancel-right (proj₂ Λ-isMapping) ⟨≈≈⟩ ˘˘) ⟩
          R / ∈ ˘
      □

  record DirectPower : Set (i ⊍ j ⊍ k₂) where
    field
        ℙ : Obj → Obj
        ∈ : {X : Obj} → Mor X (ℙ X)
        isMembership : {X : Obj} → IsMembership {X} (∈ {X})
    module _ {X : Obj} where open IsMembership (isMembership {X}) public
    powerOp : PowerOp
    powerOp = record {ℙ = ℙ; ∈ = ∈; isPower = λ {X} → isPower {X}}
```

6 Internal Galois Connections

6.1 Categoric.OSGC.Preorder.Closure

It is a well-known fact that a so-called ‘closure-operator’ can be characterized as a monotone, increasing, and idempotent function, or equivalently a function C with

$$\forall\, x, y \bullet x \preccurlyeq C\,(x) \equiv C\,(x) \preccurlyeq C\,(y)$$

— the so-called ‘first closure lemma’. It is more concise and so chosen as the characterizing definition, with the alternative being derived results.

We begin with *pre*-closure operators, i.e. those in the setting of preorders and *OSGC*s. Consequently, many results appear in the form of indirect equality, i.e. with an extra order appended here and there. Such extras disappear in the setting of partial orders, where the law of indirect equality coincides with mere equality (without the order).

```
record PreClosureOp {A : Obj} {E : Mor A A}
    (A-isPreorder : IsPreorder osgc E) (CC : Mapping A A) : Set k₁ where
  private
    module A = IsPreorder osgc A-isPreorder
    module C = Mapping CC
  open C using () renaming (mor to C)
  field char : E ⨾ C ˘ ≈ C ⨾ E ⨾ C ˘
  char˘ : C ⨾ E ˘ ≈ C ⨾ E ˘ ⨾ C ˘
  char˘ = ≈-begin
      C ⨾ E ˘
    ≈˘⟨ ˘-involutionRightConv ⟩
      (E ⨾ C ˘) ˘
    ≈⟨ ˘-cong char ⟩
      (C ⨾ E ⨾ C ˘) ˘
    ≈⟨ ˘-involution ⟨≈≈⟩ (⨾-cong₁ ˘-involutionRightConv ⟨≈≈⟩ ⨾-assoc) ⟩
      C ⨾ E ˘ ⨾ C ˘
    □
```

We use the name char as an abbreviation of characterisation.

6.1.1 Increasing

Before showing that the closure operator is increasing, let us observe that both sides of the characterization are super identities:

```
  CEC˘-supId : isSuperidentity (C ⨾ E ⨾ C ˘)
  CEC˘-supId = (λ {B} {R} → ⊑-begin
      R
    ⊑⟨ proj₁ C.total ⟩
      (C ⨾ C ˘) ⨾ R
    ⊑⟨ ⨾-monotone₁₂ A.leftSupId ⟩
      (C ⨾ E ⨾ C ˘) ⨾ R
    □), (λ {B} {S} → ⊑-begin
      S
    ⊑⟨ proj₂ C.total ⟩
      S ⨾ C ⨾ C ˘
    ⊑⟨ ⨾-monotone₂₂ A.leftSupId ⟩
      S ⨾ C ⨾ E ⨾ C ˘
    □)
  EC˘-supId : isSuperidentity (E ⨾ C ˘)
  EC˘-supId = (λ {B} {R} → ⊑-begin
      R
    ⊑⟨ proj₁ CEC˘-supId ⟩
      (C ⨾ E ⨾ C ˘) ⨾ R
    ≈˘⟨ ⨾-cong₁ char ⟩
      (E ⨾ C ˘) ⨾ R
    □), (λ {B} {S} → ⊑-begin
      S
    ⊑⟨ proj₂ CEC˘-supId ⟩
      S ⨾ (C ⨾ E ⨾ C ˘)
    ≈˘⟨ ⨾-cong₂ char ⟩
      S ⨾ E ⨾ C ˘
    □)
   -- Pointwise: ∀ x • x ≼ C (x)
  increasing : C ⊑ E
  increasing = ⊑-begin
      C
    ⊑⟨ proj₁ EC˘-supId ⟨⊑≈⟩ ⨾-assoc ⟩
      E ⨾ C ˘ ⨾ C
    ⊑⟨ proj₂ C.unival ⟩
      E
    □
   -- Pointwise: ∀ x, y • x ≼ y ⇒ x ≼ C (y)
  expansion : E ⊑ E ⨾ C ˘
  expansion = ⊑-begin
      E
    ⊑⟨ proj₂ EC˘-supId ⟩
      E ⨾ E ⨾ C ˘
    ≈⟨ ⨾-assoc ⟨≈˘≈⟩ ⨾-cong₁ A.idempot ⟩
      E ⨾ C ˘
    □
```

Consequently, we have the combinators,

```
  EC-⊑-E : E ⨾ C ⊑ E
  EC-⊑-E = ⨾-monotone₂ increasing ⟨⊑≈⟩ A.idempot
  CE-⊑-E : C ⨾ E ⊑ E
  CE-⊑-E = ⨾-monotone₁ increasing ⟨⊑≈⟩ A.idempot
```

6.1.2 Quasi-idempodentcy

Without the presence of antisymmetry, we have only been able to approximate idempotence as follows:

```
  EC˘C-⊑-CE : E ⨾ C ˘ ⨾ C ⊑ C ⨾ E
  EC˘C-⊑-CE = ⨾-assoc ⟨≈˘⊑⟩ swap-⊑-⨾-unival˘ C.unival (⊑-reflexive (char ⟨≈≈˘⟩ ⨾-assoc))
  idempE : C ⨾ C ⨾ E ≈ C ⨾ E
  idempE = ⊑-antisym (⊑-begin
```

```
        C ⨾ C ⨾ E
      ⊑⟨ ⨾-monotone₂ (proj₁ EC˘-supId) ⟩
        C ⨾ (E ⨾ C ˘) ⨾ C ⨾ E
      ≈⟨ (⨾-assoc ⟨≈˘≈˘⟩ (⨾-cong₁ char)) ⟨≈≈⟩ ⨾-assoc ⟩
        E ⨾ C ˘ ⨾ C ⨾ E
      ⊑⟨ ⨾-assoc₃₊₁ ⟨≈˘⊑⟩ (⨾-monotone₁ EC˘C-⊑-CE ⟨⊑≈⟩ (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ A.idempot)) ⟩
        C ⨾ E
      □) (⊑-begin
        C ⨾ E
      ⊑⟨ proj₁ CEC˘-supId ⟩
        (C ⨾ E ⨾ C ˘) ⨾ C ⨾ E
      ⊑⟨ (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ (⨾-assoc ⟨≈≈˘⟩ ⨾-assoc₃₊₁)) ⟨≈⊑⟩ ⨾-monotone₂₁ EC˘C-⊑-CE ⟩
        C ⨾ (C ⨾ E) ⨾ E
      ≈⟨ ⨾-cong₂ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂₂ A.idempot ⟩
        C ⨾ C ⨾ E
      □)
```

6.1.3 Monotonicity

Finally, we show that the characterization yields monotonicity. Recall that monotocity can take a multiplicity of forms:

```
  C monotonic
≡ ∀ x, y • x ≼ y ⇒ C (x) ≼ C (y)
≡ ≼ ⊑ C ⨾ ≼ ⨾ C ˘
≡ ≼ ⨾ C ⊑ C ⨾ ≼
```

This final form, the so called ‘L-simulation’, will be our monotonicity. In the converse order, we name this ‘comonotonicity’ — needless to say, the notions coincide since C is a mapping.

```
  comonotone˘ : C ˘ ⨾ E ⊑ E ⨾ C ˘
  comonotone˘ = ⊑-begin
        C ˘ ⨾ E
      ⊑⟨ ⨾-monotone₂ (proj₂ EC˘-supId) ⟩
        C ˘ ⨾ E ⨾ E ⨾ C ˘
      ≈⟨ ⨾-cong₂ (⨾-assoc ⟨≈˘≈⟩ ⨾-cong₁ A.idempot) ⟩
        C ˘ ⨾ E ⨾ C ˘
      ≈⟨ ⨾-cong₂ char ⟩
        C ˘ ⨾ C ⨾ E ⨾ C ˘
      ≈⟨ ⨾-cong₂ (⨾-assoc ⟨≈˘≈˘⟩ ⨾-cong₁ idempE) ⟩
        C ˘ ⨾ (C ⨾ C ⨾ E) ⨾ C ˘
      ≈⟨ ⨾-cong₂ ⨾-assoc₃₊₁ ⟨≈≈⟩ ⨾-assocL ⟩
        (C ˘ ⨾ C) ⨾ C ⨾ E ⨾ C ˘
      ⊑⟨ proj₁ C.unival ⟩
        C ⨾ E ⨾ C ˘
      ≈˘⟨ char ⟩
        E ⨾ C ˘
      □
  comonotone : E ˘ ⨾ C ⊑ C ⨾ E ˘
  comonotone = ⊑-begin
        E ˘ ⨾ C
    ≈˘⟨ ˘-involutionLeftConv ⟩
        (C ˘ ⨾ E) ˘
    ⊑⟨ ˘-monotone comonotone˘ ⟩
        (E ⨾ C ˘) ˘
    ≈⟨ ˘-involutionRightConv ⟩
        C ⨾ E ˘
      □

  monotone : E ⨾ C ⊑ C ⨾ E
  monotone = ⊑-begin
        E ⨾ C
      ⊑⟨ proj₁ C.total ⟨⊑≈⟩ (⨾-assoc ⟨≈≈˘⟩ ⨾-cong₂ ⨾-assoc) ⟩
        C ⨾ (C ˘ ⨾ E) ⨾ C
      ⊑⟨ ⨾-monotone₂₁ comonotone˘ ⟨⊑≈⟩ ⨾-cong₂ ⨾-assoc ⟩
        C ⨾ E ⨾ C ˘ ⨾ C
      ⊑⟨ ⨾-monotone₂ (proj₂ C.unival) ⟩
        C ⨾ E
      □

  monotone˘ : C ˘ ⨾ E ˘ ⊑ E ˘ ⨾ C ˘
  monotone˘ = ⊑-begin
        C ˘ ⨾ E ˘
      ≈˘⟨ ˘-involution ⟩
        (E ⨾ C) ˘
      ⊑⟨ ˘-monotone monotone ⟩
        (C ⨾ E) ˘
      ≈⟨ ˘-involution ⟩
        E ˘ ⨾ C ˘
      □
```

Furthermore, we have a peculiar result:

```
  CE-⊑-EC˘ : C ⨾ E ⊑ E ⨾ C ˘
  CE-⊑-EC˘ = ⊑-begin
        C ⨾ E
      ⊑⟨ ⨾-monotone₂ expansion ⟩
        C ⨾ E ⨾ C ˘
      ≈˘⟨ char ⟩
        E ⨾ C ˘
      □
```

Peculiar since it is one symbol short of expressing monotoncity of C ˘, which is generally not true!

6.1.4 Piecewise Closure Characterization

Of-course proving char directly may be a challenge in itself, luckily there is a piecewise formulation: a closure operator is precisely an increasing, idempotent, and monotonic function.

```
module _ {A : Obj} {E : Mor A A} (A-isPreorder : IsPreorder osgc E) (CC : Mapping A A)
  where
  private
      module A = IsPreorder osgc A-isPreorder
      module C = Mapping CC
  open C using () renaming (mor to C)
  piecewise-to-closure : (increasing : C ⊑ E) (idemp : C ⨾ C ≈ C) (monotone : E ⨾ C ⊑ C ⨾ E)
                                    → PreClosureOp {A} {E} A-isPreorder CC
  piecewise-to-closure increasing idemp monotone = record {char = ⊑-antisym (⊑-begin
        E ⨾ C ˘
      ⊑⟨ ⨾-monotone₂ (proj₁ C.total ⟨⊑≈⟩ ⨾-assoc) ⟩
        E ⨾ C ⨾ C ˘ ⨾ C ˘
      ≈⟨ ⨾-cong₂₂ (˘-involution ⟨≈˘≈⟩ ˘-cong idemp) ⟩
        E ⨾ C ⨾ C ˘
      ⊑⟨ ⨾-assoc ⟨≈˘⊑⟩ (⨾-monotone₁ monotone ⟨⊑≈⟩ ⨾-assoc) ⟩
        C ⨾ E ⨾ C ˘
```

```
□)(⊑-begin
    C ⨾ E ⨾ C ˘
  ⊑⟨ ⨾-monotone₁ increasing ⟩
    E ⨾ E ⨾ C ˘
  ⊑⟨ ⨾-assoc ⟨≈˘⊑⟩ ⨾-monotone₁ A.trans ⟩
    E ⨾ C ˘
  □)}
```

6.1.5 Dually: Interior Operator

Now we can dualize to obtain the notion of an interior, or co-closure operator. Given C ⨾ E ≈ C ⨾ E ⨾ C ˘, we show that C is a closure on the reverse order, i.e., a co-closure.

```
record PreCoclosureOp {A : Obj} {E : Mor A A}
    (A-isPreorder : IsPreorder osgc E) (CC : Mapping A A) : Set k₁ where
  private
    module A = IsPreorder osgc A-isPreorder
    module C = Mapping CC
  open C using () renaming (mor to C)
  field char : C ⨾ E ≈ C ⨾ E ⨾ C ˘
  private
    COC : PreClosureOp {A} {E ˘} A.˘-isPreorder₀ CC
    COC = record {char = ≈-begin
      E ˘ ⨾ C ˘
      ≈˘⟨ ˘-involution ⟩
        (C ⨾ E) ˘
      ≈⟨ ˘-cong char ⟩
        (C ⨾ E ⨾ C ˘) ˘
      ≈⟨ ˘-involution ⟨≈≈⟩ (⨾-cong₁ ˘-involutionRightConv ⟨≈≈⟩ ⨾-assoc) ⟩
        C ⨾ E ˘ ⨾ C ˘
      □}
  open PreClosureOp COC hiding (monotone; comonotone˘)
  contraction : E ⊑ C ⨾ E
  contraction = ⊑-˘-swap expansion ⟨⊑≈⟩ ˘-coinvolution
  monotone : E ⨾ C ⊑ C ⨾ E
  monotone = ⨾-cong₁ ˘˘ ⟨≈˘⊑⟩ (comonotone ⟨⊑≈⟩ ⨾-cong₂ ˘˘)
  comonotone˘ : C ˘ ⨾ E ⊑ E ⨾ C ˘
  comonotone˘ = ⨾-cong₂ ˘˘ ⟨≈˘⊑⟩ (monotone˘ ⟨⊑≈⟩ ⨾-cong₁ ˘˘)
  open PreClosureOp COC public hiding (char˘; expansion; comonotone; monotone˘)
    renaming
    (char       to char˘               -- : E ˘ ⨾ C ˘ ≈ C ⨾ E ˘ ⨾ C ˘
    ; CEC˘-supId to CE˘C˘-supId        -- : isSuperidentity (C ⨾ E ˘ ⨾ C ˘)
    ; EC˘-supId  to E˘C˘-supId         -- : isSuperidentity (E ˘ ⨾ C ˘)
    ; increasing to decreasing         -- : C ⊑ E ˘
    ; EC-⊑-E     to E˘C-⊑-E˘           -- : E ˘ ⨾ C ⊑ E ˘
    ; CE-⊑-E     to CE˘-⊑-E˘           -- : C ⨾ E ˘ ⊑ E ˘
    ; EC˘C-⊑-CE  to E˘C˘C-⊑-CE˘        -- : E ˘ ⨾ C ˘ ⨾ C ⊑ C ⨾ E ˘
    ; idempE     to idempE˘            -- : C ⨾ C ⨾ E ˘ ≈ C ⨾ E ˘
    ; CE-⊑-EC˘   to CE˘-⊑-E˘C˘         -- : C ⨾ E ˘ ⊑ E ˘ ⨾ C ˘
    ; comonotone˘ to monotone˘         -- : C ˘ ⨾ E ˘ ⊑ E ˘ ⨾ C ˘
    ; monotone   to comonotone         -- : E ˘ ⨾ C ⊑ C ⨾ E ˘
    )
```

Dually, interior operators have an equivalent piecewise formulation.

```
module _ {A : Obj} {E : Mor A A} (A-isPreorder : IsPreorder osgc E) (CC : Mapping A A)
  where
  private
    module A = IsPreorder osgc A-isPreorder
    module C = Mapping CC
  open C using () renaming (mor to C)
  piecewise-to-interior : (decreasing : C ⊑ E ˘) (idemp : C ⨾ C ≈ C) (monotone : E ⨾ C ⊑ C ⨾ E)
                          → PreCoclosureOp {A} {E} A-isPreorder CC
  piecewise-to-interior decreasing idemp monotone = record {char = ⊑-antisym (⊑-begin
      C ⨾ E
    ⊑⟨ ⨾-monotone₂ (proj₂ C.total) ⟩
      C ⨾ E ⨾ C ⨾ C ˘
    ⊑⟨ ⨾-monotone₂ (⨾-assoc ⟨≈˘⊑⟩ (⨾-monotone₁ monotone ⟨⊑≈⟩ ⨾-assoc)) ⟩
      C ⨾ C ⨾ E ⨾ C ˘
    ≈⟨ ⨾-assoc₄ ⟨≈˘≈⟩ (⨾-cong₁₁ idemp ⟨≈≈⟩ ⨾-assoc) ⟩
      C ⨾ E ⨾ C ˘
    □)(⊑-begin
      C ⨾ E ⨾ C ˘
    ⊑⟨ ⨾-monotone₂₂ (˘-monotone decreasing ⟨⊑≈⟩ ˘˘) ⟩
      C ⨾ E ⨾ E
    ⊑⟨ ⨾-monotone₂ A.trans ⟩
      C ⨾ E
    □)}
```

It is to be noted that we could have requested a weaker hypothesis, idemp : C ⨾ C ⨾ E ≈ C ⨾ E, and still proved that C is an interior operation. We have chosen not to do so, for the sake of symmetry with the definition of piecewise-to-closure.

6.1.6 Conclusion

The reason that closure operators make an appearance is due to their close relation with Galois Connections. In fact, closures are to Galois Connections as monads are to adjunctions; additionally, the former are instances of the latter. Moreover, the notion of (co)closures arises frequently in optimization problems and in limit constructions; e.g. "the smallest ..." or "the largest ..." problem statements can usually be stated as (co)closure results.

6.2 Categoric.OCC.Order.Closure

With antisymmetry in hand, we can now obtain more complete results; such as true idempotency.

```
record ClosureOp {A : Obj} {E : Mor A A}
  (A-isOrder : IsOrder E) (CC : Mapping A A) : Set k₁ where
  open IsOrder A-isOrder hiding (idempot)
  private
      module A = IsOrder A-isOrder
      module C = Mapping CC
  open C using () renaming (mor to C)
  field char : E ⨾ C ˘ ≈ C ⨾ E ⨾ C ˘
  open PreClosureOp {A} {E} {A.isPreorder₀} {CC} (record {char = char}) public hiding (char)
```

6.2.1 Idempotence and Range Closure

```
idempot : C ⨾ C ≈ C
idempot = indirect-≈ (⨾-isMapping C.prf C.prf) C.prf (⨾-assoc ⟨≈≈⟩ idempE)
```

In turn, with this, we can now give a useful characterization of closed ‘elements’:

```
ranClosed-← : {B : Obj} {R : Mor B A} →   R ⨾ C ≈ R → R ⊑ R ⨾ C ˘ ⨾ C
ranClosed-← = mapRanClosed-← C.prf idempot
ranClosed-→ :  {B : Obj} {R : Mor B A} → R ⊑ R ⨾ C ˘ ⨾ C → R ⨾ C ≈ R
ranClosed-→ = mapRanClosed-→ C.prf idempot
```

6.2.2 GLB Closure

In fact we also have closure results for glb and C:

```
glb-closed-⊑ : {I : Obj} {R : Mor I A} → R ⨾ C ≈ R → glb R ⨾ C ⊑ glb R
glb-closed-⊑ {I} {R} R⨾C≈R = ˘⊓-universal
        (⊑-begin
           lbd R ˘ ⨾ (lbd R ˘ ⊓ E) ⨾ C
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone1 ⊓-cancel-left ⟩
           E ⨾ C
        ⊑⟨ ⨾-monotone2 increasing ⟨⊑⊑⟩ trans ⟩
              E
        □)
        (⨾-assoc ⟨≈⊑⟩ \-universal ((⊑-begin
              R ˘ ⨾ (lbd R ˘ ⊓ E) ⨾ (C ⨾ E ˘)
        ⊑⟨ ⨾-cong1 (˘-cong R⨾C≈R ⟨≈˘≈⟩ ˘-involution) ⟨≈⊑⟩ ⨾-monotone21 ˘⊓-⊑-/ ⟩
           (C ˘ ⨾ R ˘) ⨾ ((R ˘ \ E ˘) / E ˘) ⨾ (C ⨾ E ˘)
        ⊑⟨ ⨾-22assoc121 ⟨≈⊑⟩ ⨾-monotone21 /-outer-⨾ ⟩
              C ˘ ⨾ ((R ˘ ⨾ (R ˘ \ E ˘)) / E ˘) ⨾ (C ⨾ E ˘)
        ⊑⟨ ⨾-monotone21 (/-monotone \-cancel-outer ⟨⊑≈⟩ order˘-/) ⟩
              C ˘ ⨾ E ˘ ⨾ (C ⨾ E ˘)
        ⊑⟨ ⨾-monotone1&21 monotone˘ ⟩
              E ˘ ⨾ C ˘ ⨾ (C ⨾ E ˘)
        ⊑⟨ ⨾-monotone2 (⨾-assocL ⟨≈⊑⟩ proj1 C.unival) ⟩
              E ˘ ⨾ E ˘
        ⊑⟨ ˘-trans ⟩
              E ˘
        □)))
glb-closed : {I : Obj} {R : Mor I A} → isTotal (glb R) → R ⨾ C ≈ R → glb R ⨾ C ≈ glb R
glb-closed glbR-total R⨾C≈R =
   total⊑unival-≈       (⨾-isTotal glbR-total C.total) glb-isUnivalent (glb-closed-⊑ R⨾C≈R)
glb-closed′    : {I : Obj} {R : Mor I A} → isTotal (glb R) → R ⨾ C ≈ R → glb R ⊑ glb R ⨾ C ˘ ⨾ C
glb-closed′ glbR-total R⨾C≈R = mapRanClosed-← C.prf idempot (glb-closed glbR-total R⨾C≈R)
```

6.2.3 Dualitiy and LUB Closure

Now we can dualize,

