Order Theory and Concept Lattices in Ordered Categories Without Meets, Formalised in Agda

AContext-2.1

Wolfram Kahl
kahl@cas.mcmaster.ca

Musa Al-hassy
alhassy@gmail.com

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

29 July 2015

Abstract

Using the the dependently-typed programming language Agda, we formalise orders, Galois connections, membership relations, and a category of algebraic contexts with relational homomorphisms presented by Jipsen (2012) and Moshier (2013) together with the duality 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

Chapter 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 results 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 the functors inducing the duality. The natural isomorphisms necessary for completing the duality proof have their constituent material defined in Chapter 9, and are assembled to a formal duality proof in Chapter 10.

The Agda source code for this development is available on-line as free software under the GPL version 3 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

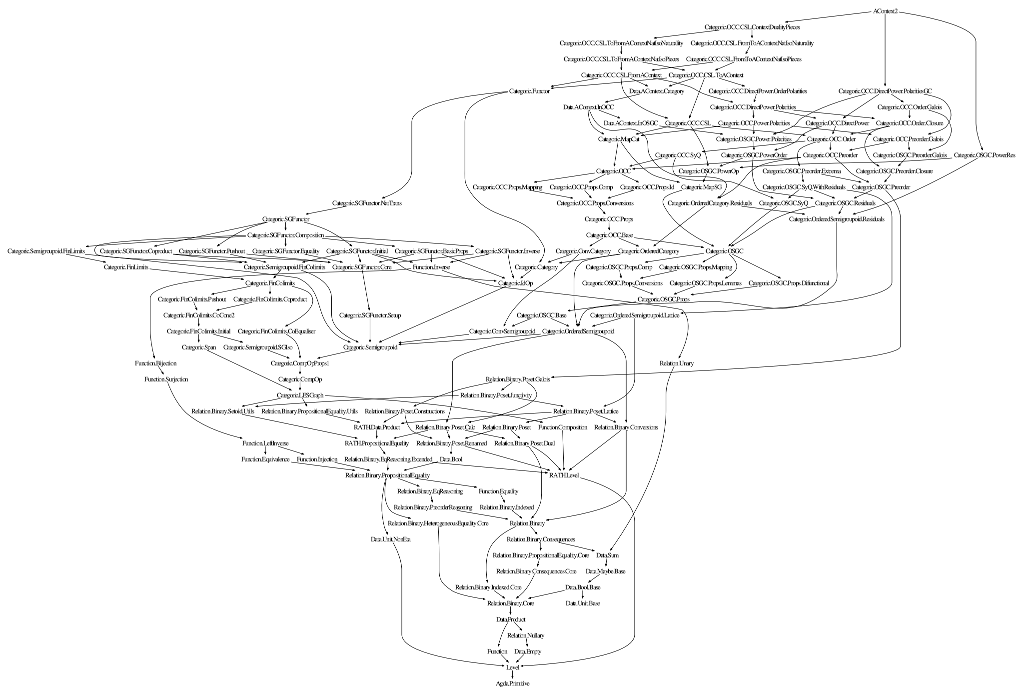
Loading this module forces typechecking of all theories contained in the AContext-2.1 document up to Chapter 9, but without the modules from Chapter 10, which require extraordinary ressources to type-check, see page 184 for more information.

```
module AContext2 where

open import Categoric.OCC.CSL.ContextDualityPieces public
open import Categoric.OCC.DirectPower.PolaritiesGC
open import Categoric.OSGC.PowerRes
```

Using this module as entry point, our whole development type-checks with the current stable version 2.4.2.3 of Agda (in about 27 minutes on a single core of an i7-4700MQ running at 2.1GHz, with 16GB or RAM installed and 13GB set as heap size, requiring over 10GB of heap space), and with development versions at least from mid July 2016 (with "`--sharing`" in about 14 minutes, requiring over 8GB of heap space).

In the following module dependency graph starting from this module AContext2, exactly Function.Inverse and Data.Bool together with the modules they depend on are part of the Agda standard library, and the upper-right cluster contains the modules reproduced in this document. Other RATH-Agda material is concentrated in the central cluster with the definition of OCCs and their properties, and in the left cluster with the definition of functors and their properties.