```
record CoclosureOp     {A : Obj} {E : Mor A A}
  (A-isOrder : IsOrder E) (CC : Mapping A A) : Set k1 where
  open IsOrder A-isOrder hiding (idempot)
  private
        module A = IsOrder A-isOrder
        module C = Mapping CC
  open C using () renaming (mor to C)
  field char : C ⨾ E ≈ C ⨾ E ⨾ C ˘
  open PreCoclosureOp {A} {E} {A.isPreorder0} {CC} (record {char = char}) public hiding (char)
  open ClosureOp {A} {E ˘} {˘-isOrder} {CC} (record {char = char˘}) public using
     (idempot         -- : C ⨾ C ≈ C
     ; ranClosed-←    -- : ∀ {R} → R ⨾ C ≈ R → R ⊑ R ⨾ C ˘ ⨾ C
     ; ranClosed-→    -- : ∀ {R} → R ⊑ R ⨾ C ˘ ⨾ C → R ⨾ C ≈ R
     )
  open ClosureOp {A} {E ˘} {˘-isOrder} {CC} (record {char = char˘}) using
     (glb-closed-⊑; glb-closed; glb-closed′)
  lub-closed-⊑ : {I : Obj} {R : Mor I A} → R ⨾ C ≈ R → lub R ⨾ C ⊑ lub R
  lub-closed-⊑ x = ⨾-cong1 (⊓-cong1 (˘-cong (\-cong2 ˘˘)))
     ⟨≈˘⊑⟩ glb-closed-⊑ x ⟨⊑≈⟩ ⊓-cong1 (˘-cong (\-cong2 ˘˘))
  lub-closed : {I : Obj} {R : Mor I A} → isTotal (lub R) → R ⨾ C ≈ R → lub R ⨾ C ≈ lub R
  lub-closed {I} {R} x y = ⨾-cong1 (⊓-cong1 (˘-cong (\-cong2 ˘˘)))
     ⟨≈˘≈⟩ glb-closed x′ y ⟨≈≈⟩ ⊓-cong1 (˘-cong (\-cong2 ˘˘))
     where x′ : isTotal ((R ˘ \ (E ˘) ˘) ˘ ⊓ E ˘)
                x′ = isSuperidentity-≈ (⨾-cong (⊓-cong1 (˘-cong (\-cong2 (≈-sym ˘˘))))
                   (˘-cong (⊓-cong1 (˘-cong (\-cong2 (≈-sym ˘˘)))))) x
  lub-closed′ : {I : Obj} {R : Mor I A} →         isTotal (lub R) → R ⨾ C ≈ R → lub R ⊑ lub R ⨾ C ˘ ⨾ C
  lub-closed′ {I} {R} x y = ⊓-cong1 (˘-cong (\-cong2 ˘˘))
     ⟨≈˘⊑⟩ glb-closed′ x′ y ⟨⊑≈⟩ ⨾-cong1 (⊓-cong1 (˘-cong (\-cong2 ˘˘)))
     where x′ : isTotal ((R ˘ \ (E ˘) ˘) ˘ ⊓ E ˘)
                x′ = isSuperidentity-≈ (⨾-cong (⊓-cong1 (˘-cong (\-cong2 (≈-sym ˘˘))))
                   (˘-cong (⊓-cong1 (˘-cong (\-cong2 (≈-sym ˘˘)))))) x
```

6.3 Categoric.OSGC.Preorder.Galois

We are now in a position to turn to internal (monotone) Galois Connections. The characterization that (L, U) constitute such a connection is precisely $\forall\ x, y \bullet L\ (x) \preccurlyeq' y \equiv x \preccurlyeq U\ (y)$, i.e. $L ⨾ \preccurlyeq' \approx \preccurlyeq ⨾ U\ \breve{}$. Formally,

```
record PreGaloisConnection {A1 A2 : Obj} {E1 : Mor A1 A1} {E2 : Mor A2 A2}
  (A1-isPreorder : IsPreorder osgc E1) (A2-isPreorder : IsPreorder osgc E2)
  (LL : Mapping A1 A2) (UU : Mapping A2 A1) : Set k1 where
    private
        module A1 = IsPreorder osgc A1-isPreorder
        module A2 = IsPreorder osgc A2-isPreorder
        module L = Mapping LL
        module U = Mapping UU
    open L using () renaming (mor to L)
    open U using () renaming (mor to U)
    field gc : L ⨾ E2 ≈ E1 ⨾ U ˘
```

6.3.1 Co-connection

The notion of being a connection is a somewhat symmetric property. That is, (L, U) are connected precisely when (U, L) are ‘co-connected.’

```
    gc˘ : U ⨾ E1 ˘ ≈ E2 ˘ ⨾ L ˘
    gc˘ = ≈-sym (˘-involution ⟨≈˘≈⟩ (˘-cong gc ⟨≈≈⟩ ˘-involutionRightConv))
```

6.3.2 Cancellation Laws

We have some immediate ‘cancellation’ properties; though without identities, they do not appear as simple as they inherently are.

The cancellation ∀ x, y • U (L (x)) ≼ x and its variants are given:

```
L-⊑-E⨾U˘ : L ⊑ E1 ⨾ U ˘
L-⊑-E⨾U˘ = A2.rightSupId ⟨⊑≈⟩ gc

LU-⊑-E : L ⨾ U ⊑ E1
LU-⊑-E = swap-⊑-⨾-unival˘ U.unival L-⊑-E⨾U˘

U˘⨾L˘-⊑-E˘ : U ˘ ⨾ L ˘ ⊑ E1 ˘
U˘⨾L˘-⊑-E˘ = ˘-involution ⟨≈˘⊑⟩ (˘-⊑-swap (LU-⊑-E ⟨⊑≈˘⟩ ˘˘))

EU˘L˘-supId : isSuperidentity (E1 ⨾ U ˘ ⨾ L ˘)
EU˘L˘-supId = (λ {B} {R} → ⊑-begin
      R
   ⊑⟨ proj1 L.total ⟩
      (L ⨾ L ˘) ⨾ R
   ⊑⟨ ⨾-monotone11 A2.rightSupId ⟩
      ((L ⨾ E2) ⨾ L ˘) ⨾ R
   ≈⟨ ⨾-cong1 (⨾-cong1 gc ⟨≈≈⟩ ⨾-assoc) ⟩
      (E1 ⨾ U ˘ ⨾ L ˘) ⨾ R
   □),
   (λ {B} {S} → swap-⨾-⊑-total L.total
      (⨾-monotone ⊑-refl L-⊑-E⨾U˘) ⟨⊑≈⟩ (⨾-assoc ⟨≈≈⟩ ⨾-cong2 ⨾-assoc))

LUE˘-supId : isSuperidentity (L ⨾ U ⨾ E1 ˘)
LUE˘-supId = (λ {B} {R} → ⊑-begin
      R
   ≈˘⟨ ˘˘ ⟩
      R ˘ ˘
   ⊑⟨ ˘-monotone (proj2 EU˘L˘-supId) ⟩
     (R ˘ ⨾ E1 ⨾ U ˘ ⨾ L ˘) ˘
   ≈⟨ ˘-cong (⨾-cong2 (⨾-assoc ⟨≈˘≈˘⟩ ⨾-cong1 ˘-involutionRightConv)) ⟩
     (R ˘ ⨾ (U ⨾ E1 ˘) ˘ ⨾ L ˘) ˘
   ≈˘⟨ ˘-cong (⨾-cong2 ˘-involution) ⟩
     (R ˘ ⨾ (L ⨾ U ⨾ E1 ˘) ˘) ˘
   ≈⟨ ˘-coinvolution ⟩
      (L ⨾ U ⨾ E1 ˘) ⨾ R
   □), (λ {B} {S} → ⊑-begin
      S
   ≈˘⟨ ˘˘ ⟩
      S ˘ ˘
   ⊑⟨ ˘-monotone (proj1 EU˘L˘-supId) ⟩
     ((E1 ⨾ U ˘ ⨾ L ˘) ⨾ S ˘) ˘
   ≈⟨      ˘-cong (⨾-cong1 (⨾-assoc ⟨≈˘≈˘⟩ ⨾-cong1 ˘-involutionRightConv)) ⟩
     (((U ⨾ E1 ˘) ˘ ⨾ L ˘) ⨾ S ˘) ˘
   ≈˘⟨ ˘-cong (⨾-cong1 ˘-involution) ⟩
     ((L ⨾ U ⨾ E1 ˘) ˘ ⨾ S ˘) ˘
   ≈⟨ ˘-coinvolution ⟩
      S ⨾ (L ⨾ U ⨾ E1 ˘)
   □)
```

The cancellation ∀ y • y ≼′ L (U (y)) and its variants are given:

```
U-⊑-E˘⨾L˘ : U ⊑ E2 ˘ ⨾ L ˘
U-⊑-E˘⨾L˘ = A1.˘-rightSupId ⟨⊑≈⟩ gc˘

UL-⊑-E˘ : U ⨾ L ⊑ E2 ˘
UL-⊑-E˘ = swap-⊑-⨾-unival˘ L.unival U-⊑-E˘⨾L˘

L˘U˘-⊑-E : L ˘ ⨾ U ˘ ⊑ E2
L˘U˘-⊑-E = ˘-involution ⟨≈˘⊑⟩ (˘-⊑-swap UL-⊑-E˘)

E˘L˘U˘-supId : isSuperidentity (E2 ˘ ⨾ L ˘ ⨾ U ˘)
E˘L˘U˘-supId = (λ {B} {R} → ⊑-begin
      R
    ⊑⟨ proj1 U.total ⟩
      (U ⨾ U ˘) ⨾ R
   ⊑⟨ ⨾-monotone12 A1.˘-leftSupId ⟩
      (U ⨾ E1 ˘ ⨾ U ˘) ⨾ R
   ≈⟨ ⨾-cong1 (⨾-assoc ⟨≈˘≈⟩ ⨾-cong1 gc˘ ⟨≈≈⟩ ⨾-assoc) ⟩
      (E2 ˘ ⨾ L ˘ ⨾ U ˘) ⨾ R
   □),
  (λ {B} {S} → swap-⨾-⊑-total U.total (⨾-monotone ⊑-refl U-⊑-E˘⨾L˘ ⟨⊑≈˘⟩ ⨾-assoc) ⟨⊑≈⟩ ⨾-assoc4)

ULE-supId : isSuperidentity (U ⨾ L ⨾ E2)
ULE-supId = (λ {B} {R} → ⊑-begin
      R
   ⊑⟨ proj1 U.total ⟩
      (U ⨾ U ˘) ⨾ R
   ⊑⟨ ⨾-monotone12 A1.leftSupId ⟩
      (U ⨾ E1 ⨾ U ˘) ⨾ R
   ≈˘⟨ ⨾-cong12 gc ⟩
      (U ⨾ L ⨾ E2) ⨾ R
   □), (λ {B} {R} → ⊑-begin
      R
   ⊑⟨ proj2 U.total ⟩
      R ⨾ (U ⨾ U ˘)
   ⊑⟨ ⨾-monotone22 A1.leftSupId ⟩
      R ⨾ (U ⨾ E1 ⨾ U ˘)
   ≈˘⟨ ⨾-cong22 gc ⟩
      R ⨾ (U ⨾ L ⨾ E2)
   □)
```

6.3.3 Monotonicity

We present four equivalent formulations of monotonicity, for each adjoint.

```
L-monotone : E1 ⨾ L ⊑ L ⨾ E2
L-monotone = ⊑-begin
        E1 ⨾ L
      ⊑⟨ ⨾-monotone2 (proj1 EU˘L˘-supId) ⟩
        E1 ⨾ (E1 ⨾ U ˘ ⨾ L ˘) ⨾ L
      ≈⟨ ⨾-22assoc121 ⟨≈˘≈⟩ ⨾-cong2 ⨾-assoc ⟨≈≈⟩ ⨾-cong1 A1.idempot ⟩
        E1 ⨾ U ˘ ⨾ L ˘ ⨾ L
      ⊑⟨ ⨾-monotone2 (proj2 L.unival) ⟩
        E1 ⨾ U ˘
      ≈˘⟨ gc ⟩
        L ⨾ E2
      □

L-monotone˘ : L ˘ ⨾ E1 ˘ ⊑ E2 ˘ ⨾ L ˘
L-monotone˘ = ˘˘ ⟨≈˘⊑⟩
    (˘-monotone (˘-coinvolution ⟨≈⊑⟩ (L-monotone ⟨⊑≈˘⟩ ˘-coinvolution)) ⟨⊑≈⟩ ˘˘)

L-comonotone : E1 ˘ ⨾ L ⊑ L ⨾ E2 ˘
L-comonotone = swap-⨾-⊑-total˘ L.total
    (⨾-assoc ⟨≈˘⊑⟩ swap-⊑-⨾-unival˘ L.unival L-monotone˘)

L-comonotone˘ : L ˘ ⨾ E1 ⊑ E2 ⨾ L ˘
L-comonotone˘ = ˘-involutionLeftConv
```

```
      ⟨≈˘⊑⟩ ˘-monotone L-comonotone ⟨⊑≈⟩ ˘-involutionRightConv

  U-comonotone : E2 ˘ ⨾ U ⊑ U ⨾ E1 ˘
  U-comonotone = ⊑-begin
          E2 ˘ ⨾ U
        ⊑⟨ ⨾-monotone2 (proj1 E˘L˘U˘-supId) ⟩
          E2 ˘ ⨾ (E2 ˘ ⨾ L ˘ ⨾ U ˘) ⨾ U
        ≈⟨ ⨾-22assoc121 ⟨≈˘≈⟩ (⨾-cong2 ⨾-assoc ⟨≈≈⟩ ⨾-cong1 A2.˘-idempot) ⟩
          E2 ˘ ⨾ L ˘ ⨾ U ˘ ⨾ U
        ⊑⟨ ⨾-monotone2 (proj2 U.unival) ⟩
          E2 ˘ ⨾ L ˘
        ≈˘⟨ gc˘ ⟩
          U ⨾ E1 ˘
        □

  U-comonotone˘ : U ˘ ⨾ E2 ⊑ E1 ⨾ U ˘
  U-comonotone˘ = ˘˘ ⟨≈˘⊑⟩ (˘-monotone (˘-involutionLeftConv
      ⟨≈⊑⟩ (U-comonotone ⟨⊑≈˘⟩ ˘-involutionRightConv)) ⟨⊑≈⟩ ˘˘)

  U-monotone : E2 ⨾ U ⊑ U ⨾ E1
  U-monotone = swap-⨾-⊑-total˘ U.total (⨾-assoc ⟨≈˘⊑⟩ swap-⊑-⨾-unival˘ U.unival U-comonotone˘)

  U-monotone˘ : U ˘ ⨾ E2 ˘ ⊑ E1 ˘ ⨾ U ˘
  U-monotone˘ = ˘˘ ⟨≈˘⊑⟩
      (˘-monotone (˘-coinvolution ⟨≈⊑⟩ (U-monotone ⟨⊑≈˘⟩ ˘-coinvolution)) ⟨⊑≈⟩ ˘˘)
```

6.3.4 Quasi-semi-inverse Laws

As is known, "an adjoint sandwiched by its friend is just the friend". That is, the adjoints are semi-inverse. Without antisymmetry, we have only been able to show that they are "indirectly" semi-inverse vis à vis the order appended.

```
  LULE≈LE : (L ⨾ U ⨾ L) ⨾ E2 ≈ L ⨾ E2
  LULE≈LE = ⊑-antisym (⊑-begin
          (L ⨾ U ⨾ L) ⨾ E2
    ≈⟨ ⨾-cong2 A2.idempot ⟨≈˘≈⟩ (⨾-assoc ⟨≈˘≈⟩ (⨾-cong1 ⨾-assoc3+1 ⟨≈≈⟩ ⨾-assoc)) ⟩
          L ⨾ (U ⨾ L ⨾ E2) ⨾ E2
    ≈⟨ (⨾-cong212 gc ⟨≈≈˘⟩ ⨾-assoc) ⟩
          (L ⨾ U ⨾ E1 ⨾ U ˘) ⨾ E2
    ⊑⟨ ⨾-monotone1 (⨾-assocL4 ⟨≈⊑⟩ ⨾-monotone11 LU-⊑-E ⟨⊑≈⟩ ⨾-assoc) ⟩
          (E1 ⨾ E1 ⨾ U ˘) ⨾ E2
    ≈⟨ ⨾-cong1 (⨾-assoc ⟨≈˘≈⟩ ⨾-cong1 A1.idempot) ⟨≈≈˘⟩ ⨾-cong1 gc ⟩
          (L ⨾ E2) ⨾ E2
    ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong2 A2.idempot ⟩
          L ⨾ E2
      □) (⊑-begin
          L ⨾ E2
      ⊑⟨ ⨾-monotone2 (proj1 ULE-supId) ⟩
          L ⨾ (U ⨾ L ⨾ E2) ⨾ E2
      ≈⟨ ⨾-assoc ⟨≈˘≈⟩ (⨾-cong1 ⨾-assoc3+1  ⟨≈˘≈⟩ ⨾-assoc) ⟩
          (L ⨾ U ⨾ L) ⨾ E2 ⨾ E2
      ≈⟨ ⨾-cong2 A2.idempot ⟩
          (L ⨾ U ⨾ L) ⨾ E2
    □)

  ULUE˘≈UE˘ : (U ⨾ L ⨾ U) ⨾ E1 ˘ ≈ U ⨾ E1 ˘
  ULUE˘≈UE˘ = ⊑-antisym (⊑-begin
          (U ⨾ L ⨾ U) ⨾ E1 ˘
      ⊑⟨ ⨾-monotone1 (⨾-assoc ⟨≈˘⊑⟩ ⨾-monotone1 UL-⊑-E˘) ⟩
          (E2 ˘ ⨾ U) ⨾ E1 ˘
      ⊑⟨ ⨾-monotone1 U-comonotone ⟨⊑≈⟩ ⨾-assoc ⟩
          U ⨾ E1 ˘ ⨾ E1 ˘
      ⊑⟨      ⨾-monotone2 A1.˘-trans ⟩
          U ⨾ E1 ˘
      □) (⊑-begin
          U ⨾ E1 ˘
      ⊑⟨ ⨾-monotone2 (proj1 L.total ⟨⊑≈⟩ ⨾-assoc) ⟩
          U ⨾ L ⨾ L ˘ ⨾ E1 ˘
      ≈˘⟨ ⨾-cong222 A1.˘-idempot ⟩
          U ⨾ L ⨾ L ˘ ⨾ E1 ˘ ⨾ E1 ˘
      ⊑⟨ ⨾-monotone22 (⨾-assoc ⟨≈˘⊑⟩ ⨾-monotone1 L-monotone˘ ⟨⊑≈⟩ ⨾-assoc) ⟩
          U ⨾ L ⨾ E2 ˘ ⨾ L ˘ ⨾ E1 ˘
      ≈⟨ ⨾-cong2 ⨾-assocL3+1 ⟨≈≈˘⟩ ⨾-cong212 gc˘ ⟩
          U ⨾ (L ⨾ U ⨾ E1 ˘) ⨾ E1 ˘
      ≈⟨ ⨾-cong2 (⨾-assoc3+1 ⟨≈≈⟩ ⨾-cong22 A1.˘-idempot) ⟨≈≈⟩ ⨾-assocL3+1 ⟩
          (U ⨾ L ⨾ U) ⨾ E1 ˘
      □)
```

6.3.5 Quasi-absorption Laws

We also have that adjoints quasi-absorb one another — due to the lack of antisymmetry.

```
  L-absE : {C : Obj} {Q R : Mor C A1}  → R ⨾ L ⨾ U ≈ Q ⨾ L ⨾ U → R ⨾ L ⨾ E2 ≈ Q ⨾ L ⨾ E2
  L-absE {C} {Q} {R} RLU≈QLU = ≈-begin
          R ⨾ L ⨾ E2
        ≈˘⟨ ⨾-cong2 (⨾-assoc3+1 ⟨≈˘≈⟩ LULE≈LE) ⟩
          R ⨾ L ⨾ U ⨾ L ⨾ E2
        ≈⟨ ⨾-assocL3+1 ⟨≈≈⟩ ⨾-cong1 RLU≈QLU ⟨≈≈⟩ ⨾-assoc3+1 ⟩
          Q ⨾ L ⨾ U ⨾ L ⨾ E2
        ≈⟨    ⨾-cong2 (⨾-assoc3+1 ⟨≈˘≈⟩ LULE≈LE) ⟩
          Q ⨾ L ⨾ E2
        □

  U-absE˘ : {C : Obj} {Q R : Mor C A2} → R ⨾ U ⨾ L ≈ Q ⨾ U ⨾ L → R ⨾ U ⨾ E1 ˘ ≈ Q ⨾ U ⨾ E1 ˘
  U-absE˘ {C} {Q} {R} RUL≈QUL = ≈-begin
          R ⨾ U ⨾ E1 ˘
        ≈⟨ ⨾-cong2 ULUE˘≈UE˘ ⟨≈˘≈˘⟩ ⨾-assoc ⟩
          (R ⨾ U ⨾ L ⨾ U) ⨾ E1 ˘
        ≈⟨ ⨾-cong1 (⨾-assoc3+1 ⟨≈˘≈⟩ ⨾-cong1 RUL≈QUL)
          ⟨≈≈⟩ (⨾-cong1 ⨾-assoc3+1 ⟨≈≈⟩ (⨾-assoc ⟨≈≈⟩ ⨾-cong2 ⨾-assoc3+1)) ⟩
          Q ⨾ U ⨾ L ⨾ U ⨾ E1 ˘
        ≈⟨ ⨾-cong2 (⨾-assoc3+1 ⟨≈˘≈⟩ ULUE˘≈UE˘) ⟩
          Q ⨾ U ⨾ E1 ˘
        □
```

6.3.6 Image Isotonicity

However, we can show the adjoints are isotonic on each others image.