In the remainder of this document, we will set "trivial" module headers in tiny font; such "trivial" module headers consist only of an un-parameterised file-module declaration, and imports of either basic RATH-Agda modules, or of modules that have been presented before and the availability of which is typically obvious from the context.

Chapter 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 remaining modules contain new material, culminating in a formalisation of Galois connections based on Agda functions between the carriers of Posets in Relation.Binary.Poset.Galois (Sect. 2.4).

2.1 Relation.Binary.Poset.Renamed

```
module Relation.Binary.Poset.Renamed where
open import RATH.Level
open import Relation.Binary
open import Data.Bool
```

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.

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
```

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

```
⌊_≤⌋ : {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

```
module Relation.Binary.Poset.Constructions where

open import RATH.Level
open import RATH.Data.Product using (Σ:•; _,_; proj₁; proj₂)
open import Relation.Binary using (Poset)
open import Relation.Binary.Poset.Renamed
open import Function using (_on_)
```

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 P_0 form a poset, where the 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 is defined by $(x, sx) \le (y, sy) = x \le y$.

```
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

```
module Relation.Binary.Poset.Junctivity where
open import RATH.Level
open import Relation.Binary using (Poset)
open import Relation.Binary.Poset.Renamed
open import Relation.Binary.Poset.Dual
open import Relation.Binary.Poset.Lattice
open import Function using (_∘_)
open import RATH.Data.Product using (Σ:•; proj₂)
open import Relation.Binary.Conversions using (posetSetoid)
open import Relation.Binary.Setoid.Utils using (module SetoidB)
```

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 f 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:

```
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

```
module Relation.Binary.Poset.Galois where
open import RATH.Level
open import Function using (_∘_; id)
open import Relation.Binary using (Poset)
open import Relation.Binary.Poset.Renamed
open import Relation.Binary.Poset.Dual
open import Relation.Binary.Poset.Calc
open import Relation.Binary.Poset.Lattice
open import Relation.Binary.Poset.Constructions using (pointwisePoset; subPoset)
open import Relation.Binary.Conversions using (posetSetoid)
open import Relation.Binary.Poset.Junctivity using (IsMonotone; module order-isos-are-junctive)
open import Relation.Binary.Setoid.Utils using (module SetoidB; module SetoidA)
open import RATH.Data.Product using (Σ:•; _,_; proj₁; proj₂)
```

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:

$$k \le m \div n \;\Leftrightarrow\; k \times n \le m$$

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:

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

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

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

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

Where _⊑_ 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: $K\,x\,y \;=\; x$. So we work with the complex by shifting to the manageable:

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

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 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 $\max\,(x, y) \;=\; \sqcup f$; or unfolding the connection:

  $$\forall\, x, y, z \bullet \max\,(x, y) \le z \;\Leftrightarrow\; x \le z \wedge y \le z$$

  Likewise, the complicated notion of "greatest common divisor" is rendered trivial, via the constant function, as $\gcd\,(x, y) \;=\; \sqcup f$ — where the order is divisibility, of course.
- **Top and Bottom:** A poset P is bounded above by ⊤ and below by ⊥ 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 ⊥?
- **Colimits:** Interpreting the inclusions as certain Hom functors and equivalence _⇔_ as isomorphism _≅_ yields the notion that "functor f has colimit ⊔ 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

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

  Then, we find

  $$\forall\, Q, R, S \bullet Q \sqsubseteq R \setminus S \Leftrightarrow R \,⨾\, Q \sqsubseteq S \Leftrightarrow R \sqsubseteq S \,/\, Q$$

where

$$x\,(R \mathbin{⨾} S)\,y \;\Leftrightarrow\; (\exists z \bullet x\,R\,z \wedge z\,S\,y) \text{ and } R \sqsubseteq S \;\Leftrightarrow\; (\forall x, y \bullet x\,R\,y \Rightarrow x\,S\,y)$$

Then, we find three connections:

$$(R \mathbin{⨾}) \dashv (R \backslash) \text{ and } (S /) \dashv^{\smile} (S \backslash) \text{ and } (\mathbin{⨾} Q) \dashv (/ Q)$$

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

$$f \dashv^{\smile} g \;\Leftrightarrow\; (\forall x, y \bullet x \le f\,y \;\Leftrightarrow\; y \le g\,x)$$

Note: $(f, \le) \dashv^{\smile} (g, \sqsubseteq) \;\Leftrightarrow\; (f, \le^{\smile}) \dashv (g, \sqsubseteq)$.
See Chapter 3 for more on residuals.