```
  L-isotone-on-U : U ⨾ L ⨾ E2 ⨾ L ˘ ⨾ U ˘ ≈ U ⨾ E1 ⨾ U ˘
  L-isotone-on-U = ≈-sym (≈-begin
          U ⨾ E1 ⨾ U ˘
      ≈⟨ ⨾-cong21 A1.idempot ⟨≈˘≈⟩ ⨾-cong2 ⨾-assoc ⟩
          U ⨾ E1 ⨾ E1 ⨾ U ˘
      ≈˘⟨ ⨾-cong22 ˘-involutionRightConv ⟩
          U ⨾ E1 ⨾ (U ⨾ E1 ˘) ˘
      ≈˘⟨ ⨾-cong22 (˘-cong ULUE˘≈UE˘) ⟩
          U ⨾ E1 ⨾ ((U ⨾ L ⨾ U) ⨾ E1 ˘) ˘
```

```
    ≈⟨ ⨾-cong₂₂ ˘-involutionRightConv ⟩
        U ⨾ E₁ ⨾ E₁ ⨾ (U ⨾ L ⨾ U) ˘
    ≈⟨ ⨾-cong₂ (⨾-assoc ⟨≈˘≈⟩ (⨾-cong₁ A₁.idempot ⟨≈≈⟩ ⨾-cong₂ ˘-involution)) ⟩
        U ⨾ E₁ ⨾ ((L ⨾ U) ˘ ⨾ U ˘)
    ≈⟨ ⨾-cong₂₂ (⨾-cong₁ ˘-involution ⟨≈≈⟩ ⨾-assoc) ⟩
        U ⨾ E₁ ⨾ (U ˘ ⨾ L ˘ ⨾ U ˘)
    ≈˘⟨ ⨾-cong₂ ⨾-assoc ⟩
        U ⨾ (E₁ ⨾ U ˘) ⨾ L ˘ ⨾ U ˘
    ≈˘⟨ ⨾-cong₂₁ gc ⟩
        U ⨾ (L ⨾ E₂) ⨾ L ˘ ⨾ U ˘
    ≈⟨ ⨾-cong₂ ⨾-assoc ⟩
        U ⨾ L ⨾ E₂ ⨾ L ˘ ⨾ U ˘
    □)
L-coisotone-on-U : U ⨾ L ⨾ E₂ ˘ ⨾ L ˘ ⨾ U ˘ ≈ U ⨾ E₁ ˘ ⨾ U ˘
L-coisotone-on-U = ≈-begin
        U ⨾ L ⨾ E₂ ˘ ⨾ L ˘ ⨾ U ˘
    ≈˘⟨ ⨾-cong₁ ⨾-assoc ⟨≈≈⟩ (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ⨾-assoc) ⟩
        ((U ⨾ L) ⨾ E₂ ˘) ⨾ L ˘ ⨾ U ˘
    ≈˘⟨ ⨾-cong₁ (˘-involution ⟨≈≈⟩ ⨾-cong₁ ˘-coinvolution) ⟩
        (E₂ ⨾ L ˘ ⨾ U ˘) ˘ ⨾ L ˘ ⨾ U ˘
    ≈˘⟨ ˘-involution ⟨≈≈⟩ ⨾-cong₁ ˘-involution ⟨≈≈⟩ ⨾-assoc ⟩
        (U ⨾ L ⨾ E₂ ⨾ L ˘ ⨾ U ˘) ˘
    ≈⟨ ˘-cong L-isotone-on-U ⟩
        (U ⨾ E₁ ⨾ U ˘) ˘
    ≈⟨ ˘-involution ⟨≈≈⟩ (⨾-cong₁ ˘-involutionRightConv ⟨≈≈⟩ ⨾-assoc) ⟩
        U ⨾ E₁ ˘ ⨾ U ˘
    □
U-coisotone-on-L : L ⨾ U ⨾ E₁ ˘ ⨾ U ˘ ⨾ L ˘ ≈ L ⨾ E₂ ˘ ⨾ L ˘
U-coisotone-on-L = ≈-sym (≈-begin
        L ⨾ E₂ ˘ ⨾ L ˘
    ≈⟨ ⨾-cong₂₁ A₂.˘-idempot ⟨≈˘≈⟩ ⨾-cong₂ ⨾-assoc ⟩
        L ⨾ E₂ ˘ ⨾ E₂ ˘ ⨾ L ˘
    ≈˘⟨ ⨾-cong₂₂ ˘-involution ⟩
        L ⨾ E₂ ˘ ⨾ (L ⨾ E₂) ˘
    ≈˘⟨ ⨾-cong₂₂ (˘-cong LULE≈LE) ⟩
        L ⨾ E₂ ˘ ⨾ ((L ⨾ U ⨾ L) ⨾ E₂) ˘
    ≈⟨ ⨾-cong₂₂ ˘-involution ⟩
        L ⨾ E₂ ˘ ⨾ E₂ ˘ ⨾ (L ⨾ U ⨾ L) ˘
    ≈⟨ ⨾-cong₂ (⨾-assoc ⟨≈˘≈⟩ (⨾-cong₁ A₂.˘-idempot ⟨≈≈⟩ ⨾-cong₂ ˘-involution)) ⟩
        L ⨾ E₂ ˘ ⨾ ((U ⨾ L) ˘ ⨾ L ˘)
    ≈⟨ ⨾-cong₂₂ (⨾-cong₁ ˘-involution ⟨≈≈⟩ ⨾-assoc) ⟩
        L ⨾ E₂ ˘ ⨾ (L ˘ ⨾ U ˘ ⨾ L ˘)
    ≈˘⟨ ⨾-cong₂ ⨾-assoc ⟩
        L ⨾ (E₂ ˘ ⨾ L ˘) ⨾ U ˘ ⨾ L ˘
    ≈˘⟨ ⨾-cong₂₁ gc˘ ⟩
        L ⨾ (U ⨾ E₁ ˘) ⨾ U ˘ ⨾ L ˘
    ≈⟨ ⨾-cong₂ ⨾-assoc ⟩
        L ⨾ U ⨾ E₁ ˘ ⨾ U ˘ ⨾ L ˘
    □)
U-isotone-on-L : L ⨾ U ⨾ E₁ ⨾ U ˘ ⨾ L ˘ ≈ L ⨾ E₂ ⨾ L ˘
U-isotone-on-L = ≈-begin
        L ⨾ U ⨾ E₁ ⨾ U ˘ ⨾ L ˘
    ≈˘⟨ ⨾-cong₁ ⨾-assoc ⟨≈≈⟩ (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ⨾-assoc) ⟩
        ((L ⨾ U) ⨾ E₁) ⨾ U ˘ ⨾ L ˘
    ≈˘⟨ ⨾-cong₁ (˘-involutionLeftConv ⟨≈≈⟩ ⨾-cong₁ ˘-coinvolution) ⟩
      (E₁ ˘ ⨾ U ˘ ⨾ L ˘) ˘ ⨾ U ˘ ⨾ L ˘
    ≈˘⟨ ˘-involution ⟨≈≈⟩ ⨾-cong₁ ˘-involution ⟨≈≈⟩ ⨾-assoc ⟩
        (L ⨾ U ⨾ E₁ ˘ ⨾ U ˘ ⨾ L ˘) ˘
    ≈⟨ ˘-cong U-coisotone-on-L ⟩
        (L ⨾ E₂ ˘ ⨾ L ˘) ˘
    ≈⟨ ˘-involution ⟨≈≈⟩ ⨾-cong₁ ˘-coinvolution ⟨≈≈⟩ ⨾-assoc ⟩
        L ⨾ E₂ ⨾ L ˘
    □
```

6.3.7 Induced Interior

Finally, we have that the lower adjoint followed by the upper constitute an interior operator.

```
interior : (U ⨾ L) ⨾ E₂ ⨾ (U ⨾ L) ˘ ≈ (U ⨾ L) ⨾ E₂
interior = ≈-begin
        (U ⨾ L) ⨾ E₂ ⨾ (U ⨾ L) ˘
    ≈⟨ ⨾-cong₂₂ ˘-involution ⟨≈≈⟩ ⨾-assoc ⟩
        U ⨾ L ⨾ E₂ ⨾ L ˘ ⨾ U ˘
    ≈⟨ L-isotone-on-U ⟩
        U ⨾ E₁ ⨾ U ˘
    ≈⟨ ⨾-cong₂ gc ⟨≈˘≈˘⟩ ⨾-assoc ⟩
        (U ⨾ L) ⨾ E₂
    □

UL : Mapping A₂ A₂
UL = OSGC-Props.mkMapping (U ⨾ L) (⨾-isMapping U.prf L.prf)

UL₀ : Mor A₂ A₂
UL₀ = Mapping.mor UL

isInterior : PreCoclosureOp osgc A₂-isPreorder UL
isInterior = record {char =          ≈-sym interior}

open PreCoclosureOp osgc isInterior public renaming
    (char˘ to interior˘                   -- E₂ ˘ ⨾ (U ⨾ L) ˘ ≈ (U ⨾ L) ⨾ E₂ ˘ ⨾ (U ⨾ L) ˘
    ; CE˘C˘-supId to UL⨾E˘⨾UL˘-supId      -- isSuperidentity ((U ⨾ L) ⨾ E₂ ˘ ⨾ (U ⨾ L) ˘)
    ; E˘C˘-supId  to E˘⨾UL˘-supId         -- isSuperidentity (E₂ ˘ ⨾ (U ⨾ L) ˘), cf ₂-can′
    ; decreasing  to UL-decreasing        -- U ⨾ L ⊑ E₂ ˘. cf ₂-can
    ; contraction to UL-contraction       -- E₂ ⊑ (U ⨾ L) ⨾ E₂
    ; E˘C-⊑-E˘      to E˘UL-⊑-E˘          -- E₂ ˘ ⨾ U ⨾ L ⊑ E₂ ˘
    ; CE˘-⊑-E˘      to UL⨾E˘-⊑-E˘         -- (U ⨾ L) ⨾ E₂ ˘ ⊑ E₂ ˘
    ; E˘C˘C-⊑-CE˘ to E˘⨾UL⨾UL˘-⊑-UL⨾E˘    -- E₂ ˘ ⨾ (U ⨾ L) ˘ ⨾ (U ⨾ L) ⊑ (U ⨾ L) ⨾ E₂ ˘
    ; idempE˘       to UL-idempE          -- (U ⨾ L) ⨾ (U ⨾ L) ⨾ E₂ ˘ ≈ (U ⨾ L) ⨾ E₂ ˘
    ; CE˘-⊑-E˘C˘  to UL⨾E˘-⊑-E˘-UL˘       -- (U ⨾ L) ⨾ E₂ ˘ ⊑ E₂ ˘ ⨾ (U ⨾ L) ˘
    ; monotone˘   to UL-monotone˘         -- (U ⨾ L) ˘ ⨾ E₂ ˘ ⊑ E₂ ˘ ⨾ (U ⨾ L) ˘
    ; monotone    to UL-monotone          -- E₂ ⨾ (U ⨾ L) ⊑ (U ⨾ L) ⨾ E₂
    ; comonotone to UL-comonotone         -- E₂ ˘ ⨾ (U ⨾ L) ⊑ (U ⨾ L) ⨾ E₂ ˘
    ; comonotone˘ to UL-comonotone˘       -- (U ⨾ L) ˘ ⨾ E₂ ⊑ E₂ ⨾ (U ⨾ L) ˘
    )
```

6.3.8 Induced Closure

While the reverse composition yields a closure operator.

```
closure : (L ⨾ U) ⨾ E₁ ⨾ (L ⨾ U) ˘ ≈ E₁ ⨾ (L ⨾ U) ˘
closure = ≈-begin
        (L ⨾ U) ⨾ E₁ ⨾ (L ⨾ U) ˘
    ≈⟨ ⨾-cong₂₂ ˘-involution ⟨≈≈⟩ ⨾-assoc ⟩
        L ⨾ U ⨾ E₁ ⨾ U ˘ ⨾ L ˘
    ≈⟨ U-isotone-on-L ⟩
        L ⨾ E₂ ⨾ L ˘
    ≈⟨ ⨾-assoc ⟨≈˘≈⟩ ⨾-cong₁ gc ⟩
```

```
        (E₁ ⨾ U ˘) ⨾ L ˘
    ≈⟨ ⨾-assoc ⟨≈≈˘⟩ ⨾-cong₂ ˘-involution ⟩
        E₁ ⨾ (L ⨾ U) ˘
    □

LU : Mapping A₁ A₁
LU = OSGC-Props.mkMapping (L ⨾ U) (⨾-isMapping L.prf U.prf)

LU₀ : Mor A₁ A₁
LU₀ = Mapping.mor LU

isClosure : PreClosureOp osgc A₁-isPreorder LU
isClosure = record {char =   ≈-sym closure}


open PreClosureOp osgc isClosure public hiding (char) renaming
    (char˘          to closure˘            -- (L ⨾ U) ⨾ E₁ ˘ ≈ (L ⨾ U) ⨾ E₁ ˘ ⨾ (L ⨾ U) ˘
    ; CEC˘-supId    to LU⨾E⨾LU˘-supId      -- isSuperidentity ((L ⨾ U) ⨾ E₁ ⨾ (L ⨾ U) ˘)
    ; EC˘-supId     to E⨾LU˘-supId         -- isSuperidentity (E₁ ⨾ (L ⨾ U) ˘)
    ; increasing    to LU-decreasing       -- L ⨾ U ⊑ E₁
    ; expansion     to LU-contraction      -- E₁ ⊑ E₁ ⨾ C ˘
    ; EC-⊑-E        to ELU-⊑-E             -- E₁ ⨾ L ⨾ U ⊑ E₁
    ; CE-⊑-E        to LU⨾E-⊑-E            -- (L ⨾ U) ⨾ E₁ ⊑ E₁
    ; EC˘C-⊑-CE     to E⨾LU˘⨾LU-⊑-LU⨾E     -- E₁ ⨾ (L ⨾ U) ˘ ⨾ (L ⨾ U) ⊑ (L ⨾ U) ⨾ E₁
    ; idempE        to LU-idempE           -- (L ⨾ U) ⨾ (L ⨾ U) ⨾ E₁ ≈ (L ⨾ U) ⨾ E₁
    ; CE-⊑-EC˘      to LU⨾E-⊑-E⨾LU˘        -- (L ⨾ U) ⨾ E₁ ⊑ E₁ ⨾ (L ⨾ U) ˘
    ; comonotone˘   to LU-comonotone˘      -- (L ⨾ U) ˘ ⨾ E₁ ⊑ E₁ ⨾ (L ⨾ U) ˘
    ; comonotone    to LU-comonotone       -- E₁ ˘ ⨾ (L ⨾ U) ⊑ (L ⨾ U) ⨾ E₁ ˘
    ; monotone      to LU-monotone         -- E₁ ⨾ (L ⨾ U) ⊑ (L ⨾ U) ⨾ E₁
    ; monotone˘     to LU-monotone˘        -- (L ⨾ U) ˘ ⨾ E₁ ˘ ⊑ E₁ ˘ ⨾ (L ⨾ U) ˘
    )
```

So much for the theory of internal Galois Connections between two preorders and in OSGCs.

6.4 Categoric.OCC.Preorder.Galois

With the addition of identities, we do not gain much. Essentially the only new results are that the sub- and super-identity formulations now take on new, equivalent, formulations via identities.

```
module _ {i j k₁ k₂} {Obj : Set i} (occ : OCC j k₁ k₂ Obj) where
  open OCC occ


  record PreGaloisConnection {A₁ A₂ : Obj} {E₁ : Mor A₁ A₁} {E₂ : Mor A₂ A₂}
                             (A₁-isPreorder : IsPreorder occ E₁)
                             (A₂-isPreorder : IsPreorder occ E₂)
                             (LL : Mapping A₁ A₂) (UU : Mapping A₂ A₁) : Set k₁ where
    private
      module A₁ = IsPreorder occ A₁-isPreorder
      module A₂ = IsPreorder occ A₂-isPreorder
      module L = Mapping LL
      module U = Mapping UU
    open L using () renaming (mor to L)
    open U using () renaming (mor to U)
    field gc : L ⨾ E₂ ≈ E₁ ⨾ U ˘
    open import Categoric.OSGC.Preorder.Galois
    open PreGaloisConnection osgc {A₁} {A₂} {E₁} {E₂}
      {A₁.isPreorder₀} {A₂.isPreorder₀} {LL} {UU} (record {gc = gc}) public hiding (gc)
```

The aforementioned cancellation laws, now, with the appearance of identities, resemble their pointwise counterparts.

```
    EU˘L˘-refl : Id ⊑ E₁ ⨾ U ˘ ⨾ L ˘
    EU˘L˘-refl = swap-⨾-⊑-total L.total (leftId ⟨≈⊑⟩ L-⊑-E⨾U˘) ⟨⊑≈⟩ ⨾-assoc
    LUE˘-refl : Id ⊑ L ⨾ U ⨾ E₁ ˘
    LUE˘-refl = Id˘ ⟨≈˘⊑⟩ (˘-monotone EU˘L˘-refl
      ⟨⊑≈⟩ (˘-cong (⨾-cong₂ ˘-involution)
      ⟨≈˘≈⟩ ˘-involutionRightConv ⟨≈≈⟩ ⨾-assoc))
    E˘L˘U˘-refl : Id ⊑ E₂ ˘ ⨾ L ˘ ⨾ U ˘
    E˘L˘U˘-refl = swap-⨾-⊑-total U.total (leftId ⟨≈⊑⟩ U-⊑-E˘⨾L˘) ⟨⊑≈⟩ ⨾-assoc
    ULE-refl : Id ⊑ U ⨾ L ⨾ E₂
    ULE-refl = proj₁ ULE-supId ⟨⊑≈⟩ rightId
```

Note that each of the above could have had an indirect proof of the shape:

```
    X-refl = proj₁ X-supId ⟨⊑≈⟩ rightId
```

However, it seems that the direct proofs result in smaller size normal forms; with the exception of

```
    ULE-refl = Id˘ ⟨≈˘⊑⟩ (˘-monotone E˘L˘U˘-refl
      ⟨⊑≈⟩ (˘-cong (⨾-cong₂ ˘-involution)
      ⟨≈˘≈⟩ (˘-coinvolution ⟨≈≈⟩ ⨾-assoc)))
```

This is nearly three times larger than the indirect proof.

6.5 Categoric.OCC.Order.Galois

Within a partial order, we have indirect equality, and so obtain full results rather than the 'quasi' forms presented earlier.

```
record GaloisConnection {A₁ A₂ : Obj} {E₁ : Mor A₁ A₁} {E₂ : Mor A₂ A₂}
  (A₁-isOrder : IsOrder E₁) (A₂-isOrder : IsOrder E₂)
  (LL : Mapping A₁ A₂) (UU : Mapping A₂ A₁) : Set k₁ where
  private
    module A₁ = IsOrder A₁-isOrder
    module A₂ = IsOrder A₂-isOrder
    module L = Mapping LL
    module U = Mapping UU
  open L using () renaming (mor to L)
  open U using () renaming (mor to U)
  field gc : L ⨾ E₂ ≈ E₁ ⨾ U ˘
  open PreGaloisConnection {A₁} {A₂} {E₁} {E₂} {A₁.isPreorder₀} {A₂.isPreorder₀} {LL} {UU}
    (record {gc = gc}) public hiding (gc)
```

Semi-inverses

```
  L-semi-inverse : L ⨾ U ⨾ L ≈ L
  L-semi-inverse = A₂.indirect-≈ (⨾-isMapping L.prf (⨾-isMapping U.prf L.prf)) L.prf LULE≈LE
  U-semi-inverse : U ⨾ L ⨾ U ≈ U
  U-semi-inverse = A₁.˘-indirect-≈ (⨾-isMapping U.prf (⨾-isMapping L.prf U.prf)) U.prf ULUE˘≈UE˘
```

Map Absorption

We also obtain another form of absorption results:

```
L-map-absorption : {C : Obj} {Q R : Mor C A₁} → isMapping Q → isMapping R
  → R ⨾ L ⨾ U ≈ Q ⨾ L ⨾ U → R ⨾ L ≈ Q ⨾ L
L-map-absorption Qmap Rmap eq = A₂.indirect-≈
  (⨾-isMapping Rmap L.prf) (⨾-isMapping Qmap L.prf)
  (⨾-assoc ⟨≈≈⟩ L-absE eq ⟨≈≈˘⟩ ⨾-assoc)
U-map-absorption : {C : Obj} {Q R : Mor C A₂} → isMapping Q → isMapping R
  → R ⨾ U ⨾ L ≈ Q ⨾ U ⨾ L → R ⨾ U ≈ Q ⨾ U
U-map-absorption Qmap Rmap eq = A₁.˘-indirect-≈
  (⨾-isMapping Rmap U.prf) (⨾-isMapping Qmap U.prf)
  (⨾-assoc ⟨≈≈⟩ U-absE˘ eq ⟨≈≈˘⟩ ⨾-assoc)
```

Idempotency and Coclosure

Likewise we obtain certain new results, among which is idempotency:

```
open CoclosureOp {A₂} {E₂} {A₂-isOrder} {UL} (record {char = ≈-sym interior})
  public using () renaming
  (idempot to UL-idempot
      -- : UL ⨾₁ UL ≈₁ UL
  ; ranClosed-← to UL-ranClosed-←
      -- : ∀ {R} → R ⨾ UL₀ ≈ R → R ⊑ R ⨾ UL₀ ˘ ⨾ UL₀
  ; ranClosed-→ to UL-ranClosed-→
      -- : ∀ {R} → R ⊑ R ⨾ UL₀ ˘ ⨾ UL₀ → R ⨾ UL₀ ≈ R
  ; lub-closed-⊑ to UL-lub-closed-⊑
      -- : ∀ {R} → R ⨾ UL₀ ≈ R → lub R ⨾ UL₀ ⊑ lub R
  ; lub-closed to UL-lub-closed
      -- : ∀ {R} → isTotal (lub R) → R ⨾ UL₀ ≈ R → lub R ⨾ UL₀ ≈ lub R
  ; lub-closed′ to UL-lub-closed′
      -- : ∀ {R} → isTotal (lub R) → R ⨾ UL₀ ≈ R → lub R ⊑ lub R ⨾ UL₀ ˘ ⨾ UL₀
  )
```

Idempotency and Closure

Dually:

```
open ClosureOp {A₁} {E₁} {A₁-isOrder} {LU} (record {char = ≈-sym closure})
  public using () renaming
  (idempot to LU-idempot
      -- : LU ⨾₁ LU ≈₁ LU
  ; ranClosed-← to LU-ranClosed-←
      -- : ∀ {R} → R ⨾ LU₀ ≈ R → R ⊑ R ⨾ LU₀ ˘ ⨾ LU₀
  ; ranClosed-→ to LU-ranClosed-→
      -- : ∀ {R} → R ⊑ R ⨾ LU₀ ˘ ⨾ LU₀ → R ⨾ LU₀ ≈ R
  ; glb-closed-⊑ to LU-glb-closed-⊑
      -- : ∀ {R} → R ⨾ LU₀ ≈ R → glb R ⨾ LU₀ ⊑ glb R
  ; glb-closed to LU-glb-closed
      -- : ∀ {R} → isTotal (glb R) → R ⨾ LU₀ ≈ R → glb R ⨾ LU₀ ≈ glb R
  ; glb-closed′ to LU-glb-closed′
      -- : ∀ {R} → isTotal (glb R) → R ⨾ LU₀ ≈ R → glb R ⊑ glb R ⨾ LU₀ ˘ ⨾ LU₀
  )
```

6.6 Categoric.OCC.DirectPower.Polarities