- Integer division: $k \le m \div n \;\Leftrightarrow\; k \times n \le m$.
  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

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

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

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

  and likewise $R\downarrow = (R^{\smile})\uparrow$. 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 $f\,R\,x \;\Leftrightarrow\; f\,x = 0$.
- Syntax is adjoint to semantics: $(\vDash\uparrow) \dashv^{\smile} (\vDash\downarrow)$, 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 $(\vDash\downarrow) \mathbin{⨾} (\vDash\uparrow)$ and the notion of 'logical consequence'?
- Connections between powersets: $f \dashv^{\smile} g$, 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 $R{\leftarrow} = (R^{\smile}){\rightarrow}$. 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 $\{P\}\,S\,\{Q\} \;\Leftrightarrow\; S{\rightarrow}(P) \subseteq Q$ and, it is usually written that, $(S{\leftarrow})^{*}(Q) = wp.S.Q$. Then the connection takes the particular shape,

$$\{P\}\,S\,\{Q\} \;\Leftrightarrow\; P \subseteq wp.S.Q$$

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 PosetMeetl using (IsMeetl; module IsMeetl)
open PosetJoinl using (IsJoinl; module IsJoinl)
```

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

$$\forall x, y \bullet L\,x \sqsubseteq y \;\Leftrightarrow\; x \le U\,y$$

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 and 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 $(f \mathrel{\dot{\le}} g) \;\Leftrightarrow\; (\forall \{x\} \to f\,x \le g\,x)$ and likewise for $\dot{\sqsubseteq}$.

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, L⟨⊓ y • U y⟩ = ⟨⊓ y • L (u y) ⟩ 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)
  }
```

Interdefinability

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, ∀ x • L x = ⊓ {y | x ≤ U y}. Exercise, fill in the blanks: ∀ y • U x = _ {x | _}.

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 will be instantianted 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).

Chapter 3

Residuals in OCCs

Due to the importance of residuals and symmetric quotients for the current development, we include the corresponding RATH-Agda modules in the current document.