```
module Categoric.OCC.DirectPower.Polarities {i j k₁ k₂} {Obj : Set i} (occ : OCC j k₁ k₂ Obj)
  (let open OCC occ)
  (leftResOp  : LeftResOp orderedSemigroupoid)
  (rightResOp : RightResOp orderedSemigroupoid)
  (syqOp      : SyqOp osgc)
  (let open OCC-DirectPower occ leftResOp rightResOp syqOp)
  (directPower : DirectPower)
  where
private
  module ℙ = DirectPower directPower
open ℙ using
  (ℙ; ∈; Ω; Ω˘; Ω-isOrder; Ω˘-isOrder; Ω-isPreorder; Ω˘-isPreorder; powerOp; Λ; Λ₀; Λ-isMapping; Λ-cong
  ; Λ⨾Ω˘; ∈⇒Λ; Ω˘-isPreorder₀; ∈⨾-∈\
  )
open SyqOp                                                       syqOp
open OCC-SyQ-Props          occ                                  syqOp
open SyQ-ResidualProps      osgc            leftResOp rightResOp syqOp
open ResidualOps                            leftResOp rightResOp
open OrdCat-Residual-Props orderedCategory  leftResOp rightResOp
open OSGC-Residuals         osgc            leftResOp rightResOp
open import Categoric.OCC.Order occ leftResOp rightResOp syqOp
open import Categoric.OCC.Order.Closure occ leftResOp rightResOp syqOp
open PowerOp osgc powerOp using (Λ-⨾∈˘; Λ⨾∈˘; ∈⨾Λ˘; map-Λ)
open Category₁ (MapCat occ)
open import Categoric.OSGC.PowerOrder       osgc leftResOp rightResOp powerOp using ()
open import Categoric.OSGC.Power.Polarities osgc leftResOp rightResOp powerOp
open import Categoric.OCC.Preorder.Galois

module _ {A B : Obj} {R : Mor A B} where
  open PreGaloisConnection occ {ℙ A} {ℙ B} {Ω} {Ω ˘} {Ω-isPreorder} {Ω˘-isPreorder} {R ↑} {R ↓}
    (record {gc = ≈-sym Galois-↓-↑}) public renaming
      (gc˘ to Galois-↓-↑-˘₀                           -- R ↓₀ ⨾ Ω ˘ ≈ Ω ˘ ˘ ⨾ (R ↑₀) ˘
          -- gc ≐ ≈-sym Galois-↓-↑ ≐ R ↑₀ ⨾ Ω ˘ ≈ Ω ⨾ (R ↓₀) ˘
      ; L-⊑-E⨾U˘   to ↑-⊑-Ω⨾↓˘                      -- R ↑₀ ⊑ Ω ⨾ R ↓₀ ˘
      ; LU-⊑-E     to  ↑↓-⊑-Ω                       -- R ↑↓₀ ⊑ Ω
      ; U˘⨾L˘-⊑-E˘ to ↓˘↑˘-⊑-Ω˘                     -- (R ↓₀) ˘ ⨾ (R ↑₀) ˘ ⊑ Ω ˘
      ; EU˘L˘-supId to Ω⨾↓˘↑˘-supId                 -- isSuperidentity (Ω ⨾ R ↓₀ ˘ ⨾ R ↑₀ ˘)
      ; LUE˘-supId to ↑⨾↓⨾Ω˘-supId                  -- isSuperidentity (R ↑₀ ⨾ R ↓₀ ⨾ Ω ˘)
      ; EU˘L˘-refl to Ω⨾↓˘↑˘-refl                   -- Id ⊑ Ω ⨾ (R ↓₀) ˘ ⨾ (R ↑₀) ˘
      ; LUE˘-refl   to ↑⨾↓⨾Ω˘-refl                  -- Id ⊑ R ↑₀ ⨾ R ↓₀ ⨾ Ω ˘
      ; U-⊑-E˘⨾L˘      to ↓-⊑-Ω˘˘↑˘                 -- R ↓₀ ⊑ Ω ˘ ˘ ⨾ R ↑₀ ˘
      ; UL-⊑-E˘        to ↓↑-⊑-Ω˘˘                  -- R ↓₀ ⨾ R ↑₀ ⊑ Ω ˘ ˘
      ; L˘U˘-⊑-E      to ↑˘↓˘-⊑-Ω˘                  -- R ↑₀ ˘ ⨾ R ↓₀ ˘ ⊑ Ω ˘
      ; E˘L˘U˘-supId  to Ω˘˘⨾↑˘↓˘-supId             -- isSuperidentity (Ω ˘ ˘ ⨾ R ↑₀ ˘ ⨾ R ↓₀ ˘)
      ; ULE-supId     to ↓⨾↑⨾Ω˘-supId               -- isSuperidentity (R ↓₀ ⨾ R ↑₀ ⨾ Ω ˘)
      ; E˘L˘U˘-refl    to Ω˘˘⨾↑˘↓˘-refl             -- Id ⊑ Ω ˘ ˘ ⨾ R ↑₀ ˘ ⨾ R ↓₀ ˘
      ; ULE-refl      to ↓⨾↑⨾Ω˘-refl                -- Id ⊑ R ↓₀ ⨾ R ↑₀ ⨾ Ω ˘
      ; L-monotone    to ↑-antitone                 -- Ω ⨾ R ↑₀ ⊑ R ↑₀ ⨾ Ω ˘
      ; L-monotone˘   to ↑-antitone˘₀               -- R ↑₀ ˘ ⨾ Ω ˘ ⊑ Ω ˘ ˘ ⨾ R ↑₀ ˘
      ; L-comonotone to ↑-coantitone₀               -- Ω ˘ ⨾ R ↑₀ ⊑ R ↑₀ ⨾ Ω ˘ ˘
      ; L-comonotone˘ to ↑-coantitone˘              -- R ↑₀ ˘ ⨾ Ω ⊑ Ω ˘ ⨾ R ↑₀ ˘
      ; U-comonotone to ↓-coantitone₀               -- Ω ˘ ˘ ⨾ R ↓₀ ⊑ R ↓₀ ⨾ Ω ˘
      ; U-comonotone˘ to ↓-coantitone˘              -- R ↓₀ ˘ ⨾ Ω ˘ ⊑ Ω ⨾ R ↓₀ ˘
      ; U-monotone    to ↓-antitone                 -- Ω ˘ ⨾ R ↓₀ ⊑ R ↓₀ ⨾ Ω
      ; U-monotone˘   to ↓-antitone˘₀               -- R ↓₀ ˘ ⨾ Ω ˘ ˘ ⊑ Ω ˘ ⨾ R ↓₀ ˘
```

```
    ; LULE≈LE  to ↑↓↑-semi-Ω˘                  -- (R ↑0 ⨾ R ↓↑0) ⨾ Ω ˘ ≈ R ↑0 ⨾ Ω ˘
    ; ULUE˘≈UE˘  to ↓↑↓-semi-Ω˘                -- (R ↓0 ⨾ R ↑↓0) ⨾ Ω ˘ ≈ R ↓0 ⨾ Ω ˘
    ; L-absE to ↑↓-abs-Ω
        -- {C : Obj} {Q S : Mor C (ℙ A)} → S ⨾ R ↑↓0 ≈ Q ⨾ R ↑↓0 → S ⨾ R ↑0 ⨾ Ω ˘ ≈ Q ⨾ R ↑0 ⨾ Ω ˘
    ; U-absE˘ to ↓↑abs-Ω˘
        -- {C : Obj} {Q S : Mor C (ℙ B)} → S ⨾ R ↓↑0 ≈ Q ⨾ R ↓↑0 → S ⨾ R ↓0 ⨾ Ω ˘ ≈ Q ⨾ R ↓0 ⨾ Ω ˘
    ; L-isotone-on-U     to ↑-isotone-on-↓      -- R ↓0 ⨾ R ↑0 ⨾ Ω ˘ ⨾ R ↑0 ˘ ⨾ R ↓0 ˘ ≈ R ↓0 ⨾ Ω ⨾ R ↓0 ˘
    ; L-coisotone-on-U to ↑-coisotone-on-↓0     -- R ↓0 ⨾ R ↑0 ⨾ Ω ˘ ˘ ⨾ R ↑0 ˘ ⨾ R ↓0 ˘ ≈ R ↓0 ⨾ Ω ˘ ⨾ R ↓0 ˘
    ; U-coisotone-on-L to ↓-coisotone-on-↑0     -- R ↑0 ⨾ R ↓0 ⨾ Ω ˘ ⨾ R ↓0 ˘ ⨾ R ↑0 ˘ ≈ R ↑0 ⨾ Ω ˘ ˘ ⨾ R ↑0 ˘
    ; U-isotone-on-L     to ↓-isotone-on-↑      -- R ↑0 ⨾ R ↓0 ⨾ Ω ⨾ R ↓0 ˘ ⨾ R ↑0 ˘ ≈ R ↑0 ⨾ Ω ˘ ⨾ R ↑0 ˘
    ; interior           to ↓↑-interior               -- R ↓↑0 ⨾ Ω ˘ ⨾ R ↓↑0 ˘ ≈ R ↓↑0 ⨾ Ω ˘
    ; interior˘          to ↓↑-interior˘0             -- Ω ˘ ˘ ⨾ R ↓↑0 ˘ ≈ R ↓↑0 ⨾ Ω ˘ ˘ ⨾ R ↓↑0 ˘
    ; UL⨾E˘-⊑-E˘ to ↓↑⨾Ω˘˘-⊑-Ω˘˘                  -- R ↓↑0 ⨾ Ω ˘ ˘ ⊑ Ω ˘ ˘
    ; E˘⨾UL˘-supId   to Ω˘˘⨾↓↑˘-supId             -- isSuperidentity (Ω ˘ ˘ ⨾ R ↓↑0 ˘)
    ; UL⨾E˘⨾UL˘-supId to ↓↑⨾Ω˘˘⨾↓↑˘-supId          -- isSuperidentity (R ↓↑0 ⨾ Ω ˘ ˘ ⨾ (R ↓↑0) ˘)
    ; UL-decreasing   to ↓↑-decreasing0            -- R ↓↑0 ⊑ Ω ˘ ˘
    ; UL-contraction to ↓↑-contraction             -- Ω ˘ ⊑ R ↓↑0 ⨾ Ω ˘
    ; E˘⨾UL⨾UL˘-⊑-UL⨾E˘ to Ω˘˘⨾↓↑⨾↓↑˘-⊑-↓↑⨾Ω˘˘       -- Ω ˘ ˘ ⨾ R ↓↑0 ˘ ⨾ R ↓↑0 ⊑ R ↓↑0 ⨾ Ω ˘ ˘
    ; UL-idempE        to ↓↑-idempΩ˘               -- R ↓↑0 ⨾ R ↓↑0 ⨾ Ω ˘ ˘ ≈ R ↓↑0 ⨾ Ω ˘ ˘
    ; UL⨾E˘-⊑-E˘-UL˘  to ↓↑⨾Ω˘˘-⊑-Ω˘˘-↓↑˘           -- R ↓↑0 ⨾ Ω ˘ ˘ ⊑ Ω ˘ ˘ ⨾ R ↓↑0 ˘
    ; UL-monotone˘  to ↓↑-comonotone˘0             -- R ↓↑0 ˘ ⨾ Ω ˘ ˘ ⊑ Ω ˘ ˘ ⨾ R ↓↑0 ˘
    ; UL-monotone    to ↓↑-comonotone              -- Ω ˘ ⨾ R ↓↑0 ⊑ R ↓↑0 ⨾ Ω ˘
    ; UL-comonotone to ↓↑-monotone0                -- Ω ˘ ˘ ⨾ R ↓↑0 ⊑ R ↓↑0 ⨾ Ω ˘ ˘
    ; UL-comonotone˘ to ↓↑-monotone˘               -- R ↓↑0 ˘ ⨾ Ω ˘ ⊑ Ω ˘ ⨾ R ↓↑0 ˘
    ; closure            to ↑↓-closure             -- R ↑↓0 ⨾ Ω ⨾ R ↑↓0 ˘ ≈ Ω ⨾ R ↑↓0 ˘
    ; closure˘           to ↑↓-closure˘            -- R ↑↓0 ⨾ Ω ˘ ≈ R ↑↓0 ⨾ Ω ˘ ⨾ R ↑↓0 ˘
    ; LU⨾E⨾LU˘-supId to ↑↓⨾Ω⨾↑↓˘-supId             -- isSuperidentity (R ↑↓0 ⨾ Ω ⨾ R ↓↑0 ˘)
    ; E⨾LU˘-supId     to Ω⨾↑↓˘-supId               -- isSuperidentity (Ω ⨾ R ↑↓0 ˘)
    ; LU-decreasing   to ↑↓-increasing             -- R ↑↓0 ⊑ Ω
    ; LU-contraction to ↑↓-contraction             -- Ω ⊑ Ω ⨾ R ↑↓0 ˘
    ; ELU-⊑-E          to Ω↓↑-⊑-Ω                  -- Ω ⨾ R ↑↓0 ⊑ Ω
      -- E˘UL-⊑-E˘ to Ω˘˘↓↑-⊑-Ω˘˘ : Ω ˘ ˘ ⨾ R ↓↑0 ⊑ Ω ˘ ˘
    ; LU⨾E-⊑-E          to ↑↓⨾Ω-⊑-Ω                -- R ↑↓0 ⨾ Ω ⊑ Ω
    ; E⨾LU˘⨾LU-⊑-LU⨾E to Ω⨾↑↓˘⨾↑↓-⊑-↑↓⨾Ω            -- Ω ⨾ R ↑↓0 ˘ ⨾ R ↑↓0 ⊑ R ↑↓0 ⨾ Ω
    ; LU-idempE        to ↑↓-idempΩ                -- R ↑↓0 ⨾ R ↑↓0 ⨾ Ω ≈ R ↑↓0 ⨾ Ω
    ; LU⨾E-⊑-E⨾LU˘  to ↑↓⨾Ω-⊑-Ω⨾↑↓˘                 -- R ↑↓0 ⨾ Ω ⊑ Ω ⨾ R ↑↓0 ˘
    ; LU-comonotone˘ to ↑↓-comonotone˘             -- R ↑↓0 ˘ ⨾ Ω ⊑ Ω ⨾ R ↑↓0 ˘
    ; LU-comonotone to ↑↓-comonotone               -- Ω ˘ ⨾ R ↑↓0 ⊑ R ↑↓0 ⨾ Ω ˘
    ; LU-monotone     to ↑↓-monotone               -- Ω ⨾ R ↑↓0 ⊑ R ↑↓0 ⨾ Ω
    ; LU-monotone˘   to ↑↓-monotone˘               -- R ↑↓0 ˘ ⨾ Ω ˘ ⊑ Ω ˘ ⨾ R ↑↓0 ˘
    )
```

Now we make some variants by eliminating double converses.

```
Galois-↓-↑-˘ : R ↓0 ⨾ Ω ˘ ≈ Ω ⨾ (R ↑0) ˘
Galois-↓-↑-˘ = Galois-↓-↑-˘0 ⟨≈≈⟩ ⨾-cong1 ˘˘

↑↓⨾Ω˘-refl : Id ⊑ R ↑↓0 ⨾ Ω ˘
↑↓⨾Ω˘-refl = ↑⨾↓⨾Ω˘-refl ⟨⊑≈˘⟩ ⨾-assoc

↑↓⨾Ω˘-supId : isSuperidentity (R ↑↓0 ⨾ Ω ˘)
↑↓⨾Ω˘-supId = reflexiveIsSuperidentity ↑↓⨾Ω˘-refl

↓↑-⊑-Ω : R ↓↑0 ⊑ Ω
↓↑-⊑-Ω = ↓↑-⊑-Ω˘˘ ⟨⊑≈⟩ ˘˘

↓-⊑-Ω↑˘ : R ↓0 ⊑ Ω ⨾ R ↑0 ˘
↓-⊑-Ω↑˘ = ↓-⊑-Ω˘˘↑˘ ⟨⊑≈⟩ ⨾-cong1 ˘˘

Ω⨾↑˘↓˘-refl : Id ⊑ Ω ⨾ R ↑0 ˘ ⨾ R ↓0 ˘
Ω⨾↑˘↓˘-refl = Ω˘˘⨾↑˘↓˘-refl ⟨⊑≈⟩ ⨾-cong1 ˘˘

Ω⨾↑˘↓˘-supId : isSuperidentity (Ω ⨾ R ↑0 ˘ ⨾ R ↓0 ˘)
Ω⨾↑˘↓˘-supId = reflexiveIsSuperidentity Ω⨾↑˘↓˘-refl

↓↑⨾Ω˘-refl : Id ⊑ R ↓↑0 ⨾ Ω ˘
↓↑⨾Ω˘-refl = ↓⨾↑⨾Ω˘-refl ⟨⊑≈˘⟩ ⨾-assoc

↓↑⨾Ω˘-supId : isSuperidentity (R ↓↑0 ⨾ Ω ˘)
↓↑⨾Ω˘-supId = reflexiveIsSuperidentity ↓↑⨾Ω˘-refl

↑-antitone˘ : R ↑0 ˘ ⨾ Ω ˘ ⊑ Ω ⨾ R ↑0 ˘
↑-antitone˘ = ↑-antitone˘0 ⟨⊑≈⟩ ⨾-cong1 ˘˘

↑-coantitone : Ω ˘ ⨾ R ↑0 ⊑ R ↑0 ⨾ Ω
↑-coantitone = ↑-coantitone0 ⟨⊑≈⟩ ⨾-cong2 ˘˘

↓-comonotone : Ω ⨾ R ↓0 ⊑ R ↓0 ⨾ Ω ˘
↓-comonotone = ⨾-cong1 ˘˘ ⟨≈˘⊑⟩ ↓-coantitone0

↓-antitone˘ : R ↓0 ˘ ⨾ Ω ⊑ Ω ˘ ⨾ R ↓0 ˘
↓-antitone˘ = ⨾-cong2 ˘˘ ⟨≈˘⊑⟩ ↓-antitone˘0

↓-coisotone-on-↑ : R ↑0 ⨾ R ↓0 ⨾ Ω ˘ ⨾ R ↓0 ˘ ⨾ R ↑0 ˘ ≈ R ↑0 ⨾ Ω ⨾ R ↑0 ˘
↓-coisotone-on-↑ = ↓-coisotone-on-↑0 ⟨≈≈⟩ ⨾-cong21 ˘˘

↑-coisotone-on-↓ : R ↓0 ⨾ R ↑0 ⨾ Ω ⨾ R ↑0 ˘ ⨾ R ↓0 ˘ ≈ R ↓0 ⨾ Ω ˘ ⨾ R ↓0 ˘
↑-coisotone-on-↓ = ⨾-cong221 ˘˘ ⟨≈˘≈⟩ ↑-coisotone-on-↓0

↓↑-interior˘ : Ω ⨾ R ↓↑0 ˘ ≈ R ↓↑0 ⨾ Ω ⨾ R ↓↑0 ˘
↓↑-interior˘ = ⨾-cong1 ˘˘ ⟨≈˘≈⟩ ↓↑-interior˘0 ⟨≈≈⟩ ⨾-cong21 ˘˘

↓↑⨾Ω-⊑-Ω : R ↓↑0 ⨾ Ω ⊑ Ω
↓↑⨾Ω-⊑-Ω = ⨾-cong2 ˘˘ ⟨≈˘⊑⟩ ↓↑⨾Ω˘˘-⊑-Ω˘˘ ⟨⊑≈⟩ ˘˘

Ω⨾↓↑˘-refl : Id ⊑ Ω ⨾ R ↓↑0 ˘
Ω⨾↓↑˘-refl = proj1 Ω˘˘⨾↓↑˘-supId ⟨⊑≈⟩ (rightId ⟨≈≈⟩ ⨾-cong1 ˘˘)

Ω⨾↓↑˘-supId : isSuperidentity (Ω ⨾ R ↓↑0 ˘)
Ω⨾↓↑˘-supId = reflexiveIsSuperidentity Ω⨾↓↑˘-refl

↓↑⨾Ω⨾↓↑˘-refl : Id ⊑ R ↓↑0 ⨾ Ω ⨾ (R ↓↑0) ˘
↓↑⨾Ω⨾↓↑˘-refl = proj1 ↓↑⨾Ω˘˘⨾↓↑˘-supId ⟨⊑≈⟩ (rightId ⟨≈≈⟩ ⨾-cong21 ˘˘)

↓↑⨾Ω⨾↓↑˘-supId : isSuperidentity (R ↓↑0 ⨾ Ω ⨾ (R ↓↑0) ˘)
↓↑⨾Ω⨾↓↑˘-supId = reflexiveIsSuperidentity ↓↑⨾Ω⨾↓↑˘-refl

Ω⨾↓↑⨾↓↑˘-⊑-↓↑⨾Ω : Ω ⨾ R ↓↑0 ˘ ⨾ R ↓↑0 ⊑ R ↓↑0 ⨾ Ω
Ω⨾↓↑⨾↓↑˘-⊑-↓↑⨾Ω = ⨾-cong1 ˘˘ ⟨≈˘⊑⟩ (Ω˘˘⨾↓↑⨾↓↑˘-⊑-↓↑⨾Ω˘˘ ⟨⊑≈⟩ ⨾-cong2 ˘˘)

↓↑-idempΩ : R ↓↑0 ⨾ R ↓↑0 ⨾ Ω ≈ R ↓↑0 ⨾ Ω
↓↑-idempΩ = ⨾-cong22 ˘˘ ⟨≈˘≈⟩ (↓↑-idempΩ˘˘ ⟨≈≈⟩ ⨾-cong2 ˘˘)

↓↑⨾Ω-⊑-Ω-↓↑˘ : R ↓↑0 ⨾ Ω ⊑ Ω ⨾ R ↓↑0 ˘
↓↑⨾Ω-⊑-Ω-↓↑˘ = ⨾-cong2 ˘˘ ⟨≈˘⊑⟩ (↓↑⨾Ω˘˘-⊑-Ω˘˘-↓↑˘ ⟨⊑≈⟩ ⨾-cong1 ˘˘)

↓↑-comonotone˘ : R ↓↑0 ˘ ⨾ Ω ⊑ Ω ⨾ R ↓↑0 ˘
↓↑-comonotone˘ = ⨾-cong2 ˘˘ ⟨≈˘⊑⟩ (↓↑-comonotone˘0 ⟨⊑≈⟩ ⨾-cong1 ˘˘)

↓↑-monotone-GC : Ω ⨾ R ↓↑0 ⊑ R ↓↑0 ⨾ Ω
↓↑-monotone-GC = ⨾-cong1 ˘˘ ⟨≈˘⊑⟩ ↓↑-monotone0 ⟨⊑≈⟩ ⨾-cong2 ˘˘

Ω⨾↑˘ : {A B : Obj} {S : Mor A B} → Ω ⨾ (S ↑0) ˘ ≈ ∈ \ (S ˘ / ∈ ˘)
Ω⨾↑˘ {A} {B} {S} = ≈-begin
      Ω ⨾ (S ↑0) ˘
    ≈⟨⟩
      (∈ \ ∈) ⨾ Λ0 (∈ \ S) ˘
    ≈⟨ \-outer-⨾-≈ Λ-isMapping ⟩
      ∈ \ (∈ ⨾ Λ0 (∈ \ S) ˘)
    ≈⟨ \-cong2 ∈⨾Λ˘ ⟩
      ∈ \ ((∈ \ S) ˘)
    ≈⟨ \-cong2 \-˘ ⟩
      ∈ \ (S ˘ / ∈ ˘)
    □

Ω⨾↓˘ : {A B : Obj} {S : Mor A B} → Ω ⨾ (S ↓0) ˘ ≈ ∈ \ (S / ∈ ˘)
```

```
Ω⨾↓˘ {A} {B} {S} = ≈-begin
    Ω ⨾ (S ↓₀) ˘
  ≈⟨⟩
    (∈ \ ∈) ⨾ Λ₀ (∈ \ S ˘) ˘
  ≈⟨ \-outer-⨾-≈ Λ-isMapping ⟩
    ∈ \ (∈ ⨾ Λ₀ (∈ \ S ˘) ˘)
  ≈⟨ \-cong₂ ∈⨾Λ˘ ⟩
    ∈ \ ((∈ \ S ˘) ˘)
≈⟨ \-cong₂ \˘-˘ ⟩
    ∈ \ (S / ∈ ˘)
  □


↓⨾∈˘ : {A B : Obj} {S : Mor A B} → S ↓₀ ⨾ ∈ ˘ ≈ ∈ \ S ˘
↓⨾∈˘ {A} {B} {S} = ≈-begin
    S ↓₀ ⨾ ∈ ˘
  ≈⟨⟩
    Λ₀ (∈ \ S ˘) ⨾ ∈ ˘
  ≈⟨ Λ⨾∈˘ ⟩
    ∈ \ S ˘
  □


↑⨾∈˘ : {A B : Obj} {S : Mor A B} → S ↑₀ ⨾ ∈ ˘ ≈ ∈ \ S
↑⨾∈˘ {A} {B} {S} = ≈-begin
    S ↑₀ ⨾ ∈ ˘
  ≈⟨⟩
    Λ₀ (∈ \ S) ⨾ ∈ ˘
  ≈⟨ Λ⨾∈˘ ⟩
    ∈ \ S
  □


Ω-⨾-↓↑˘ : {A B : Obj} {S : Mor A B} → Ω ⨾ (S ↓↑₀) ˘ ≈ (∈ \ S ˘) / (∈ \ S ˘)
Ω-⨾-↓↑˘ {A} {B} {S} = ≈-begin
    Ω ⨾ (S ↓↑₀) ˘
  ≈⟨ ⨾-cong₂ ˘-involution ⟨≈≈⟩ (⨾-assocL ⟨≈≈⟩ ⨾-cong₁ Ω⨾↑˘) ⟩
    (∈ \ (S ˘ / ∈ ˘)) ⨾ (S ↓₀) ˘
  ≈⟨ \-outer-⨾-≈ Λ-isMapping ⟩
    ∈ \ ((S ˘ / ∈ ˘) ⨾ (S ↓₀) ˘)
  ≈⟨ \-cong₂ (/-inner-⨾ Λ-isMapping) ⟩
    ∈ \ (S ˘ / (S ↓₀ ⨾ ∈ ˘))
  ≈⟨ \-cong₂ (/-cong₂ ↓⨾∈˘) ⟨≈≈⟩ \/-≈ ⟩
    (∈ \ S ˘) / (∈ \ S ˘)
  □


Ω-⨾-↑↓˘ : {A B : Obj} {S : Mor A B} → Ω ⨾ (S ↑↓₀) ˘ ≈ (∈ \ S) / (∈ \ S)
Ω-⨾-↑↓˘ {A} {B} {S} = ≈-begin
    Ω ⨾ (S ↑↓₀) ˘
  ≈⟨ ⨾-cong₂ ˘-involution ⟩
    Ω ⨾ S ↓₀ ˘ ⨾ S ↑₀ ˘
  ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ Ω⨾↓˘ ⟩
    (∈ \ (S / ∈ ˘)) ⨾ (S ↑₀) ˘
  ≈⟨ \-outer-⨾-≈ Λ-isMapping ⟩
    ∈ \ ((S / ∈ ˘) ⨾ (S ↑₀) ˘)
  ≈⟨ \-cong₂ (/-inner-⨾ Λ-isMapping) ⟩
    ∈ \ (S / (S ↑₀ ⨾ ∈ ˘))
  ≈⟨ \/-≈ ⟩
    (∈ \ S) / (S ↑₀ ⨾ ∈ ˘)
  ≈⟨ /-cong₂ ↑⨾∈˘ ⟩
    (∈ \ S) / (∈ \ S)
  □


↓≈≬ : {A B : Obj} {S : Mor A B} → S ↓₀ ≈ (S / ∈ ˘) ≬ ∈
↓≈≬ {A} {B} {S} = ≈-begin
  S ↓₀
  ≈⟨⟩
    Λ₀ (∈ \ S ˘)
  ≈⟨⟩
    (∈ \ S ˘) ˘ ≬ ∈
  ≈⟨ ≬-cong₁ \˘-˘ ⟩
    (S / ∈ ˘) ≬ ∈
  □


↑≈≬ : {A B : Obj} {S : Mor A B} → S ↑₀ ≈ (S ˘ / ∈ ˘) ≬ ∈
↑≈≬ {A} {B} {S} = ≈-begin
  S ↑₀
  ≈⟨⟩
    Λ₀ (∈ \ S)
  ≈⟨⟩
    (∈ \ S) ˘ ≬ ∈
  ≈⟨ ≬-cong₁ \-˘ ⟩
    (S ˘ / ∈ ˘) ≬ ∈
  □


↓↑≈≬ : {A B : Obj} {S : Mor A B} → S ↓↑₀ ≈ (S ˘ / (∈ \ S ˘)) ≬ ∈
↓↑≈≬ {A} {B} {S} = ≈-begin
    S ↓↑₀
  ≈⟨ ⨾-cong₂ ↑≈≬ ⟩
    S ↓₀ ⨾ ((S ˘ / ∈ ˘) ≬ ∈)
  ≈⟨ ≬-in-left Λ-isMapping ⟩
    ((S ˘ / ∈ ˘) ⨾ S ↓₀ ˘) ≬ ∈
  ≈⟨ ≬-cong₁ (/-inner-⨾ Λ-isMapping) ⟩
    (S ˘ / (S ↓₀ ⨾ ∈ ˘)) ≬ ∈
  ≈⟨ ≬-cong₁ (/-cong₂ ↓⨾∈˘) ⟩
    (S ˘ / (∈ \ S ˘)) ≬ ∈
  □


↑↓≈≬ : {A B : Obj} {S : Mor A B} → S ↑↓₀ ≈ (S / (∈ \ S)) ≬ ∈
↑↓≈≬ {A} {B} {S} = ≈-begin
    S ↑↓₀
  ≈⟨ ⨾-cong₂ ↓≈≬ ⟩
    S ↑₀ ⨾ ((S / ∈ ˘) ≬ ∈)
  ≈⟨ ≬-in-left Λ-isMapping ⟩
    ((S / ∈ ˘) ⨾ S ↑₀ ˘) ≬ ∈
  ≈⟨ ≬-cong₁ (/-inner-⨾ Λ-isMapping) ⟩
    (S / (S ↑₀ ⨾ ∈ ˘)) ≬ ∈
  ≈⟨ ≬-cong₁ (/-cong₂ ↑⨾∈˘) ⟩
    (S / (∈ \ S)) ≬ ∈
  □


↑-↓˘≈≬ : {A : Obj} {S : Mor A A} → S ↑₀ ⨾ (S ↓₀) ˘ ≈ (S ˘ / ∈ ˘) ≬ (S / ∈ ˘)
↑-↓˘≈≬ {A} {S} = ≈-begin
    S ↑₀ ⨾ (S ↓₀) ˘
  ≈⟨ ⨾-cong₁ ↑≈≬ ⟩
    ((S ˘ / ∈ ˘) ≬ ∈) ⨾ (S ↓₀) ˘
  ≈⟨ ≬-M-in-right Λ-isMapping ⟩
    (S ˘ / ∈ ˘) ≬ (∈ ⨾ (S ↓₀) ˘)
```

```
  ≈⟨ ⨉-cong₂ ∊⨾Λ˘ ⟩
    (S ˘ / ∊ ˘) ⨉ (∊ \ S ˘) ˘
  ≈⟨ ⨉-cong₂ \˘-˘ ⟩
    (S ˘ / ∊ ˘) ⨉ (S / ∊ ˘)
  □
```

```
ℙlbd-⨾↓↑-˘ : {A B C : Obj} {R : Mor A (ℙ B)} {S : Mor C B}
            → ℙ.lbd (R ⨾ S ↓↑₀) ˘ ≈ ((∊ \ S ˘) / (∊ \ S ˘)) / R
ℙlbd-⨾↓↑-˘ {A} {B} {C} {R} {S} = ≈-begin
    ℙ.lbd (R ⨾ S ↓↑₀) ˘
  ≈⟨ ˘\˘-˘ ⟩
    Ω / (R ⨾ S ↓↑₀)
  ≈⟨ /-flip (Mapping.prf (S ↓↑)) ⟩
    (Ω ⨾ S ↓↑₀ ˘) / R
  ≈⟨ /-cong₁ Ω-⨾-↓↑˘ ⟩
    ((∊ \ S ˘) / (∊ \ S ˘)) / R
  □
```

```
↓↑-monotone : {A B : Obj} {S : Mor A B} → Ω ⨾ S ↓↑₀ ⊑ S ↓↑₀ ⨾ Ω
↓↑-monotone {A} {B} {S} = ⊑-begin
    Ω ⨾ S ↓↑₀
  ⊑⟨ \-universal (⊑-begin
        (S ˘ / (∊ \ S ˘)) ⨾ (∊ \ ∊) ⨾ S ↓↑₀
      ⊑⟨ ⨾-assocL₃₊₁ ⟨≈⊑⟩ (⨾-monotone₁ (⊑-begin
          (S ˘ / (∊ \ S ˘)) ⨾ (∊ \ ∊) ⨾ S ↓₀
        ⊑⟨ /-universal (⊑-begin
            ((S ˘ / (∊ \ S ˘)) ⨾ (∊ \ ∊) ⨾ S ↓₀) ⨾ ∊ ˘
          ≈⟨ ⨾-assoc₃₊₁ ⟨≈≈⟩ ⨾-cong₂₂ ↓⨾∊˘ ⟩
            (S ˘ / (∊ \ S ˘)) ⨾ (∊ \ ∊) ⨾ (∊ \ S ˘)
          ⊑⟨ ⨾-monotone₂ (⊑-begin
              (∊ \ ∊) ⨾ (∊ \ S ˘)
            ⊑⟨ \-cancel-middle ⟩
              ∊ \ S ˘
            □) ⟨⊑⊑⟩ /-cancel-outer ⟩
            S ˘
          □) ⟩
          S ˘ / ∊ ˘
        ≈˘⟨ \-˘ ⟩
          (∊ \ S) ˘
        □)⟨⊑⊑⟩ ⨉-cancel-left) ⟩
        ∊
      □) ⟩
    (S ˘ / (∊ \ S ˘)) \ ∊
  ≈˘⟨ \-cong₁ (\-˘ ⟨≈≈⟩ /-cong₂ /˘-˘) ⟩
    ((S / ∊ ˘) \ S) ˘ \ ∊
  ≈˘⟨ \-cong₁ (˘-cong ↓↑⨾∊˘) ⟩
    (S ↓↑₀ ⨾ ∊ ˘) ˘ \ ∊
  ≈⟨ \-cong₁ ˘-involutionRightConv ⟨≈≈˘⟩ \-inner-⨾ (Mapping.prf (S ↓↑)) ⟩
    S ↓↑₀ ⨾ (∊ \ ∊)
  ≈⟨⟩
    S ↓↑₀ ⨾ Ω
  □
```

Trying a different presentation:

```
↓↑-monotone′ : {A B : Obj} {S : Mor A B} → Ω ⨾ S ↓↑₀ ⊑ S ↓↑₀ ⨾ Ω
↓↑-monotone′ {A} {B} {S} = ⊑-begin
    Ω ⨾ S ↓↑₀
  ⊑⟨ \-universal (⊑-begin
        (S ˘ / (∊ \ S ˘)) ⨾ (∊ \ ∊) ⨾ S ↓₀ ⨾ S ↑₀
      ≈⟨ ⨾-assocL₃₊₁ ⟨≈≈⟩ ⨾-cong₂ ↑≈⨉ ⟩
        ((S ˘ / (∊ \ S ˘)) ⨾ (∊ \ ∊) ⨾ S ↓₀) ⨾ ((S ˘ / ∊ ˘) ⨉ ∊)
      ⊑⟨ ⨾-monotone₁ (/-universal (⊑-begin
          ((S ˘ / (∊ \ S ˘)) ⨾ (∊ \ ∊) ⨾ S ↓₀) ⨾ ∊ ˘
        ≈⟨ ⨾-assoc₃₊₁ ⟨≈≈⟩ ⨾-cong₂₂ ↓⨾∊˘ ⟩
          (S ˘ / (∊ \ S ˘)) ⨾ (∊ \ ∊) ⨾ (∊ \ S ˘)
        ⊑⟨ ⨾-monotone₂ \-cancel-middle ⟩
          (S ˘ / (∊ \ S ˘)) ⨾ (∊ \ S ˘)
        ⊑⟨ /-cancel-outer ⟩
          S ˘
        □)) ⟩
        (S ˘ / ∊ ˘) ⨾ ((S ˘ / ∊ ˘) ⨉ ∊)
      ⊑⟨ ⨉-cancel-left ⟩
        ∊
      □) ⟩
    (S ˘ / (∊ \ S ˘)) \ ∊
  ≈˘⟨ \-cong₁ (\-˘ ⟨≈≈⟩ /-cong₂ /˘-˘) ⟩
    ((S / ∊ ˘) \ S) ˘ \ ∊
  ≈˘⟨ \-cong₁ (˘-cong ↓↑⨾∊˘) ⟩
    (S ↓↑₀ ⨾ ∊ ˘) ˘ \ ∊
  ≈⟨ \-cong₁ ˘-involutionRightConv ⟨≈≈˘⟩ \-inner-⨾ (Mapping.prf (S ↓↑)) ⟩
    S ↓↑₀ ⨾ (∊ \ ∊)
  □
```

```
ℙglb-preserves-↓↑ :   {A B C : Obj} {R : Mor A (ℙ B)} {S : Mor C B}
  → R ⨾ S ↓↑₀ ≈ R
  → ℙ.glb R ⊑ ℙ.glb R ⨾ (S ↓↑₀) ˘ ⨾ S ↓↑₀
ℙglb-preserves-↓↑ {A} {B} {C} {R} {S} R-closed = glb-closed′ ℙ.glbΩ-total R-closed
  where
    open PreGaloisConnection occ {ℙ B} {ℙ C} {Ω} {Ω ˘} {Ω-isPreorder} {Ω˘-isPreorder} {S ↓} {S ↑}
      (record {gc = Galois-↓-↑-˘}) using (closure)
    open ClosureOp {ℙ B} {Ω} {Ω-isOrder} {S ↓↑} (record {char = ≈-sym closure})
```

6.7 Categoric.OCC.DirectPower.PolaritiesGC

This module reproduces much of the content of Categoric.OCC.DirectPower.Polarities (Sect. 6.6) using the internal Galois connection between R ↓ and R ↑ that is induced by any R. Since the resulting proof terms are much larger than those of the "manual" proofs in Categoric.OCC.DirectPower.Polarities (Sect. 6.6), it currently is recommended to base further developments on Categoric.OCC.DirectPower.Polarities instead of on this module.

```
module Categoric.OCC.DirectPower.PolaritiesGC {i j k₁ k₂} {Obj : Set i} (occ : OCC j k₁ k₂ Obj)
  (let open OCC occ)
  (leftResOp   : LeftResOp orderedSemigroupoid)
  (rightResOp  : RightResOp orderedSemigroupoid)
  (syqOp       : SyqOp osgc)
  (let open OCC-DirectPower occ leftResOp rightResOp syqOp)
  (directPower : DirectPower)
  where
open DirectPower directPower using
  (ℙ; Ω; Ω˘; Ω-isOrder; Ω˘-isOrder; Ω-isPreorder; Ω˘-isPreorder; powerOp)
open import Categoric.OSGC.Power.Polarities osgc leftResOp rightResOp powerOp
```

Now that we have access to the connection, let us open our modules and only name some of the relevant results — to avoid name clashes we prime some names.

```
module _ {A B : Obj} {R : Mor A B} where
  open import Categoric.OCC.Preorder.Galois
  open PreGaloisConnection occ {ℙ A} {ℙ B} {Ω} {Ω ˘} {Ω-isPreorder} {Ω˘-isPreorder} {R ↑} {R ↓}
    (record {gc = ≈-sym Galois-↓-↑}) public renaming
      (gc˘ to      Galois-↓-↑-˘₀              -- R ↓₀ ⨾ Ω ˘ ≈ Ω ˘ ˘ ⨾ (R ↑₀) ˘
          -- gc ≐ ≈-sym Galois-↓-↑ ≐ R ↑₀ ⨾ Ω ˘ ≈ Ω ⨾ (R ↓₀) ˘
      ; LU-⊑-E to ↑↓-⊑-Ω                 -- : R ↑↓₀ ⊑ Ω
      ; UL-⊑-E˘ to ↓↑-⊑-Ω˘˘              -- : R ↓₀ ⨾ R ↑₀ ⊑ Ω ˘ ˘
      ; L-absE to ↑↓-abs-Ω
          -- : ∀ {Q S} → S ⨾ R ↑↓₀ ≈ Q ⨾ R ↑↓₀ → S ⨾ R ↑₀ ⨾ Ω ˘ ≈ Q ⨾ R ↑₀ ⨾ Ω ˘
      ; U-absE˘ to ↓↑abs-Ω˘
          -- : ∀ {Q S} → S ⨾ R ↓↑₀ ≈ Q ⨾ R ↓↑₀ → S ⨾ R ↓₀ ⨾ Ω ˘ ≈ Q ⨾ R ↓₀ ⨾ Ω ˘
      ; interior to ↓↑-interior           -- : R ↓↑₀ ⨾ Ω ˘ ⨾ R ↓↑₀ ˘ ≈ R ↓↑₀ ⨾ Ω ˘
      ; closure to ↑↓-closure             -- : R ↑↓₀ ⨾ Ω ⨾ R ↑↓₀ ˘ ≈ Ω ⨾ R ↑↓₀ ˘
      ; UL-idempE to ↓↑-idempΩ˘˘          -- : R ↓↑₀ ⨾ R ↓↑₀ ⨾ Ω ˘ ˘ ≈ R ↓↑₀ ⨾ Ω ˘ ˘
      ; LU-idempE to ↑↓-idempΩ            -- : R ↑↓₀ ⨾ R ↑↓₀ ⨾ Ω ≈ R ↑↓₀ ⨾ Ω
      ; LULE≈LE to ↑↓↑-semi-Ω˘            -- : (R ↑₀ ⨾ R ↓↑₀) ⨾ Ω ˘ ≈ R ↑₀ ⨾ Ω ˘
      ; ULUE˘≈UE˘ to ↓↑↓-semi-Ω˘          -- : (R ↓₀ ⨾ R ↑↓₀) ⨾ Ω ˘ ≈ R ↓₀ ⨾ Ω ˘
      ; L-monotone     to ↑-antitone      -- : Ω ⨾ R ↑₀ ⊑ R ↑₀ ⨾ Ω ˘
      ; U-monotone     to ↓-antitone      -- : Ω ˘ ⨾ R ↓₀ ⊑ R ↓₀ ⨾ Ω
      ; L-isotone-on-U to ↑-isotone-on-↓  -- : R ↓₀ ⨾ R ↑₀ ⨾ Ω ˘ ⨾ R ↑₀ ˘ ⨾ R ↓₀ ˘ ≈ R ↓₀ ⨾ Ω ⨾ R ↓₀ ˘
      ; U-isotone-on-L to ↓-isotone-on-↑  -- : R ↑₀ ⨾ R ↓₀ ⨾ Ω ⨾ R ↓₀ ˘ ⨾ R ↑₀ ˘ ≈ R ↑₀ ⨾ Ω ˘ ⨾ R ↑₀ ˘
      ; UL-comonotone to ↓↑-monotone₀     -- : Ω ˘ ˘ ⨾ R ↓↑₀ ⊑ R ↓↑₀ ⨾ Ω ˘ ˘
      ; LU-monotone    to ↑↓-monotone     -- : Ω ⨾ R ↑↓₀ ⊑ R ↑↓₀ ⨾ Ω
      )
```

Now we turn to those derivable from the partial order properties,

```
open import Categoric.OCC.Order.Galois occ leftResOp rightResOp syqOp
open GaloisConnection {ℙ A} {ℙ B} {Ω} {Ω ˘} {Ω-isOrder} {Ω˘-isOrder} {R ↑} {R ↓}
  (record {gc = ≈-sym Galois-↓-↑}) public using () renaming
  (L-map-absorption to ↑-map-absorption
      -- : ∀ {Q S} → isMapping Q → isMapping S → S ⨾ R ↑↓₀ ≈ Q ⨾ R ↑↓₀ → S ⨾ R ↑₀ ≈ Q ⨾ R ↑₀
  ; U-map-absorption to ↓-map-absorption
      -- : ∀ {Q S} → isMapping Q → isMapping S → S ⨾ R ↓↑₀ ≈ Q ⨾ R ↓↑₀ → S ⨾ R ↓₀ ≈ Q ⨾ R ↓₀
  ; UL-idempot to ↓↑-idempot   -- : R ↓↑ ⨾₁ R ↓↑ ≈₁ R ↓↑
  ; LU-idempot to ↑↓-idempot   -- : R ↑↓ ⨾₁ R ↑↓ ≈₁ R ↑↓
  ; L-semi-inverse to ↑↓↑≈↑     -- : R ↑₀ ⨾ R ↓₀ ⨾ R ↑₀ ≈ R ↑₀
  ; U-semi-inverse to ↓↑↓≈↓     -- : R ↓₀ ⨾ R ↑₀ ⨾ R ↓₀ ≈ R ↓₀
  ; UL-ranClosed-← to ↓↑-ranClosed-←′   -- : ∀ {S} → S ⨾ R ↓↑₀ ≈ S → S ⊑ S ⨾ R ↓↑₀ ˘ ⨾ R ↓↑₀
  ; UL-ranClosed-→ to ↓↑-ranClosed-→′   -- : ∀ {S} → S ⊑ S ⨾ R ↓↑₀ ˘ ⨾ R ↓↑₀ → S ⨾ R ↓↑₀ ≈ S
  ; LU-ranClosed-← to ↑↓-ranClosed-←    -- : ∀ {S} → S ⨾ R ↑↓₀ ≈ S → S ⊑ S ⨾ R ↑↓₀ ˘ ⨾ R ↑↓₀
  ; LU-ranClosed-→ to ↑↓-ranClosed-→    -- : ∀ {S} → S ⊑ S ⨾ R ↑↓₀ ˘ ⨾ R ↑↓₀ → S ⨾ R ↑↓₀ ≈ S
  ; UL-lub-closed-⊑ to ↓↑-lub-closed-⊑  -- : ∀ {S} → S ⨾ R ↓↑₀ ≈ S → lub S ⨾ R ↓↑₀ ⊑ lub S
  ; LU-glb-closed-⊑ to ↑↓-glb-closed-⊑  -- : ∀ {S} → S ⨾ R ↑↓₀ ≈ S → glb S ⨾ R ↑↓₀ ⊑ glb S
  )
```

Besides some occurecnes of double converses, which can be cheaply eliminated, we shall present a table comparing the costs. This would be of use to those whose interests lie in efficiency or compiler design.

Interestingly, saving the pretty-printed normal form to a file and compressing it with xz -9 yields roughly as many bytes as the number of lines in that file. These line numbers are therefore already a reasonably proxy measure for the complexity of the generated terms. We also include the rough CPU time required for interactive Agda-2.4.2.3 (from 2015-03-21) to perform the respective normalisation on a (6-core) 2.8GHz AMD Phenom with 16GB of RAM, running with 10GB of Haskell heap:

name	direct proof			via Galois connection		
	lines	xz	min	lines	xz	min
↓↑-monotone	13902	14904	8	653203	861160	280
↑↓⊑Ω :	3171	5000	2.5	7691	10744	3
↓↑↓-semi	27231	26716	19	49274	46396	40
↓↑-idemp	27860	28552	18	336908	324960	171
↓↑-ranClosed-→	62026	61316	25	-	-	> 60

Besides the costs, notice that many theorems fall out of the connection; above we also included a few that were not needed in (Kahl, 2014a). However, with our modules in hand, such results can now immediately be instantiated and thus save time.

6.8 Categoric.OCC.DirectPower.OrderPolarities

Here we collect properties of the polarities ≼ ↑ amd ≼ ↓ of an order relation ≼.