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

```
module Categoric.OrderedSemigroupoid.Residuals where
open import RATH.Level
open import RATH.Data.Product using (_,_)
open import Categoric.OrderedSemigroupoid

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
  infix 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

  /-cancel-outer-⊑ : {A B C : Obj} {S : Mor A C} {P R : Mor B C} → P ⊑ R → (S / R) ⨾ P ⊑ S
  /-cancel-outer-⊑ P⊑R = ⨾-monotone₂ P⊑R ⟨⊑⊑⟩ /-cancel-outer

  /-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
  infix 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
  \-cancel-outer-⊑ :  {A B C : Obj} {S : Mor A C} {P Q : Mor A B} → P ⊑ Q → P ⨾ (Q \ S) ⊑ S
  \-cancel-outer-⊑ P⊑Q = ⨾-monotone₁ P⊑Q ⟨⊑⊑⟩ \-cancel-outer
  \-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)


S/∘\S-into-/-⊑ : {A B C D : Obj} {S : Mor A D} {R : Mor A C} {Q : Mor B C}
                   → (S / ((R / Q) \ S)) ⊑ (S / (R \ S)) / Q
S/∘\S-into-/-⊑ {A} {B} {C} {D} {S} {R} {Q} = ⊑-begin
     S / ((R / Q) \ S)
   ⊑⟨ /-antitone (\-antitone (/-monotone ⊑-S/∘\S)) ⟩
     S / (((S / (R \ S)) / Q) \ S)
   ≈⟨ /-cong₂ (\-cong₁ //) ⟩
     S / ((S / (Q ⨾ (R \ S))) \ S)
   ≈⟨ S/∘\S∘S/ ⟩
     S / (Q ⨾ (R \ S))
   ≈˘⟨ // ⟩
     (S / (R \ S)) / Q
   □
S/∘\S-into-/-⊒ : {A B C D : Obj} {S : Mor A D} {R : Mor A C} {Q : Mor B C}
                   → ((R / Q) \ S ⊑ Q ⨾ (R \ S))
                   → (S / (R \ S)) / Q ⊑ (S / ((R / Q) \ S))
S/∘\S-into-/-⊒ {A} {B} {C} {D} {S} {R} {Q} assumption = /-universal (⊑-begin
     ((S / (R \ S)) / Q) ⨾ ((R / Q) \ S)
   ≈⟨ ⨾-cong₁ // ⟩
     (S / (Q ⨾ (R \ S))) ⨾ ((R / Q) \ S)
   ⊑⟨ ⨾-monotone₂ assumption ⟩
     (S / (Q ⨾ (R \ S))) ⨾ (Q ⨾ (R \ S))
   ⊑⟨ /-cancel-outer ⟩
     S
   □)
S/∘\S-into-/ : {A B C D : Obj} {S : Mor A D} {R : Mor A C} {Q : Mor B C}
                   → ((R / Q) \ S ⊑ Q ⨾ (R \ S))
                   → (S / ((R / Q) \ S)) ≈ (S / (R \ S)) / Q
S/∘\S-into-/ assumption = ⊑-antisym S/∘\S-into-/-⊑ (S/∘\S-into-/-⊒ assumption)
```



```
/-below-S/∘\S-cancel-outer : {A B C D : Obj} {S : Mor A D} {R : Mor A C} {Q : Mor B C}
                               → (S / ((R / Q) \ S)) ⨾ Q ⊑ S / (R \ S)
/-below-S/∘\S-cancel-outer {A} {B} {C} {D} {S} {R} {Q} = ⊑-begin
     (S / ((R / Q) \ S)) ⨾ Q
   ⊑⟨ ⨾-monotone₁ S/∘\S-into-/-⊑ ⟩
     ((S / (R \ S)) / Q) ⨾ Q
   ⊑⟨ /-cancel-outer ⟩
     S / (R \ S)
   □


\S∘S/-into-\-⊑ : {A B C D : Obj} {S : Mor A D} {R : Mor B D} {Q : Mor B C}
                   → (S / (Q \ R)) \ S ⊑ Q \ ((S / R) \ S)
\S∘S/-into-\-⊑ {A} {B} {C} {D} {S} {R} {Q} = ⊑-begin
     (S / (Q \ R)) \ S
   ⊑⟨ \-antitone (/-antitone (\-monotone ⊑-\S∘S/)) ⟩
     (S / (Q \ ((S / R) \ S))) \ S
   ≈⟨ \-cong₁ (/-cong₂ \\) ⟩
     (S / (((S / R) ⨾ Q) \ S)) \ S
   ≈⟨ \S∘S/∘\S ⟩
     ((S / R) ⨾ Q) \ S
   ≈˘⟨ \\ ⟩
     Q \ ((S / R) \ S)
   □
\S∘S/-into-\-⊒ : {A B C D : Obj} {S : Mor A D} {R : Mor B D} {Q : Mor B C}
                   → (S / (Q \ R) ⊑ (S / R) ⨾ Q)
                   → Q \ ((S / R) \ S) ⊑ (S / (Q \ R)) \ S
\S∘S/-into-\-⊒ {A} {B} {C} {D} {S} {R} {Q} assumption = \-universal (⊑-begin
     (S / (Q \ R)) ⨾ (Q \ ((S / R) \ S))
   ≈⟨ ⨾-cong₂ \\ ⟩
     (S / (Q \ R)) ⨾ (((S / R) ⨾ Q) \ S)
   ⊑⟨ ⨾-monotone₁ assumption ⟩
     ((S / R) ⨾ Q) ⨾ (((S / R) ⨾ Q) \ S)
   ⊑⟨ \-cancel-outer ⟩
     S
   □)
\S∘S/-into-\ : {A B C D : Obj} {S : Mor A D} {R : Mor B D} {Q : Mor B C}
                   → (S / (Q \ R) ⊑ (S / R) ⨾ Q)
                   → (S / (Q \ R)) \ S ≈ Q \ ((S / R) \ S)
\S∘S/-into-\ assumption = ⊑-antisym \S∘S/-into-\-⊑ (\S∘S/-into-\-⊒ assumption)


\-below-\S∘S/-cancel-outer : {A B C D : Obj} {S : Mor A D} {R : Mor B D} {Q : Mor B C}
                               → Q ⨾ ((S / (Q \ R)) \ S) ⊑ (S / R) \ S
\-below-\S∘S/-cancel-outer {A} {B} {C} {D} {S} {R} {Q} = ⊑-begin
     Q ⨾ ((S / (Q \ R)) \ S)
   ⊑⟨ ⨾-monotone₂ \S∘S/-into-\-⊑ ⟩
     Q ⨾ (Q \ ((S / R) \ S))
   ⊑⟨ \-cancel-outer ⟩
     (S / R) \ S
   □
```