```
module Categoric.OCC.DirectPower.OrderPolarities {i j k₁ k₂} {Obj : Set i} (occ : OCC j k₁ k₂ Obj)
  (let open OCC occ)
  (leftResOp  : LeftResOp orderedSemigroupoid)
  (rightResOp : RightResOp orderedSemigroupoid)
  (syqOp      : SyqOp osgc)
  (let open OCC-DirectPower occ leftResOp rightResOp syqOp)
  (directPower : DirectPower)
  where
private
  module ℙ = DirectPower directPower
open ℙ using (ℙ; ∈; Ω; Ω˘; powerOp; Λ; Λ₀; Λ-isMapping; Λ-cong; Λ⨾Ω˘; ∈⇒Λ; Ω˘-isPreorder₀; ∈⨾-∈\)
open SyqOp                                                        syqOp
open OCC-SyQ-Props           occ                                  syqOp
open SyQ-ResidualProps       osgc             leftResOp rightResOp syqOp
open ResidualOps                              leftResOp rightResOp
open OrdCat-Residual-Props orderedCategory   leftResOp rightResOp
open OSGC-Residuals          osgc             leftResOp rightResOp
open import Categoric.OCC.Order occ leftResOp rightResOp syqOp using (IsOrder; module IsOrder)
open PowerOp osgc powerOp using (Λ-⨾∈˘; Λ⨾∈˘; ∈⨾Λ˘; map-Λ)
open Category₁ (MapCat occ)
open import Categoric.OSGC.PowerOrder      osgc leftResOp rightResOp powerOp using ()
open import Categoric.OSGC.Power.Polarities osgc leftResOp rightResOp powerOp
open import Categoric.OCC.DirectPower.Polarities  occ leftResOp rightResOp syqOp directPower

module OrderPolarities {A : Obj} {≼ : Mor A A} (≼-isOrder : IsOrder ≼) where
  open IsOrder ≼-isOrder renaming
    (refl to ≼-refl; trans to ≼-trans; antisym≈ to ≼⩚≼; ˘-antisym≈ to ≽⩚≽)

  ≼↓⨾∈˘ : ≼ ↓₀ ⨾ ∈ ˘ ≈ lbd (∈ ˘)
  ≼↓⨾∈˘ = ≈-begin
      ≼ ↓₀    ⨾ ∈ ˘
    ≈⟨⟩
      Λ₀ (∈ \ ≼ ˘) ⨾ ∈ ˘
```

```
    ≈⟨ Λ⨾∈˘  ⟩
      ∈ \ ≼ ˘
    ≈˘⟨ \-cong₁ ˘˘ ⟩
      lbd (∈ ˘)
    □

  ≼↑⨾∈˘ : ≼ ↑₀ ⨾ ∈ ˘ ≈ ubd (∈ ˘)
  ≼↑⨾∈˘ = ≈-begin
      ≼ ↑₀    ⨾ ∈ ˘
    ≈⟨ Λ⨾∈˘ ⟩
      ∈ \ ≼
    ≈˘⟨ ubd-˘ ⟩
      ubd (∈ ˘)
    □

  ≼↑↓⨾∈˘ : ≼ ↑↓₀ ⨾ ∈ ˘ ≈ lbd (ubd (∈ ˘))
  ≼↑↓⨾∈˘ = ≈-begin
      (≼ ↑₀   ⨾ ≼ ↓₀) ⨾ ∈ ˘
    ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ≼↓⨾∈˘ ⟩
      ≼ ↑₀    ⨾ lbd (∈ ˘)
    ≈⟨ Mapping-⨾-lbd ℙ.Λ-isMapping ⟩
      lbd (≼ ↑₀  ⨾ ∈ ˘)
    ≈⟨ lbd-cong ≼↑⨾∈˘ ⟩
      lbd (ubd (∈ ˘))
    □

  ≼↓↑⨾∈˘ : ≼ ↓↑₀ ⨾ ∈ ˘ ≈ ubd (lbd (∈ ˘))
  ≼↓↑⨾∈˘ = ≈-begin
      (≼ ↓₀   ⨾ ≼ ↑₀) ⨾ ∈ ˘
    ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ≼↑⨾∈˘ ⟩
      ≼ ↓₀    ⨾ ubd (∈ ˘)
    ≈⟨ Mapping-⨾-ubd ℙ.Λ-isMapping ⟩
      ubd (≼ ↓₀ ⨾ ∈ ˘)
    ≈⟨ ubd-cong ≼↓⨾∈˘ ⟩
      ubd (lbd (∈ ˘))
    □

  ≼↑↓≈ΛlbdUbd∈˘ : ≼ ↑↓₀ ≈ Λ₀ (lbd (ubd (∈ ˘)))
  ≼↑↓≈ΛlbdUbd∈˘ = Λ-⨾∈˘ {f = ≼ ↑↓} ⟨≈˘≈⟩ Λ-cong ≼↑↓⨾∈˘

  ≼↓↑≈ΛubdLbd∈˘ : ≼ ↓↑₀ ≈ Λ₀ (ubd (lbd (∈ ˘)))
  ≼↓↑≈ΛubdLbd∈˘ = Λ-⨾∈˘ {f = ≼ ↓↑} ⟨≈˘≈⟩ Λ-cong ≼↓↑⨾∈˘

ΛubdLbd∈˘-⨾-lbd∈˘ : Λ₀ (ubd (lbd (∈ ˘))) ⨾ lbd (∈ ˘) ≈ lbd (∈ ˘)
ΛubdLbd∈˘-⨾-lbd∈˘ = ≈-begin
    Λ₀ (ubd (lbd (∈ ˘))) ⨾ lbd (∈ ˘)
  ≈⟨⟩
    Λ₀ (ubd (lbd (∈ ˘))) ⨾ ((∈ ˘) ˘ \ ≼ ˘)
  ≈⟨ \-inner-⨾ ℙ.Λ-isMapping ⟨≈≈˘⟩ \-cong₁ ˘-involution ⟩
    (Λ₀ (ubd (lbd (∈ ˘))) ⨾ ∈ ˘) ˘ \ ≼ ˘
  ≈⟨ \-cong₁ (˘-cong Λ⨾∈˘) ⟩
    (ubd (lbd (∈ ˘))) ˘ \ ≼ ˘
  ≈⟨ lbd-ubd-lbd ⟩
    lbd (∈ ˘)
  □
```

```
downset : Mor A (ℙ A)
downset = Λ₀ (≼ ˘)
downset-isInjectiveI : isInjectiveI downset
downset-isInjectiveI = ⨾-cong₂ ╳-˘ ⟨≈⊑⟩ ╳-cancel-middle ⟨⊑≈⟩ ╳-cong ˘˘ ˘˘ ⟨⊑≈⟩ ≼╳≼
downset-isInjective : isInjective downset
downset-isInjective = isInjective-from-I downset-isInjectiveI

upset : Mor A (ℙ A)
upset = Λ₀ ≼

down-up-nat : downset ⨾ ∈ ˘ ≈ ∈ ⨾ upset ˘
down-up-nat = Λ⨾∈˘ ⟨≈≈˘⟩ ∈⨾Λ˘
```

The least upper bound of the downset of x is x. Intuitively, this requires antisymmetry, so we culminate in ≽╳≽.

```
  downset-⨾-lub∈˘-⊑ : downset ⨾ lub (∈ ˘) ⊑ Id
  downset-⨾-lub∈˘-⊑ = ⊑-begin
        downset ⨾ lub (∈ ˘)
      ≈⟨⟩
        Λ₀ (≼ ˘) ⨾ lub (∈ ˘)
      ≈⟨⟩
        ((≼ ˘) ˘ ╳ ∈) ⨾ (ubd (∈ ˘) ˘ ╳ ≼ ˘)
      ≈⟨ ⨾-cong (╳-cong₁ ˘˘) (╳-cong₁ ˘\-˘) ⟩
        (≼ ╳ ∈) ⨾ ((≼ ˘ / ∈ ˘) ╳ ≼ ˘)
      ⊑⟨ ˘╳˘-universal
          (⊑-begin
            ≼ ˘ ⨾ (≼ ╳ ∈) ⨾ ((≼ ˘ / ∈ ˘) ╳ ≼ ˘)
          ⊑⟨ ⨾-monotone₂₁ ╳-⊑-/ ⟩
            ≼ ˘ ⨾ (≼ ˘ / ∈ ˘) ⨾ ((≼ ˘ / ∈ ˘) ╳ ≼ ˘)
          ⊑⟨ ⨾-monotone₂ ╳-cancel-left ⟩
            ≼ ˘ ⨾ ≼ ˘
          ⊑⟨ ˘-trans ⟩
            ≼ ˘
          □)
          (⊑-begin
            ((≼ ╳ ∈) ⨾ ((≼ ˘ / ∈ ˘) ╳ ≼ ˘)) ⨾ ≼
          ⊑⟨ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ ╳-cancel-right˘ ⟩
            (≼ ╳ ∈) ⨾ (≼ ˘ / ∈ ˘) ˘
          ≈⟨ ⨾-cong₂ ˘/˘-˘ ⟩
            (≼ ╳ ∈) ⨾ (∈ \ ≼)
          ⊑⟨ ⨾-monotone₁ ╳-⊑-\ ⟨⊑⊑⟩ \-cancel-middle ⟩
            ≼ \ ≼
          ≈⟨ order-\ ⟩
            ≼
          □)
        ⟩
        ≼ ˘ ╳ ≼ ˘
      ≈⟨ ≽╳≽ ⟩
        Id
      □

  meet : Mor (ℙ A) A
  meet = glb (∈ ˘)

  ≼↓↑-⨾-glb∈˘ :  ≼ ↓↑₀ ⨾ meet ≈ meet
  ≼↓↑-⨾-glb∈˘ = ≈-begin
      ≼ ↓↑₀ ⨾ meet
    ≈⟨⟩
      ≼ ↓↑₀ ⨾ (lbd (∈ ˘) ˘ ╳ ≼)
    ≈⟨ ╳-in-left (Mapping.prf (≼ ↓↑)) ⟨≈≈˘⟩ ╳-cong₁ ˘-involution ⟩
      (≼ ↓↑₀ ⨾ lbd (∈ ˘)) ˘ ╳ ≼
```

```
    ≈⟨ ⨉-cong₁ (˘-cong (Mapping-⨾-lbd (Mapping.prf (≼ ↓↑)))) ⟩
      (lbd (≼ ↓↑₀ ⨾ ∈ ˘)) ˘ ⨉ ≼
    ≈⟨   ⨉-cong₁ (˘-cong (lbd-cong ≼↓↑⨾∈˘)) ⟩
      (lbd (ubd (lbd (∈ ˘)))) ˘ ⨉ ≼
    ≈⟨   ⨉-cong₁ (˘-cong lbd-ubd-lbd) ⟩
      lbd (∈ ˘) ˘ ⨉ ≼
    ≈⟨⟩
      meet
    □

  ΛubdLbd∈˘-⨾-glb∈˘ : Λ₀ (ubd (lbd (∈ ˘))) ⨾ meet ≈ meet
  ΛubdLbd∈˘-⨾-glb∈˘ = ≈-begin
      Λ₀ (ubd (lbd (∈ ˘))) ⨾ meet
    ≈⟨⟩
      Λ₀ (ubd (lbd (∈ ˘))) ⨾ (lbd (∈ ˘) ˘ ⨉ ≼)
    ≈⟨ ⨉-in-left ℙ.Λ-isMapping ⟨≈≈˘⟩ ⨉-cong₁ ˘-involution ⟩
      (Λ₀ (ubd (lbd (∈ ˘))) ⨾ lbd (∈ ˘)) ˘ ⨉ ≼
    ≈⟨   ⨉-cong₁ (˘-cong (ΛubdLbd∈˘-⨾-lbd∈˘)) ⟩
      lbd (∈ ˘) ˘ ⨉ ≼
    ≈⟨⟩
      meet
    □

module Complete (lub-isMapping : {I : Obj} {R : Mor I A} → isMapping (lub R))
                (glb-isMapping : {I : Obj} {R : Mor I A} → isMapping (glb R)) where
  private
    lub-Mapping : {I : Obj} → Mor I A → Mapping I A
    lub-Mapping R = record {mor = lub R; prf = lub-isMapping}
    glb-Mapping : {I : Obj} → Mor I A → Mapping I A
    glb-Mapping R = record {mor = glb R; prf = glb-isMapping}
    lub-total : {I : Obj} {R : Mor I A} → isTotal (lub R)
    lub-total = proj₂ lub-isMapping
    glb-total : {I : Obj} {R : Mor I A} → isTotal (glb R)
    glb-total = proj₂ glb-isMapping

    downset-char : downset ⨾ Ω ˘ ≈ ≼ ˘ ⨾ lub (∈ ˘) ˘
    downset-char = ≈-begin
          Λ₀ (≼ ˘) ⨾ Ω ˘
        ≈⟨ Λ⨾Ω˘ ⟩
          ≼ ˘ / ∈ ˘
        ≈˘⟨ ˘-cong (\-cong₁ ˘˘) ⟨≈≈⟩ \-˘ ⟩
          ubd (∈ ˘) ˘
        ≈˘⟨ ˘-cong (total-lub-⨾-order lub-total) ⟩
          (lub (∈ ˘) ⨾ ≼) ˘
        ≈⟨ ˘-involution ⟩
          ≼ ˘ ⨾ lub (∈ ˘) ˘
        □

    open import Categoric.OSGC.Preorder.Galois
    open PreGaloisConnection osgc
      {A} {ℙ A} {≼ ˘} {Ω ˘} {˘-isPreorder₀} {Ω˘-isPreorder₀} {Λ (≼ ˘)} {lub-Mapping (∈ ˘)}
      (record {gc = downset-char}) using () renaming (LULE≈LE to downset-semi)

    downset-⨾-lub∈˘ : downset ⨾ lub (∈ ˘) ≈ Id
    downset-⨾-lub∈˘ = total⊑unival-≈ (⨾-isTotal ℙ.∈-comprehensive lub-total)
                                     idUnivalent
                                     downset-⨾-lub∈˘-⊑

    downset-semi-inverse : {Z : Obj} {R : Mor A Z} → downset ⨾ lub (∈ ˘) ⨾ R ≈ R
    downset-semi-inverse = ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ downset-⨾-lub∈˘ ⟨≈≈⟩ leftId

    glb⨾downset : {I : Obj} {R : Mor I A} → glb R ⨾ downset ≈ Λ₀ (lbd R)
    glb⨾downset {I} {R} = ≈-begin
        glb R ⨾ downset
      ≈⟨⟩
        glb R ⨾ Λ₀ (≼ ˘)
      ≈⟨ map-Λ {f = glb-Mapping R} ⟩
        Λ₀ (glb R ⨾ ≼ ˘)
      ≈⟨ Λ-cong (total-glb-⨾-order˘ glb-total) ⟩
        Λ₀ (lbd R)
      □

    lemma₀ : {I : Obj} {R : Mor I A} → isMapping R → (≼ ⨾ R ˘) ↓₀ ≈ Λ₀ (lbd (∈ ˘ ⨾ R))
    lemma₀ {I} {R} R-isMapping = ≈-begin
        (≼ ⨾ R ˘) ↓₀
      ≈⟨ Λ-cong (\-cong₂ ˘-involutionRightConv) ⟩
        Λ₀ (∈ \ (R ⨾ ≼ ˘))
      ≈⟨   Λ-cong (\-flip-M R-isMapping ⟨≈˘≈˘⟩ \-cong₁ ˘-involutionLeftConv) ⟩
        Λ₀ (lbd (∈ ˘ ⨾ R))
      □
```

Moshier mentions (in his notation) ≤↓ <↑ A = ↓ ∨ A before Lemma2.2; the proof requires totality of lub:

```
    ≼↑↓-≈-lub∋⨾downset : ≼ ↑↓₀ ≈ lub (∈ ˘) ⨾ downset
    ≼↑↓-≈-lub∋⨾downset = ≈-begin
            ≼ ↑₀ ⨾ ≼ ↓₀
          ≈⟨ ≼↑↓≈ΛlbdUbd∈˘ ⟩
            Λ₀ (lbd (ubd (∈ ˘)))
        ≈⟨⟩
            Λ₀ (ubd (∈ ˘) ˘ \ ≼ ˘)
        ≈⟨   Λ-cong (⊑-antisym (⊑-begin
              ubd (∈ ˘) ˘ \ ≼ ˘
            ⊑⟨ proj₁ lub-total ⟨⊑≈⟩ ⨾-assoc ⟩
              lub (∈ ˘) ⨾ lub (∈ ˘) ˘ ⨾ (ubd (∈ ˘) ˘ \ ≼ ˘)
            ⊑⟨ ⨾-monotone₂₁ (⨉-˘ ⟨≈⊑⟩ ⨉-⊑-\) ⟩
              lub (∈ ˘) ⨾ (≼ ˘ \ ubd (∈ ˘) ˘) ⨾ (ubd (∈ ˘) ˘ \ ≼ ˘)
            ⊑⟨ ⨾-monotone₂ \-cancel-middle ⟩
              lub (∈ ˘) ⨾ (≼ ˘ \ ≼ ˘)
            ≈⟨ ⨾-cong₂ order˘-\ ⟩
              lub (∈ ˘) ⨾ ≼ ˘
            □)
            (\-universal (⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ⨉-cancel-left ⟨⊑⊑⟩ ˘-trans))) ⟩
            Λ₀ ((ubd (∈ ˘) ˘ ⨉ ≼ ˘) ⨾ ≼ ˘)
        ≈⟨⟩
            Λ₀ (lub (∈ ˘) ⨾ ≼ ˘)
        ≈˘⟨ map-Λ {f = lub-Mapping (∈ ˘)} ⟩
          lub (∈ ˘) ⨾ Λ₀ (≼ ˘)
        ≈⟨⟩
            lub (∈ ˘) ⨾ downset
        □
```

7 Abstract Contexts

7.1 Data.AContext.InOSGC

```
module Data.AContext.InOSGC {i j k1 k2} {Obj : Set i} (osgc : OSGC j k1 k2 Obj)
                            (leftResOp  : LeftResOp (OSGC.orderedSemigroupoid osgc))
                            (rightResOp : RightResOp (OSGC.orderedSemigroupoid osgc))
                            (powerOp : PowerOp osgc) where
open OSGC osgc
open ResidualOps leftResOp rightResOp
open OSGC-Residuals osgc leftResOp rightResOp
open PowerOp osgc powerOp
open import Categoric.OSGC.PowerOrder osgc leftResOp rightResOp powerOp
  using (Ω; Λ0⨾Ω˘; Lub; Lub-cocontinuous; Glb; Glb-cong)
open import Categoric.OSGC.Power.Polarities osgc leftResOp rightResOp powerOp
private
  module MapSG = Semigroupoid (MapSG osgc)
open Semigroupoid1 (MapSG osgc)
```

We name the necessary conditions for ↓↑↓-Lub-cocontinuous as “source compatibility” respectively “target compatibility”, following Moshier (2013), and show equivalent formulations:

```
SrcCompat : {A B1 B2 : Obj} (X : Mor A B1) (R : Mor A B2) → Set k1
SrcCompat X R = R ↓ ⨾1 X ↑↓ ≈1 R ↓
TrgCompat : {A1 A2 B : Obj} (R : Mor A1 B) (Y : Mor A2 B) → Set k1
TrgCompat R Y = Y ↓↑ ⨾1 R ↓ ≈1 R ↓

SrcCompat-⇒ : {A B1 B2 : Obj} (X : Mor A B1) (R : Mor A B2)
             → SrcCompat X R → X ↑↓0 ⨾ ∊ ˘ ⊑ R ↑↓0 ⨾ ∊ ˘
SrcCompat-⇒ X R srcCompat = ⊑-begin
    X ↑↓0 ⨾ ∊ ˘
  ≈⟨ ↑↓⨾∊˘′ ⟩
    (X ˘ / ∊ ˘) \ (X ˘)
  ⊑⟨ \-antitone (/-antitone ∊˘⊑↑↓⨾∊˘) ⟩
    (X ˘ / (R ↑↓0 ⨾ ∊ ˘)) \ (X ˘)
  ≈˘⟨ \-cong1 (/-inner-⨾ (Mapping.prf (R ↑↓))) ⟩
    ((X ˘ / ∊ ˘) ⨾ R ↑↓0 ˘) \ (X ˘)
  ≈˘⟨ \-inner-⨾ (Mapping.prf (R ↑↓)) ⟩
    R ↑↓0 ⨾ ((X ˘ / ∊ ˘) \ (X ˘))
  ≈˘⟨ ⨾-cong2 ↑↓⨾∊˘′ ⟩
    R ↑↓0 ⨾ X ↑↓0 ⨾ ∊ ˘
  ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong2 (⨾-assocL ⟨≈≈⟩ ⨾-cong1 srcCompat) ⟨≈≈⟩ ⨾-assocL ⟩
    R ↑↓0 ⨾ ∊ ˘
  □
SrcCompat-⇐ : {A B1 B2 : Obj} (X : Mor A B1) (R : Mor A B2)
             → X ↑↓0 ⨾ ∊ ˘ ⊑ R ↑↓0 ⨾ ∊ ˘ → SrcCompat X R
SrcCompat-⇐ X R X▪R = ↓⨾↑↓′ (˘-isotone (↑↓⨾∊˘ ⟨≈˘⊑⟩ X▪R ⟨⊑≈⟩ ↑↓⨾∊˘))

TrgCompat-⇒ : {A1 A2 B : Obj} (R : Mor A1 B) (Y : Mor A2 B)
             → TrgCompat R Y → Y ↓↑0 ⨾ ∊ ˘ ⊑ R ↓↑0 ⨾ ∊ ˘
TrgCompat-⇒ R Y trgCompat = ⊑-begin
    Y ↓↑0 ⨾ ∊ ˘
  ⊑⟨ ⨾-monotone2 ∊˘⊑↓↑⨾∊˘ ⟩
    Y ↓↑0 ⨾ R ↓↑0 ⨾ ∊ ˘
  ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong1 (⨾-assocL ⟨≈≈⟩ ⨾-cong1 trgCompat) ⟩
    R ↓↑0 ⨾ ∊ ˘
  □
TrgCompat-⇐ : {A1 A2 B : Obj} (R : Mor A1 B) (Y : Mor A2 B)
             → Y ↓↑0 ⨾ ∊ ˘ ⊑ R ↓↑0 ⨾ ∊ ˘ → TrgCompat R Y
TrgCompat-⇐ R Y Y▪R = ∊⇒Λ {f = Y ↓↑ ⨾1 R ↓} (≈-begin
    (Y ↓↑0 ⨾ R ↓0) ⨾ ∊ ˘
  ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong2 Λ⨾∊˘ ⟩
    Y ↓↑0 ⨾ (∊ \ R ˘)
  ≈⟨ \-inner-⨾ (Mapping.prf (Y ↓↑)) ⟩
    (∊ ⨾ Y ↓↑0 ˘) \ R ˘
  ≈⟨ \-cong1 (˘-involutionRightConv ⟨≈˘≈⟩ ˘-cong ↓↑⨾∊˘) ⟩
    ((Y / ∊ ˘) \ Y) ˘ \ R ˘
  ≈˘⟨ /-˘ ⟩
    (R / ((Y / ∊ ˘) \ Y)) ˘
  ≈⟨ ˘-cong (T/∘\S∘S/ (↓↑⨾∊˘ ⟨≈˘⊑⟩ Y▪R ⟨⊑≈⟩ ↓↑⨾∊˘)) ⟩
    (R / ∊ ˘) ˘
  ≈⟨ /˘-˘ ⟩
    ∊ \ R ˘
  □)

record AContext : Set (i ⊍ j) where
  field
    ent : Obj          -- “entities”
    att : Obj          -- “attributes”
    inc : Mor ent att  -- “incidence”
```

A context homomorphism, following Moshier (2013) and Jipsen (2012), includes the compatibility properties necessary for ↓↑↓-Lub-cocontinuous.