The following is mainly an exploration:

```
\S∘S/-⨾\S-⊑ : {A B C D : Obj} {S : Mor A D} {R : Mor A C} {Q : Mor B C}
               → ((R / Q) \ S ⊑ Q ⨾ (R \ S))
               → (S / (Q ⨾ (R \ S))) \ S ⊑ Q ⨾ (R \ S)
```

```
\S∘S/-⨾\S-⊑ {A} {B} {C} {D} {S} {R} {Q} assumption = ⊑-begin
    (S / (Q ⨾ (R \ S))) \ S
  ≈˘⟨ \-cong₁ // ⟩
    ((S / (R \ S)) / Q) \ S
  ⊑⟨ \-antitone (/-monotone ⊑-S/∘\S) ⟩
    (R / Q) \ S
  ⊑⟨ assumption ⟩
    Q ⨾ (R \ S)
  □

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 Categoric.OrderedCategory.Residuals where
open import RATH.Level
open import Categoric.OrderedCategory
open import Categoric.OrderedSemigroupoid.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
      □)

  S/∘\S-S : {A B : Obj} {S : Mor A B} → S / (S \ S) ≈ S
  S/∘\S-S = /-cong₂ (\-cong₁ /-Id) ⟨≈˘≈⟩ S/∘\S∘S/ ⟨≈≈⟩ /-Id
  \S∘S/-S : {A B : Obj} {S : Mor A B} → (S / S) \ S ≈ S
  \S∘S/-S = \-cong₁ (/-cong₂ Id-\) ⟨≈˘≈⟩ \S∘S/∘\S ⟨≈≈⟩ Id-\
```

3.3 Categoric.OSGC.Residuals

```
module Categoric.OSGC.Residuals where
open import RATH.Level
open import Categoric.OSGC
open import Categoric.OrderedSemigroupoid.Residuals
open import RATH.Data.Product using (_,_; proj₁; proj₂)

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 (⨾˘-˘ ⟨≈˘⊑⟩ ˘-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 (˘⨾-˘ ⟨≈˘⊑⟩ ˘-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-⨾-˘M   : {A B C D : Obj} {S : Mor A D} {Q : Mor A B} {F : Mor B C}
                   → isMapping (F ˘) → F ˘ ⨾ (Q \ S) ≈ (Q ⨾ F) \ S
  \-inner-⨾-˘M F˘-isMapping = \-inner-⨾ F˘-isMapping ⟨≈≈⟩ \-cong₁ (⨾-cong₂ ˘˘)
  /-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)
  /-inner-⨾-˘M   : {A B C D : Obj} {S : Mor A D} {R : Mor C D} {F : Mor C B}
                 → isMapping (F ˘) → (S / R) ⨾ F ≈ S / (F ˘ ⨾ R)
  /-inner-⨾-˘M F˘-isMapping = ⨾-cong₂ ˘˘ ⟨≈˘≈⟩ /-inner-⨾ F˘-isMapping

  /-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-isMapping  =  ⊑-antisym \-outer-⨾ (\-outer-⨾-⊒ F-isMapping)
\-outer-⨾-≈-˘M :  {A B C D : Obj} {F : Mor C D} {S : Mor A C} {Q : Mor A B}
                  → isMapping (F ˘) → (Q \ S) ⨾ F ≈ Q \ (S ⨾ F)
\-outer-⨾-≈-˘M F˘-isMapping  =  ⨾-cong₂ ˘˘ ⟨≈˘≈⟩ \-outer-⨾-≈ F˘-isMapping ⟨≈≈⟩ \-cong₂ (⨾-cong₂ ˘˘)

/-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) ˘
    ≈˘⟨ ⨾-˘ ⟩
      ((S / R) ⨾ R) ˘
    ⊑⟨ ˘-monotone /-cancel-outer ⟩
      S ˘
    □))
  (⊑-˘-swap (/-universal (⊑-begin
      (R ˘ \ S ˘) ˘ ⨾ R
    ≈˘⟨ ˘⨾-˘ ⟩
      (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 /-˘)
```