```
record AContextHom (X Y : AContext) : Set (i ⊍ j ⊍ k1 ⊍ k2) where
  private module X = AContext X
          module Y = AContext Y
  field   mor : Mor X.ent Y.att
          srcCompat : SrcCompat X.inc mor
          trgCompat : TrgCompat mor Y.inc

record AContextHom′ (X Y : AContext) : Set (i ⊍ j ⊍ k1 ⊍ k2) where
  private module X = AContext X
          module Y = AContext Y
  field   mor : Mor X.ent Y.att
          srcCompat : mor ↓ ⨾1 X.inc ↑↓ ≈1 mor ↓
          trgCompat : Y.inc ↓↑ ⨾1 mor ↓ ≈1 mor ↓
```

Context homomorphism equality F ≋ G is defined as the underlying morphism equality F.mor ≈ G.mor:

```
infix 4 _≋_
_≋_ : {X Y : AContext} → AContextHom X Y → AContextHom X Y → Set k1
R ≋ S = AContextHom.mor R ≈ AContextHom.mor S
```

For each context, its incidence defines its identity homomorphism:

```
AContext-Id : {X : AContext} → AContextHom X X
AContext-Id {X} = record {mor = AContext.inc X; srcCompat = ↓⨾↑↓; trgCompat = ↓↑⨾↓}
```

7.2 Data.AContext.InOCC

Abstract Contexts

```
module Data.AContext.InOCC {i j k₁ k₂} {Obj : Set i} (occ : OCC j k₁ k₂ Obj)
                           (leftResOp   : LeftResOp (OCC.orderedSemigroupoid occ))
                           (rightResOp : RightResOp (OCC.orderedSemigroupoid occ))
                           (powerOp : PowerOp (OCC.osgc occ)) where
open OCC occ
open ResidualOps leftResOp rightResOp
open OSGC-Residuals osgc leftResOp rightResOp
open OrdCat-Residual-Props orderedCategory leftResOp rightResOp
open PowerOp osgc powerOp
open import Categoric.OSGC.PowerOrder osgc leftResOp rightResOp powerOp
  using (Lub; Lub-cocontinuous; Glb)
open import Categoric.OSGC.Power.Polarities osgc leftResOp rightResOp powerOp
private
  module MapCat = Category (MapCat occ)
open Category₁ (MapCat occ)
open import Data.AContext.InOSGC osgc leftResOp rightResOp powerOp
```

It turns out that moving from OSGCs to OCCs by adding identities is sufficient for obtaining a partial inverse to the operator _↓.

The key is that Λ Id : Mapping A (ℙ A) can be understood as mapping each "element" a : A to the singleton "set" $\{a\}$: ℙ A.

The "relation" singletons A relates a "subset of A" with all singletons contained in it:

```
singletons : {A : Obj} → Mor (ℙ A) (ℙ A)
singletons = ∈ ˘ ⨾ Λ₀ Id
```

Applying Lub to this produces the identity mapping on ℙ A:

```
Lub-singletons : {A : Obj} → Lub (singletons {A}) ≈₁ Id₁ {ℙ A}
Lub-singletons {A} = ≈₁-begin
    Λ ((∈ ˘ ⨾ Λ₀ Id) ⨾ ∈ ˘)
  ≈₁⟨ Λ-cong (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘) ⟩
    Λ (∈ ˘ ⨾ Id {A})
  ≈₁⟨ Λ-cong (rightId ⟨≈≈˘⟩ leftId) ⟨≈≈⟩ Λ-⨾∈˘ {f = Id₁ {ℙ A}} ⟩
    Id₁ {ℙ A}
  □₁
```

The operator [_] has the opposite type of _↓, and [f] relates a with b if and only if $a \in$ f$\{b\}$:

```
[_] : {A B : Obj} → Mapping (ℙ B) (ℙ A) → Mor A B
[f] = (Λ₀ Id ⨾ Mapping.mor f ⨾ ∈ ˘) ˘

[ ]-cong : {A B : Obj} {f₁ f₂ : Mapping (ℙ B) (ℙ A)} → f₁ ≈₁ f₂ → [f₁] ≈ [f₂]
[ ]-cong f₁≈f₂ = ˘-cong (⨾-cong₂₁ f₁≈f₂)
```

We always have [R ↓] ≈ R:

```
[↓] : {A B : Obj} (R : Mor A B) → [R ↓] ≈ R
[↓] R = ≈-begin
    (Λ₀ Id ⨾ Λ₀ (∈ \ (R ˘)) ⨾ ∈ ˘) ˘
  ≈⟨ ˘-cong (⨾-cong₂ Λ⨾∈˘) ⟩
    (Λ₀ Id ⨾ (∈ \ (R ˘))) ˘
  ≈⟨ ˘-cong (\-inner-⨾ Λ-mapping) ⟩
    ((∈ ⨾ (Λ₀ Id) ˘) \ (R ˘)) ˘
  ≈⟨ \˘-˘ ⟩
    R / ((∈ ⨾ (Λ₀ Id) ˘) ˘)
  ≈⟨ /-cong₂ ˘-involutionRightConv ⟩
    R / (Λ₀ Id ⨾ ∈ ˘)
  ≈⟨ /-cong₂ Λ⨾∈˘ ⟨≈≈⟩ /-Id ⟩
    R
  □
```

For the opposite composition, [f] ↓ ≈₁ f, we need Lub-cocontinuity of f:

```
[ ]↓ : {A B : Obj} (f : Mapping (ℙ B) (ℙ A)) → Lub-cocontinuous f → [f] ↓ ≈₁ f
[ ]↓ f f-cocontinuous = ≈₁-begin
    [f] ↓
  ≈₁⟨ ≈-refl ⟩
    Λ (∈ \ ([f] ˘))
  ≈₁⟨ Λ-cong (\-cong₂ (˘˘ ⟨≈≈⟩ ⨾-assocL)) ⟩
    Λ (∈ \ (Mapping.mor (Λ Id ⨾₁ f) ⨾ ∈ ˘))
  ≈₁˘⟨ Λ-cong (\-cong₂ (⨾-cong₁ ˘˘)) ⟩
    Λ (∈ \ ((Λ₀ Id ⨾ Mapping.mor f) ˘ ˘ ⨾ ∈ ˘))
  ≈₁˘⟨ Λ-cong (\-cong₁ ˘-involutionLeftConv ⟨≈≈⟩ \-flip (˘-isBijective (Mapping.prf (Λ Id ⨾₁ f)))) ⟩
    Λ ((∈ ˘ ⨾ Λ₀ Id ⨾ Mapping.mor f) ˘ \ ∈ ˘)
  ≈₁˘⟨ Λ-cong (\-cong₁ (˘-cong ⨾-assoc)) ⟩
    Glb (singletons ⨾ Mapping.mor f)
  ≈₁˘⟨ f-cocontinuous singletons ⟩
    Lub singletons ⨾₁ f
  ≈₁⟨ ⨾-cong₁ Lub-singletons ⟨≈≈⟩ leftId ⟩
    f
  □₁
```

The last two steps represent the argument of Moshier (2013) that "If f sends unions to intersections, its behavior is determined by its behavior on singletons."

7.3 Data.AContext.Category

```
module Data.AContext.Category {i j k₁ k₂} {Obj : Set i} (occ : OCC j k₁ k₂ Obj)
                              (leftResOp   : LeftResOp (OCC.orderedSemigroupoid occ))
                              (rightResOp : RightResOp (OCC.orderedSemigroupoid occ))
                              (powerOp : PowerOp (OCC.osgc occ)) where
open OCC occ
open ResidualOps leftResOp rightResOp
open OSGC-Residuals osgc leftResOp rightResOp
open OrdCat-Residual-Props orderedCategory leftResOp rightResOp
open PowerOp osgc powerOp
open import Categoric.OSGC.PowerOrder osgc leftResOp rightResOp powerOp
  using (Lub-cocontinuous)
open import Categoric.OSGC.Power.Polarities osgc leftResOp rightResOp powerOp
open Category₁ (MapCat occ)
open import Data.AContext.InOSGC osgc leftResOp rightResOp powerOp
open import Data.AContext.InOCC occ leftResOp rightResOp powerOp
```

We formalise the definition Moshier (2013) gives for composition of AContext homomorphisms, and prove that this gives rise to a category.

```
module AContextHom-Comp {X Y Z : AContext} (F : AContextHom X Y) (G : AContextHom Y Z)
  where
  private
    module X = AContext X
    module Y = AContext Y
    module Z = AContext Z
    module F = AContextHom F
    module G = AContextHom G

  G↓⨾Y↑⨾F↓ : Mapping (ℙ Z.att) (ℙ X.ent)
  G↓⨾Y↑⨾F↓ = G.mor ↓ ⨾₁ Y.inc ↑ ⨾₁ F.mor ↓

  G↓⨾Y↑⨾F↓-Lub-cocontinuous : Lub-cocontinuous G↓⨾Y↑⨾F↓
  G↓⨾Y↑⨾F↓-Lub-cocontinuous = ↓↑↓-Lub-cocontinuous F.mor Y.inc G.mor F.trgCompat G.srcCompat

  [⨾⨾] ↓ : [ G↓⨾Y↑⨾F↓ ] ↓ ≈₁ G↓⨾Y↑⨾F↓
  [⨾⨾] ↓ = [ ] ↓ G↓⨾Y↑⨾F↓ (↓↑↓-Lub-cocontinuous F.mor Y.inc G.mor F.trgCompat G.srcCompat)

  infixr 9 _⨾⨾_
  _⨾⨾_ : AContextHom X Z
  _⨾⨾_ = record
    {mor = [ G↓⨾Y↑⨾F↓ ]
    ; srcCompat = ≈₁-begin
        [ G↓⨾Y↑⨾F↓ ] ↓ ⨾₁ X.inc ↑ ⨾₁ X.inc ↓
      ≈₁⟨ ⨾-cong₁ [⨾⨾] ↓ ⟨≈≈⟩ ⨾-assoc₃₊₁ ⟩
        G.mor ↓ ⨾₁ Y.inc ↑ ⨾₁ F.mor ↓ ⨾₁ X.inc ↑ ⨾₁ X.inc ↓
      ≈₁⟨ ⨾-cong₂₂ F.srcCompat ⟩
        G.mor ↓ ⨾₁ Y.inc ↑ ⨾₁ F.mor ↓
      ≈₁˘⟨ [⨾⨾] ↓ ⟩
        [ G↓⨾Y↑⨾F↓ ] ↓
      □₁
    ; trgCompat = ≈₁-begin
        (Z.inc ↓ ⨾₁ Z.inc ↑) ⨾₁ [ G↓⨾Y↑⨾F↓ ] ↓
      ≈₁⟨ ⨾-cong₂ [⨾⨾] ↓ ⟩
        (Z.inc ↓ ⨾₁ Z.inc ↑) ⨾₁ G.mor ↓ ⨾₁ Y.inc ↑ ⨾₁ F.mor ↓
      ≈₁⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ G.trgCompat ⟩
        G.mor ↓ ⨾₁ Y.inc ↑ ⨾₁ F.mor ↓
      ≈₁˘⟨ [⨾⨾] ↓ ⟩
        [ G↓⨾Y↑⨾F↓ ] ↓
      □₁
    }
open AContextHom-Comp public

ACH-leftId : {X Y : AContext} {F : AContextHom X Y} → AContext-Id ⨾⨾ F ≋ F
ACH-leftId {X} {Y} {F} = ≈-begin
    [ F.mor ↓ ⨾₁ X.inc ↑ ⨾₁ X.inc ↓ ]
  ≈⟨ [ ] -cong {f₁ = F.mor ↓ ⨾₁ X.inc ↑ ⨾₁ X.inc ↓} {F.mor ↓} F.srcCompat ⟩
    [ F.mor ↓ ]
  ≈⟨ [↓] F.mor ⟩
    F.mor
  □
  where
    module X = AContext X
    module F = AContextHom F

ACH-rightId : {X Y : AContext} {F : AContextHom X Y} → F ⨾⨾ AContext-Id ≋ F
ACH-rightId {X} {Y} {F} = ≈-begin
    [ Y.inc ↓ ⨾₁ Y.inc ↑ ⨾₁ F.mor ↓ ]
  ≈⟨ [ ] -cong {f₁ = Y.inc ↓ ⨾₁ Y.inc ↑ ⨾₁ F.mor ↓} {F.mor ↓} (⨾-assocL ⟨≈≈⟩ F.trgCompat) ⟩
    [ F.mor ↓ ]
  ≈⟨ [↓] F.mor ⟩
    F.mor
  □
  where
    module Y = AContext Y
    module F = AContextHom F
```

X₁ —F→ X₂ —G→ X₃ —H→ X₄

```
ACH-assoc : {X₁ X₂ X₃ X₄ : AContext}
            {F : AContextHom X₁ X₂} {G : AContextHom X₂ X₃} {H : AContextHom X₃ X₄}
          → (F ⨾⨾ G) ⨾⨾ H ≋ F ⨾⨾ (G ⨾⨾ H)
ACH-assoc {X₁} {X₂} {X₃} {X₄} {F} {G} {H} = [ ] -cong
  {f₁ = H.mor ↓ ⨾₁ X₃.inc ↑ ⨾₁ FG.mor ↓}
  {f₂ = GH.mor ↓ ⨾₁ X₂.inc ↑ ⨾₁ F.mor ↓}
  (≈₁-begin
    H.mor ↓ ⨾₁ X₃.inc ↑ ⨾₁ FG.mor ↓
  ≈₁⟨ ⨾-cong₂₂ ([⨾⨾] ↓ F G) ⟩
    H.mor ↓ ⨾₁ X₃.inc ↑ ⨾₁ G.mor ↓ ⨾₁ X₂.inc ↑ ⨾₁ F.mor ↓
  ≈₁⟨ ⨾-assocL₃₊₁ ⟨≈≈˘⟩ ⨾-cong₁ ([⨾⨾] ↓ G H) ⟩
    GH.mor ↓ ⨾₁ X₂.inc ↑ ⨾₁ F.mor ↓
  □₁)
  where
    FG = F ⨾⨾ G
    GH = G ⨾⨾ H
    module X₂ = AContext X₂
    module X₃ = AContext X₃
    module F = AContextHom F
    module G = AContextHom G
    module H = AContextHom H
    module FG = AContextHom FG
    module GH = AContextHom GH

ACH-cong : {X₁ X₂ X₃ : AContext} {F₁ F₂ : AContextHom X₁ X₂} {G₁ G₂ : AContextHom X₂ X₃}
         → F₁ ≋ F₂ → G₁ ≋ G₂ → F₁ ⨾⨾ G₁ ≋ F₂ ⨾⨾ G₂
ACH-cong {X₁} {X₂} {X₃} {F₁} {F₂} {G₁} {G₂} F₁≋F₂ G₁≋G₂ = [ ] -cong
  {f₁ = G₁.mor ↓ ⨾₁ X₂.inc ↑ ⨾₁ F₁.mor ↓}
  {f₂ = G₂.mor ↓ ⨾₁ X₂.inc ↑ ⨾₁ F₂.mor ↓}
  (⨾-cong (↓-cong G₁≋G₂) (⨾-cong₂ (↓-cong F₁≋F₂)))
  where
    module X₂ = AContext X₂
    module F₁ = AContextHom F₁
    module F₂ = AContextHom F₂
    module G₁ = AContextHom G₁
    module G₂ = AContextHom G₂

ACH-Category : Category (i ⊍ j ⊍ k₁ ⊍ k₂) k₁ AContext
ACH-Category = record
  {semigroupoid = record
    {Hom = λ X Y → record
      {Carrier = AContextHom X Y
      ; _≈_ = _≋_
      ; isEquivalence = record {refl = ≈-refl; sym = ≈-sym; trans = ≈-trans}
```

```
      }
    ; compOp = record
      {_⨾_ = _⨾⨾_
      ; ⨾-cong = λ   {X₁} {X₂} {X₃} {F₁} {F₂} {G₁} {G₂}
        → ACH-cong {X₁} {X₂} {X₃} {F₁} {F₂} {G₁} {G₂}
      ; ⨾-assoc = λ {X₁} {X₂} {X₃} {X₄} {F} {G} {H} → ACH-assoc {F = F} {G} {H}
      }
    }
  ; idOp = record
    {Id = AContext-Id
    ; leftId = λ {X} {Y} {F} → ACH-leftId {X} {Y} {F}
    ; rightId = λ {X} {Y} {F} → ACH-rightId {X} {Y} {F}
    }
  }
```

8 Abstract Complete Semilattices in OCCs

8.1 Categoric.OCC.CSL

Abstract Complete Semilattices

```
module Categoric.OCC.CSL {i j k₁ k₂} {Obj : Set i} (occ : OCC j k₁ k₂ Obj)
  (let open OCC occ)
  (leftResOp  : LeftResOp orderedSemigroupoid)
  (rightResOp : RightResOp orderedSemigroupoid)
  (syqOp      : SyqOp osgc)
  where
open SyqOp                                                   syqOp
open OCC-SyQ-Props         occ                               syqOp
open SyQ-ResidualProps     osgc             leftResOp rightResOp syqOp
open ResidualOps                            leftResOp rightResOp
open OrdCat-Residual-Props orderedCategory  leftResOp rightResOp
open OSGC-Residuals        osgc             leftResOp rightResOp
open import Categoric.OCC.Order occ leftResOp rightResOp syqOp using (IsOrder; module IsOrder)
open Category₁ (MapCat occ)
open import Categoric.OSGC.PowerOp osgc   -- using ()

record ACSL : Set (i ⊍ j ⊍ k₂) where
  field
    Carrier : Obj
    ≼       : Mor Carrier Carrier
    ≼-isOrder : IsOrder ≼

  open IsOrder ≼-isOrder public renaming
    (refl to ≼-refl; trans to ≼-trans; antisym≈ to ≼⨉≼; ˘-antisym≈ to ≽⨉≽)
  field
    glb-total : {I : Obj} (R : Mor I Carrier) → isTotal (glb R)
  glb-isMapping : {I : Obj} (R : Mor I Carrier) → isMapping (glb R)
  glb-isMapping R = glb-isUnivalent, glb-total R
  glb-Mapping : {I : Obj} (R : Mor I Carrier) → Mapping I Carrier
  glb-Mapping R = record {mor = glb R; prf = glb-isMapping R}
  lub-total : {I : Obj} (Q : Mor I Carrier) → isTotal (lub Q)
  lub-total {I} Q = total-glb→total-lub {I} (λ {Q} → glb-total {I} Q) {Q}
  lub-isMapping : {I : Obj} (R : Mor I Carrier) → isMapping (lub R)
  lub-isMapping R = lub-isUnivalent, lub-total R
  lub-Mapping : {I : Obj} (R : Mor I Carrier) → Mapping I Carrier
  lub-Mapping R = record {mor = lub R; prf = lub-isMapping R}
```

Although we can prove monotonicity from continuity, we still include both in our definition of CSL homomorphisms to allow for more efficient implementations in cases where these proofs are relevant.

```
record ACSLHom (A B : ACSL) : Set (i ⊍ j ⊍ k₁ ⊍ k₂) where   -- (i ⊍ j ⊍ k₁ ⊍ k₂) where
  module A = ACSL A
  module B = ACSL B
  field
    map : Mapping A.Carrier B.Carrier
  map₀ : Mor A.Carrier B.Carrier
  map₀ = Mapping.mor map
  field
    monotone : A.≼ ⨾ map₀ ⊑ map₀ ⨾ B.≼
    continuous : {I : Obj} {S : Mor I A.Carrier} → A.glb S ⨾ map₀ ≈ B.glb (S ⨾ map₀)
```

Since monotonicity follows from continuity, we also provide a constructor that only requires a proof of continuity. Let us assume a continuous map between the carriers of two ACSLs:

```
module _ (A B : ACSL)
         (let module A = ACSL A)
         (let module B = ACSL B)
         (map : Mapping A.Carrier B.Carrier)
         (let map₀ = Mapping.mor map)
         (continuous : {I : Obj} {S : Mor I A.Carrier} → A.glb S ⨾ map₀ ≈ B.glb (S ⨾ map₀))
  where
```

For the purpose of proving monotonicity from continuity, we first show a little lemma: For set-based orders, the greatest lower bound of the image of the "up-set" of any element exists and is the image of that element.

```
  glb-≼⨾continuous : B.glb (A.≼ ⨾ map₀) ≈ map₀
  glb-≼⨾continuous = ≈-begin
      B.glb (A.≼ ⨾ map₀)
    ≈˘⟨ continuous ⟩
      A.glb A.≼ ⨾ map₀
    ≈⟨ ⨾-cong₁ A.glb-order ⟨≈≈⟩ leftId ⟩
      map₀
    □
```

The proof of monotonicity only needs totality and continuity of map; it does not even need completeness (totality of glb). The proof below essentially proves monotonicity in the shape map₀ ˘ ⨾ A.≼ ⨾ map₀ ⊑ B.≼ by replacing the first map₀ with B.glb (A.≼ ⨾ map₀) using the lemma above, and then using the glb definition in B. The step using B.order-\ at the end of the calculation corresponds to using an "indirect inclusion" argument.

```
  mkACSLHom : ACSLHom A B
  mkACSLHom = record
    {map = map
    ; continuous = continuous
    ; monotone = ⊑-begin
        A.≼ ⨾ map₀
      ⊑⟨ proj₁ (mappingTotal map) ⟨⊑≈⟩ ⨾-assoc ⟩
        map₀ ⨾ map₀ ˘ ⨾ A.≼ ⨾ map₀
      ≈⟨ ⨾-cong₂₁ (˘-cong glb-≼⨾continuous ⟨≈˘≈⟩ (⨉-˘ ⟨≈≈⟩ ⨉-cong₂ ˘\˘-˘)) ⟩
        map₀ ⨾ (B.≼ ⨉ (B.≼ / (A.≼ ⨾ map₀))) ⨾ A.≼ ⨾ map₀
      ⊑⟨ ⨾-monotone₂ (\-universal (⊑-begin
          B.≼ ⨾ (B.≼ ⨉ (B.≼ / (A.≼ ⨾ map₀))) ⨾ A.≼ ⨾ map₀
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ⨉-cancel-left ⟩
          (B.≼ / (A.≼ ⨾ map₀)) ⨾ A.≼ ⨾ map₀
        ⊑⟨ /-cancel-outer ⟩
          B.≼
        □)) ⟩
        map₀ ⨾ (B.≼ \ B.≼)
```

```
        ≈⟨ ⨾-cong₂ B.order-\ ⟩
           map₀ ⨾ B.≼
        □
      }

_≋_ : {A B : ACSL} → ACSLHom A B → ACSLHom A B → Set k₁
F ≋ G = ACSLHom.map F ≈₁ ACSLHom.map G

ACSL-Id : {A : ACSL} → ACSLHom A A
ACSL-Id {A} = let open ACSL A in record
   {map = MappingId
   ; monotone = ⊑-reflexive (rightId ⟨≈≈˘⟩ leftId)
   ; continuous = λ {I} {S} → rightId ⟨≈≈˘⟩ glb-cong rightId
   }

_⨾⨾_ : {A B C : ACSL} → ACSLHom A B → ACSLHom B C → ACSLHom A C
_⨾⨾_ {A} {B} {C} F G = let
     module A = ACSL A
     module B = ACSL B
     module C = ACSL C
     module F = ACSLHom F
     module G = ACSLHom G
     FG = F.map ⨾₁ G.map
   in record
     {map = FG
     ; monotone = ⊑-begin
           A.≼ ⨾ F.map₀ ⨾ G.map₀
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ F.monotone ⟨⊑≈⟩ ⨾-assoc ⟩
           F.map₀ ⨾ B.≼ ⨾ G.map₀
        ⊑⟨ ⨾-monotone₂ G.monotone ⟨⊑≈⟩ ⨾-assocL ⟩
           (F.map₀ ⨾ G.map₀) ⨾ C.≼
        □
     ; continuous = λ {I} {S} → ≈-begin
           A.glb S ⨾ F.map₀ ⨾ G.map₀
        ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ F.continuous ⟩
           B.glb (S ⨾ F.map₀) ⨾ G.map₀
        ≈⟨ G.continuous ⟨≈≈⟩ C.glb-cong ⨾-assoc ⟩
           C.glb (S ⨾ F.map₀ ⨾ G.map₀)
        □
     }

open import Categoric.Semigroupoid
open import Categoric.IdOp

ACSL-Hom : LocalSetoid ACSL (i ⊍ j ⊍ k₁ ⊍ k₂) k₁
ACSL-Hom A B = record
   {Carrier = ACSLHom A B
   ; _≈_ = _≋_
   ; isEquivalence = record {refl = ≈-refl; sym = ≈-sym; trans = ≈-trans}
   }

ACSL-CompOp : CompOp ACSL-Hom
ACSL-CompOp = record
   {_⨾_ = _⨾⨾_
   ; ⨾-cong = ⨾-cong
   ; ⨾-assoc = ⨾-assoc
   }

ACSL-IdOp : IdOp ACSL-Hom _⨾⨾_
ACSL-IdOp = record
   {Id = ACSL-Id
   ; leftId = leftId
   ; rightId = rightId
   }

ACSL-Semigroupoid : Semigroupoid (i ⊍ j ⊍ k₁ ⊍ k₂) k₁ ACSL
ACSL-Semigroupoid = record {Hom = ACSL-Hom; compOp = ACSL-CompOp}

ACSL-Category : Category (i ⊍ j ⊍ k₁ ⊍ k₂) k₁ ACSL
ACSL-Category = record {semigroupoid = ACSL-Semigroupoid; idOp = ACSL-IdOp}
```

8.2 Categoric.OCC.CSL.ToAContext

```
module Categoric.OCC.CSL.ToAContext {i j k₁ k₂} {Obj : Set i} (occ : OCC j k₁ k₂ Obj)
   (let open OCC occ)
   (leftResOp  : LeftResOp orderedSemigroupoid)
   (rightResOp : RightResOp orderedSemigroupoid)
   (syqOp      : SyqOp osgc)
   (let open OCC-DirectPower occ leftResOp rightResOp syqOp)
   (directPower : DirectPower)
   (splitSymIdempot : {A : Obj} {E : Mor A A} (isSymIdempot : IsSymIdempot E) → SymSplitting E)
   where

open SyqOp                                                      syqOp
open OCC-SyQ-Props          occ                                 syqOp
open SyQ-ResidualProps      osgc             leftResOp rightResOp syqOp
open ResidualOps                             leftResOp rightResOp
open OrdCat-Residual-Props orderedCategory   leftResOp rightResOp
open OSGC-Residuals         osgc             leftResOp rightResOp

open DirectPower directPower using
   (ℙ; ∈; Ω; Ω˘; Ω˘-isPreorder₀; powerOp
   ; Λ; Λ-isMapping; Λ₀; Λ-cong; Λ⨾Ω˘)
open PowerOp osgc powerOp using (Λ⨾∈˘; map-Λ)
open Category₁ (MapCat occ)

open import Categoric.OSGC.Power.Polarities osgc leftResOp rightResOp powerOp
open import Data.AContext.InOSGC          osgc leftResOp rightResOp powerOp
open import Categoric.OCC.CSL             occ  leftResOp rightResOp syqOp
open import Categoric.OCC.DirectPower.OrderPolarities occ leftResOp rightResOp syqOp directPower
open import Categoric.OCC.DirectPower.Polarities occ leftResOp rightResOp syqOp directPower
```

The following “extended ACSL” module will be used to provide, for derived order properties, names qualified with “A.” and “B.” in the context of a ACSLHom A B below.

```
module ACSL′ (A : ACSL) where
   open ACSL A public
   open OrderPolarities ≼-isOrder public
   open Complete (λ {I} {R} → lub-isMapping R) (λ {I} {R} → glb-isMapping R) public

fromACSL : ACSL → AContext
fromACSL A = record {ent = Carrier; att = Carrier; inc = ≼}
   where open ACSL A
```

Note the contravariance of fromACSLHom:

```
fromACSLHom : {A B : ACSL} → ACSLHom A B → AContextHom (fromACSL B) (fromACSL A)
fromACSLHom {A} {B} hom = record {mor = ≼B ⨾ map₀ ˘
   ; srcCompat = ≈-begin
         (≼B ⨾ map₀ ˘) ↓₀ ⨾ ≼B ↑↓₀
      ≈⟨ ⨾-cong₂ ↑↓≈⊠ ⟩
         (≼B ⨾ map₀ ˘) ↓₀ ⨾ ((≼B / (∈ \ ≼B)) ⊠ ∈)
```

```
    ≈⟨ ≬-in-left (Mapping.prf ((≼B ⨾ map₀ ˘) ↓)) ⟩
      ((≼B / (∈ \ ≼B)) ⨾ (≼B ⨾ map₀ ˘) ↓₀ ˘) ≬ ∈
    ≈⟨ ≬-cong₁ (/-inner-⨾ (Mapping.prf ((≼B ⨾ map₀ ˘) ↓))) ⟩
      (≼B / ((≼B ⨾ map₀ ˘) ↓₀ ⨾ (∈ \ ≼B))) ≬ ∈
    ≈⟨ ≬-cong₁ (/-cong₂ (\-inner-⨾ (Mapping.prf ((≼B ⨾ map₀ ˘) ↓)) ⟨≈≈˘⟩ \-cong₁ ˘-involutionRightConv)) ⟩
      (≼B / (((≼B ⨾ map₀ ˘) ↓₀ ⨾ ∈ ˘) ˘ \ ≼B)) ≬ ∈
    ≈⟨ ≬-cong₁ (/-cong₂ (\-cong₁ (˘-cong ↓⨾∈˘ ⟨≈≈⟩ \˘-˘))) ⟩
      (≼B / (((≼B ⨾ map₀ ˘) / ∈ ˘) \ ≼B)) ≬ ∈
    ≈˘⟨ ≬-cong₁ (/-cong₂ (\-cong₁ (/-flip (Mapping.prf map)))) ⟩
      (≼B / ((≼B / (∈ ˘ ⨾ map₀)) \ ≼B)) ≬ ∈
    ≈⟨ ≬-cong₁ S/∘\S∘S/ ⟩
      (≼B / (∈ ˘ ⨾ map₀)) ≬ ∈
    ≈⟨ ≬-cong₁ (/-flip (Mapping.prf map)) ⟩
      ((≼B ⨾ map₀ ˘) / ∈ ˘) ≬ ∈
    ≈˘⟨ ↓≈≬ ⟩
      (≼B ⨾ map₀ ˘) ↓₀
    □
  ; trgCompat = ≈-begin
      ≼A ↓↑₀ ⨾ (≼B ⨾ map₀ ˘) ↓₀
    ≈⟨ ⨾-cong₂ lemma ⟩
      ≼A ↓↑₀ ⨾ A.meet ⨾ map₀ ⨾ B.downset
    ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ A.≼↓↑-⨾-glb∈˘ ⟩
      A.meet ⨾ map₀ ⨾ B.downset
    ≈˘⟨ lemma ⟩
      (≼B ⨾ map₀ ˘) ↓₀
    □}
  where open ACSLHom hom hiding (module A; module B)
    module A = ACSL′ A
    module B = ACSL′ B
    open A using () renaming (≼ to ≼A; Carrier to A₀)
    open B using () renaming (≼ to ≼B; Carrier to B₀)
```

Above, there is a direct proof for srcCompat, obtained mainly by expanding definitions and using properties of residuals and symmetric quotients. We also include an alternative proof that remains at the level of order concepts:

```
    srcCompat-Λ : (≼B ⨾ map₀ ˘) ↓₀ ⨾ ≼B ↑↓₀ ≈ (≼B ⨾ map₀ ˘) ↓₀
    srcCompat-Λ = ≈-begin
        (≼B ⨾ map₀ ˘) ↓₀ ⨾ ≼B ↑↓₀
      ≈⟨ ⨾-cong (B.lemma₀ (Mapping.prf map)) B.≼↑↓≈ΛlbdUbd∈˘ ⟩
        Λ₀ (B.lbd (∈ ˘ ⨾ map₀)) ⨾ Λ₀ (B.lbd (B.ubd (∈ ˘)))
      ≈⟨ map-Λ {f = Λ (B.lbd (∈ ˘ ⨾ map₀))} ⟩
        Λ₀ (Λ₀ (B.lbd (∈ ˘ ⨾ map₀)) ⨾ B.lbd (B.ubd (∈ ˘)))
      ≈⟨ Λ-cong (B.Mapping-⨾-lbd-ubd Λ-isMapping) ⟩
        Λ₀ (B.lbd (B.ubd (Λ₀ (B.lbd (∈ ˘ ⨾ map₀)) ⨾ ∈ ˘)))
      ≈⟨ Λ-cong (B.lbd-cong (B.ubd-cong Λ⨾∈˘)) ⟩
        Λ₀ (B.lbd (B.ubd (B.lbd (∈ ˘ ⨾ map₀))))
      ≈⟨ Λ-cong B.lbd-ubd-lbd ⟩
        Λ₀ (B.lbd (∈ ˘ ⨾ map₀))
      ≈˘⟨ B.lemma₀ (Mapping.prf map) ⟩
        (≼B ⨾ map₀ ˘) ↓₀
      □
```

The following lemma₁ is used to show lemma below, which is used in the proof of trgCompat above.

Due to continuity, the lower bounds of the map-image of some set are exactly the members of the downset of the map-image of the glb of that set:

```
    lemma₁ : Λ₀ (B.lbd (∈ ˘ ⨾ map₀)) ≈ A.glb (∈ ˘) ⨾ map₀ ⨾ B.downset
    lemma₁ = ≈-begin
        Λ₀ (B.lbd (∈ ˘ ⨾ map₀))
      ≈˘⟨ Λ-cong (B.total-glb-⨾-order˘ (B.glb-total (∈ ˘ ⨾ map₀))) ⟩
        Λ₀ (B.glb (∈ ˘ ⨾ map₀) ⨾ ≼B ˘)
      ≈˘⟨ Λ-cong (⨾-cong₁ continuous) ⟩
        Λ₀ ((A.glb (∈ ˘) ⨾ map₀) ⨾ ≼B ˘)
      ≈⟨ map-Λ {f = A.glb-Mapping (∈ ˘) ⨾₁ map} ⟨≈˘≈⟩ ⨾-assoc ⟩
        A.glb (∈ ˘) ⨾ map₀ ⨾ B.downset
      □

    lemma : (≼B ⨾ map₀ ˘) ↓₀ ≈ A.glb (∈ ˘) ⨾ map₀ ⨾ B.downset
    lemma = ≈-begin
        (≼B ⨾ map₀ ˘) ↓₀
      ≈⟨ B.lemma₀ (Mapping.prf map) ⟩
        Λ₀ (B.lbd (∈ ˘ ⨾ map₀))
      ≈⟨ lemma₁ ⟩
        A.glb (∈ ˘) ⨾ map₀ ⨾ B.downset
      □
```

8.3 Categoric.OCC.CSL.FromAContext

```
module Categoric.OCC.CSL.FromAContext {i j k₁ k₂} {Obj : Set i} (occ : OCC j k₁ k₂ Obj)
  (let open OCC occ)
  (leftResOp  : LeftResOp orderedSemigroupoid)
  (rightResOp : RightResOp orderedSemigroupoid)
  (syqOp      : SyqOp osgc)
  (let open OCC-DirectPower occ leftResOp rightResOp syqOp)
  (directPower : DirectPower)
  (splitSymIdempot : {A : Obj} {E : Mor A A} (isSymIdempot : IsSymIdempot E) → SymSplitting E)
  where
open SyqOp                                                          syqOp
open OCC-SyQ-Props          occ                                     syqOp
open SyQ-ResidualProps      osgc              leftResOp rightResOp syqOp
open ResidualOps                              leftResOp rightResOp
open OrdCat-Residual-Props orderedCategory leftResOp rightResOp
open OSGC-Residuals         osgc              leftResOp rightResOp

open import Categoric.OCC.Order            occ  leftResOp rightResOp syqOp
  using (IsOrder; module IsOrder; module SubOrder; IsOrder-subst)
module ℙ = DirectPower directPower
open ℙ using (powerOp; ℙ; Ω; Ω-isOrder; Ω-trans; Ω-refl; Ω≬Ω)
open import Categoric.OSGC.Power.Polarities osgc leftResOp rightResOp powerOp
open import Data.AContext.InOSGC            osgc leftResOp rightResOp powerOp
open import Categoric.OCC.CSL               occ  leftResOp rightResOp syqOp
open import Categoric.OCC.DirectPower.Polarities occ leftResOp rightResOp syqOp directPower
  using (ℙglb-preserves-↓↑)

toACSL : AContext → ACSL
toACSL A = record
    {Carrier = ↓↑-image
    ; ≼ = ≼  -- = Q˘ ⨾ Ω ⨾ Q – leave, subset ordering, then come back
    ; ≼-isOrder = ≼-isOrder
    ; glb-total = λ {I : Obj} (R : Mor I ↓↑-image) → isTotal-from-I (⊑-begin
          Id
        ⊑⟨ ℙ.glbΩ-totalI ⟩
          ℙ.glb (R ⨾ Q˘) ⨾ ℙ.glb (R ⨾ Q˘) ˘
```

```
        ⊑⟨ ⨾-monotone ≼-glb-R-⊒ (˘-monotone ≼-glb-R-⊒ ⟨⊑≈⟩ ˘-involution) ⟩
          (≼.glb R ⨾ Q˘) ⨾ Q˘ ˘ ⨾ ≼.glb R ˘
        ⊑⟨ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ (⨾-assocL ⟨≈⊑⟩ proj₁ Q˘-isInjective) ⟩
          ≼.glb R ⨾ ≼.glb R ˘
        □)
  }
where
open AContext A
inc↓↑ : Mapping (ℙ att) (ℙ att)
inc↓↑ = inc ↓↑
inc↓↑₀ : Mor (ℙ att) (ℙ att)
inc↓↑₀ = Mapping.mor inc↓↑
Ξ : Mor (ℙ att) (ℙ att)
Ξ = inc↓↑₀ ˘ ⨾ inc↓↑₀
Ξ-isSubIdentity : isSubidentity Ξ
Ξ-isSubIdentity = mappingUnivalent inc↓↑
Ξ-isSymIdempot : IsSymIdempot Ξ
Ξ-isSymIdempot = record
  {symmetric = ˘-involutionLeftConv
  ; idempotent = ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ (mappingBiDifunctional inc↓↑)
  }
split : SymSplitting Ξ
split = splitSymIdempot {ℙ att} {Ξ} Ξ-isSymIdempot
module Q = SymSplitting split
open Q renaming (obj to ↓↑-image; mor to Q; proof to isSymSplitting)
Q˘ : Mor ↓↑-image (ℙ att)
Q˘ = Q ˘
Q˘-isInjective : isInjective Q˘
Q˘-isInjective = IsSymSplitting-isinjective isSymSplitting
Q˘-isMapping : isMapping Q˘
Q˘-isMapping = IsSymSplitting-isMapping isSymSplitting Ξ-isSubIdentity
  -- Q : Mor (ℙ att) ↓↑-image is a univalent surjection; due to splitId.
Q-isUnivalent : isUnivalent Q
Q-isUnivalent = isUnivalentFromInjective Q˘-isInjective
Q-isInjective : isInjective Q
Q-isInjective = isInjectiveFromUnivalent (proj₁ Q˘-isMapping)
F : Mapping ↓↑-image (ℙ att)
F = record {mor = Q ˘; prf = Q˘-isMapping}
≼ : Mor ↓↑-image ↓↑-image
≼ = Q˘ ⨾ Ω ⨾ Q   -- leave, subset ordering, then come back
open SubOrder {ℙ att} {Ω} Ω-isOrder {↓↑-image} F Q˘-isInjective using ()
  renaming (subOrder to ≼′; subOrder-isOrder to ≼′-isOrder)
≼-isOrder : IsOrder ≼
≼-isOrder = IsOrder-subst (⨾-cong₂₂ ˘˘) ≼′-isOrder
module ≼ = IsOrder ≼-isOrder
module _ {I : Obj} {R : Mor I ↓↑-image} where
  ≼-lbd-˘ : ≼.lbd R ˘ ≈ Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘
  ≼-lbd-˘ = ≈-begin
        ≼.lbd R ˘
    ≈⟨ ˘\˘-˘ ⟨≈≈˘⟩ /-outer-⨾-≈ Q˘-isMapping ⟩
        Q˘ ⨾ ((Ω ⨾ Q) / R)
    ≈˘⟨ ⨾-cong₂ (/-flip-˘ Q˘-isMapping) ⟩
        Q˘ ⨾ (Ω / (R ⨾ Q˘))
    ≈˘⟨ ⨾-cong₂ ˘\˘-˘ ⟩
        Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘
    □
  R⨾Q˘-closed : (R ⨾ Q˘) ⨾ inc↓↑₀ ≈ R ⨾ Q˘
  R⨾Q˘-closed = mapRanClosed-→ (Mapping.prf inc↓↑) ↓↑-idempotent (⊑-reflexive (≈-begin
          R ⨾ Q˘
        ≈⟨ ⨾-cong₂ Q.˘-rightClosed ⟨≈˘≈⟩ ⨾-assocL ⟩
          (R ⨾ Q˘) ⨾ inc↓↑₀ ˘ ⨾ inc↓↑₀
        □))
  ≼-glb-R-⊒ : ℙ.glb (R ⨾ Q˘) ⊑ ≼.glb R ⨾ Q˘
  ≼-glb-R-⊒ = ⊑-begin
        ℙ.glb (R ⨾ Q˘)
      ⊑⟨ ℙglb-preserves-↓↑ R⨾Q˘-closed ⟩
        ℙ.glb (R ⨾ Q˘) ⨾ inc↓↑₀ ˘ ⨾ inc↓↑₀
      ≈˘⟨ ⨾-cong₂ factors ⟩
        ℙ.glb (R ⨾ Q˘) ⨾ Q ⨾ Q˘
      ≈⟨⟩
        ((ℙ.lbd (R ⨾ Q˘)) ˘ ⨉ Ω) ⨾ Q ⨾ Q˘
      ⊑⟨ ⨾-monotone₁ ⨉-cancel-inner ⟩
        ((Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘) ⨉ (Q˘ ⨾ Ω)) ⨾ Q ⨾ Q˘
      ≈˘⟨ ⨾-cong₂₁ ˘˘ ⟩
        ((Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘) ⨉ (Q˘ ⨾ Ω)) ⨾ Q˘ ˘ ⨾ Q˘
      ≈˘⟨ ⨾-cong₁ (⨉-in-right (˘-isBijective Q˘-isMapping)) ⟨≈˘≈⟩ ⨾-assoc ⟩
        ((Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘) ⨉ ((Q˘ ⨾ Ω) ⨾ Q˘ ˘)) ⨾ Q˘
      ≈˘⟨ ⨾-cong₁ (⨉-cong ≼-lbd-˘ (⨾-assocL ⟨≈≈˘⟩ ⨾-cong₂ ˘˘)) ⟩
        (≼.lbd R ˘ ⨉ ≼) ⨾ Q˘
      ≈⟨⟩
        ≼.glb R ⨾ Q˘
      □
```

9 Conclusion

Beyond the theoretically interesting fact that order theory, where the condition of antisymmetry is naturally formalised using meets, can be formalised without meets in OCCs with residuals and symmetric quotients, this development also demonstrates that such an essentially theoretical development can be fully mechanised and still be presented in readable calculational style, where writing is not significantly more effort that a conventional calculational presentation in LaTeX.

Beyond basic order concepts, such as bounds and extrema, we also formalised Galois connections, direct powers, polarities, and a category of contexts in the sense of formal concept analysis, with the dual functors connecting it with the category of complete lower semilattices.

Large parts of these developments do not even require the presence of identities; we separated most of these and set them in the context of ordered semigroupoids with residuals and/or symmetric quotients as appropriate.

Bibliography

Chritiene Aarts, Roland C. Backhouse, Paul Hoogendijk, Ed Voermans, and Jaap van der Woude. A relational theory of datatypes. Working document, December 1992. 387 pp., available at
http://www.cs.nott.ac.uk/~rcb/MPC/book.ps.gz.

Musa Al-hassy. A mechanisation of internal galois connections in order theory formalised without meets. M.sc. thesis, McMaster Unviersity, Dept. of Computing and Software, April 2015.

Rudolf Berghammer, Gunther Schmidt, and Hans Zierer. Symmetric quotients. Technical Report TUM-INFO 8620, Technische Universität München, Fakultät für Informatik, 1986. 18 p.

Rudolf Berghammer, Gunther Schmidt, and Hans Zierer. Symmetric quotients and domain constructions. *Inform. Process. Lett.*, 33(3):163–168, 1989.

Richard S. Bird and Oege de Moor. *Algebra of Programming*, volume 100 of *International Series in Computer Science*. Prentice Hall, 1997.

Marcel Erné. Categories of contexts. Preprint, `http://www.iazd.uni-hannover.de/~erne/preprints/CatConts.pdf`, 2005.

Peter J. Freyd and Andre Scedrov. *Categories, Allegories*, volume 39 of *North-Holland Mathematical Library*. North-Holland, Amsterdam, 1990. ISBN 0-444-70368-3 and 0-444-70367-5 (pbk).

Hitoshi Furusawa and Wolfram Kahl. A study on symmetric quotients. Technical Report 1998-06, Fakultät für Informatik, Universität der Bundeswehr München, December 1998.

Ralf Hinze. Kan extensions for program optimisation — or: Art and Dan explain an old trick. In Jeremy Gibbons and Pablo Nogueira, editors, *Mathematics of Program Construction, MPC 2012*, volume 7342 of *LNCS*, pages 324–362. Springer, 2012. doi: 10.1007/978-3-642-31113-0_16.

Pascal Hitzler, Markus Krötzsch, and Guo-Qiang Zhang. A categorical view on algebraic lattices in formal concept analysis. *Fund. Inform.*, 74(2–3):301–328, 2006.

Peter Jipsen. Categories of algebraic contexts equivalent to idempotent semirings and domain semirings. In Wolfram Kahl and Timothy G. Griffin, editors, *Relational and Algebraic Methods in Computer Science, RAMiCS 2012*, volume 7560 of *LNCS*, pages 195–206. Springer, 2012. ISBN 978-3-642-33314-9. doi: 10.1007/978-3-642-33314-9_13.

Wolfram Kahl. Relational semigroupoids: Abstract relation-algebraic interfaces for finite relations between infinite types. *J. Logic and Algebraic Programming*, 76(1):60–89, 2008. doi: 10.1016/jlap.2007.10.008.

Wolfram Kahl. Dependently-typed formalisation of relation-algebraic abstractions. In Harrie de Swart, editor, *Relational and Algebraic Methods in Computer Science, RAMiCS 2011*, volume 6663 of *LNCS*, pages 230–247. Springer, 2011. doi: 10.1007/978-3-642-21070-9_18.

Wolfram Kahl. A mechanised abstract formalisation of concept lattices. In Peter Höfner, Peter Jipsen, Wolfram Kahl, and Martin Eric Müller, editors, *Relational and Algebraic Methods in Computer Science, RAMiCS 2014*, volume 8428 of *LNCS*, pages 242–260. Springer, 2014a. ISBN 978-3-319-06250-1. doi: 10.1007/978-3-319-06251-8.

Wolfram Kahl. Relation-Algebraic Theories in Agda — RATH-Agda-2.0.1. Mechanically checked Agda theories available for download, with 456 pages literate document output. `http://RelMiCS.McMaster.ca/RATH-Agda/`., February 2014b. With contributions by Yuhang Zhao.

M. Andrew Moshier. A relational category of polarities. (unpublished draft), November 2013.

Gunther Schmidt and Thomas Ströhlein. *Relations and Graphs, Discrete Mathematics for Computer Scientists*. EATCS-Monographs on Theoret. Comput. Sci. Springer, 1993. ISBN 3-540-56254-0, 0-387-56254-0.

Rudolf Wille. Formal concept analysis as mathematical theory of concepts and concept hierarchies. In B. Ganter et al., editors, *Formal Concept Analysis*, volume 3626 of *LNAI*, pages 1–33. Springer, 2005.

Hans Zierer. Relation-algebraic domain constructions. *Theoretical Computer Science*, 87:163–188, 1991.