Two variants of the "inner composition laws" that leave the direction of the mapping unchanged:

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


  \-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 ⨾˘-˘ ⟩
      ((S ˘ / Q ˘) ⨾ Q ˘) ˘
    ⊑⟨ ˘-monotone /-cancel-outer ⟩
      (S ˘) ˘
    ≈⟨ ˘˘ ⟩
      S
    □
  ; \-universal = λ {_} {_} {_} {S} {Q} {R} Q⨾R⊑S → ⊑-˘-swap (/-universal (⊑-begin
      R ˘ ⨾ Q ˘
    ≈⟨ ≈-sym ⨾-˘ ⟩
      (Q ⨾ R) ˘
    ⊑⟨ ˘-monotone Q⨾R⊑S ⟩
      S ˘
    □))
  }
```

3.4 Categoric.OSGC.SyQ

```
module Categoric.OSGC.SyQ where
open import RATH.Level
open import Categoric.OSGC
open import Categoric.OSGC.Props.Comp
open import RATH.Data.Product using (_×_; _,_; proj₁; proj₂)
```

For background on **symmetric quotients**, see the material 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 = ⨾˘-˘ ⟨≈˘⊑⟩ ˘-⊑-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
˘╳-cancel-right  :  {A B C : Obj} {Q : Mor B A} {S : Mor A C} → (Q ˘ ╳ S) ⨾ S ˘ ⊑ Q
˘╳-cancel-right = ╳-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) ˘
    ≈˘⟨ ⨾˘-˘ ⟩
      ((Q ╳ S) ⨾ S ˘) ˘
    ⊑⟨ ˘-monotone ╳-cancel-right ⟩
      Q ˘ ˘
    ≈⟨ ˘˘ ⟩
      Q
  □)
  (⊑-begin
      (Q ╳ S) ˘ ⨾ Q ˘
    ≈˘⟨ ⨾-˘ ⟩
      (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 ⨾-˘ ⟩
      (Q ╳ S) ⨾ S ˘ ⨾ P ˘
    ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ╳-cancel-right ⟩
```

```
          Q ˘ ⨾ P ˘
      ≈˘⟨ ⨾-˘ ⟩
          (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-right      :  {A B C : Obj} {Q : Mor A B} {S : Mor C A}
                               → isTotal (Q ╳ S ˘) → (Q ╳ S ˘) ⨾ S ≈ Q ˘
╳˘-total-cancel-right isTot = ⨾-cong₂ ˘˘ ⟨≈˘≈⟩ ╳-total-cancel-right isTot
╳-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))
```

The following proof is mostly intended as documentation for the necessity of the assumptions — the result can be obtained more easily via univalSurj-identity F-unival F-surj together with ╳-cancel-inner.

```
╳-surjFct-shift-right      :  {A B C D : Obj} {F : Mor A B} {Q : Mor B C} {S : Mor B D}
                               → isUnivalent F → isSurjective F → (F ⨾ Q) ╳ (F ⨾ S) ≈ Q ╳ (F ˘ ⨾ F ⨾ S)
╳-surjFct-shift-right {A} {B} {C} {D} {F} {Q} {S} F-unival F-surj = ⊑-antisym
    (╳-universal (⊑-begin
```

```
          Q ⨾ ((F ⨾ Q) ╳ (F ⨾ S))
        ⊑⟨ proj₁ F-surj ⟨⊑≈⟩ ⨾-assoc ⟩
          F ˘ ⨾ F ⨾ Q ⨾ ((F ⨾ Q) ╳ (F ⨾ S))
        ⊑⟨ ⨾-monotone₂ (⨾-assocL ⟨≈⊑⟩ ╳-cancel-left) ⟩
          F ˘ ⨾ F ⨾ S
        □) (⊑-begin
          ((F ⨾ Q) ╳ (F ⨾ S)) ⨾ (F ˘ ⨾ F ⨾ S) ˘
        ≈⟨ ⨾-cong₂ ˘⨾-˘ ⟨≈≈⟩ ⨾-assocL ⟩
          (((F ⨾ Q) ╳ (F ⨾ S)) ⨾ (F ⨾ S) ˘) ⨾ F
        ⊑⟨ ⨾-monotone₁ (╳-cancel-right ⟨⊑≈⟩ ⨾-˘) ⟩
          (Q ˘ ⨾ F ˘) ⨾ F
        ⊑⟨ ⨾-assoc ⟨≈⊑⟩ proj₂ F-unival ⟩
          Q ˘
        □))
   (╳-universal (⊑-begin
          (F ⨾ Q) ⨾ (Q ╳ (F ˘ ⨾ F ⨾ S))
        ⊑⟨ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ ╳-cancel-left ⟩
          F ⨾ F ˘ ⨾ F ⨾ S
        ⊑⟨ ⨾-monotone₂ (⨾-assocL ⟨≈⊑⟩ proj₁ F-unival) ⟩
          F ⨾ S
        □) (⊑-begin
          (Q ╳ (F ˘ ⨾ F ⨾ S)) ⨾ (F ⨾ S) ˘
        ≈⟨ ⨾-cong (╳-cong₂ (⨾-assocL ⟨≈≈⟩ proj₁ (univalSurj-identity F-unival F-surj))) ⨾-˘ ⟩
          (Q ╳ S) ⨾ S ˘ ⨾ F ˘
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ╳-cancel-right ⟨⊑≈˘⟩ ⨾-˘ ⟩
          (F ⨾ Q) ˘
        □))


╳-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 ˘
      ≈˘⟨ ⨾˘-˘ ⟩
        (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
```

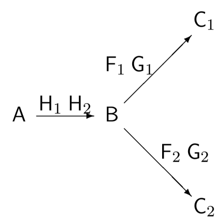


```
        (F ˘ ⨾ ((Q ⨾ F ˘) ╳ S)) ⨾ S ˘
      ⊑⟨ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ ╳-cancel-right ⟩
        F ˘ ⨾ (Q ⨾ F ˘) ˘
      ≈⟨ ⨾-cong₂ ⨾˘-˘ ⟩
        F ˘ ⨾ F ⨾ Q ˘
      ⊑⟨ ⨾-assoc ⟨≈˘⊑⟩ proj₁ isUnival ⟩
        Q ˘
      □)
╳-in-left    : {A B C D : Obj} {Q : Mor C B} {S : Mor C D} {F : Mor A B}
               → 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 ⨾-˘        ⟨≈≈⟩ ⨾-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 ⨾-˘ ⟩
       (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)
```

A related theorem can be seen as moving F_1 and F_2 into a symmetric quotient of two larger relations G_1 and G_2, while retracting the quotient to a different source object A, given appropriate mutual absorption properties relating the new relations H_1 and H_2 with all the old.




```
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 ˘
         □)
```

```
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˘  =  ˘⨾-˘ ⟨≈˘⊑⟩ ˘-monotone left
    right : Q ⨾ Q ˘ ⊑ Q
    right  =  ╳-universal-left Q⊑Q˘╳Q ⟨⊑≈⟩ ˘˘
    right˘ : Q ⨾ Q ˘ ⊑ Q ˘
    right˘  =  ⨾˘-˘ ⟨≈˘⊑⟩ ˘-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 Categoric.OSGC.SyQ.WithResiduals where
open import RATH.Level
open import Categoric.OSGC
open import Categoric.OrderedSemigroupoid.Lattice using (module LocOrdMeet)
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.SyQ

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
    □)
    (⨾˘-˘ ⟨≈˘⊑⟩ ˘-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
    □)
  (⨾˘-˘ ⟨≈˘⊑⟩ ˘-monotone (⊑-begin
      S ⨾ ((P ⨾ Q) ╳ (P ⨾ S)) ˘
    ⊑⟨ ⨾-monotone₂ (╳-˘ ⟨≈⊑⟩ ╳-⊑-\) ⟩
      S ⨾ ((P ⨾ S) \ (P ⨾ Q))
    ⊑⟨ \-cancel-⨾-inner ⟩
      P \ (P ⨾ Q)
    ⊑⟨ \-cancel-inner-⊑-precise incl3 incl4 ⟩
      Q
    □))
```

3.6 Categoric.OCC.SyQ

```
module Categoric.OCC.SyQ where
open import RATH.Level
open import Categoric.OCC
open import Categoric.OSGC.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)
```

Chapter 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

```
open import RATH.Level
open import RATH.Data.Product using (_,_; proj₁; proj₂)
open import Categoric.LESGraph
open import Categoric.Semigroupoid
open import Categoric.OSGC
open import Categoric.MapSG
```

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˘ = ⨾˘-˘ ⟨≈˘≈⟩ ˘-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 : Mor A B} {f : Mapping A ℙB} → (f ≈₁ Λ R ↔ Mapping.mor f ⨾ ∈ ˘ ≈ 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⨾∈˘ ⟨≈˘≈⟩ ⨾˘-˘) ⟨≈≈⟩ ⨾-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:

```
  Idℙ-Id : {I : Mor ℙB ℙB} → isIdentity I → Idℙ₀ ≈ I
  Idℙ-Id {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 ˘
    ≈˘⟨ ⨾˘-˘ ⟩
      (R ⨾ ∈ ˘) ˘
    ⊑⟨ ˘-⊑-swap R⨾∈˘⊑∈˘ ⟩
      ∈
    □))
```

4.2 Categoric.OSGC.PowerOrder

```
open import RATH.Level
open import RATH.Data.Product
open import Categoric.OSGC
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.PowerOp
open import Categoric.Semigroupoid
open import RATH.Data.Product using (proj₁ ; proj₂)


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 ˘ \ ∈ ˘)
Glb0 : {X A : Obj} (R : Mor X (ℙ A)) → Mor X (ℙ A)
Glb0 R = Mapping.mor (Glb 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 : Obj} (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

```
open import RATH.Level
open import RATH.Data.Product using (_,_; proj1; proj2)
open import Categoric.Semigroupoid
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC
open import Categoric.OSGC.PowerOp
```

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 ⇔ ΛS a ⊇ ΛR b ⇔ a (ΛS ⨾ ⊆ ⨾ ΛR ˘) b ⇔ a (ΛS ⨾ Ω˘ ⨾ ΛR ˘) 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 ˘⨾-˘ ⟩
        Λ0 S ⨾ Ω ˘ ⨾ (R ˘ ⨾ Λ0 R) ˘
      ⊑⟨ ⨾-monotone22 (˘-monotone ˘⨾Λ) ⟩
        Λ0 S ⨾ Ω ˘ ⨾ ∈ ˘
      ⊑⟨ ⨾-monotone2 (⨾-˘ ⟨≈˘⊑⟩ ˘-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
          ⊑⟨ ˘⨾-˘ ⟨≈˘⊑⟩ ˘-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

```
open import RATH.Level
open import RATH.Data.Product
open import Categoric.OSGC
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.PowerOp
open import Categoric.Semigroupoid
open import Categoric.MapSG
open import Categoric.LESGraph
open import RATH.Data.Product using (proj₁ ; proj₂)
open import Function using (_∘_)

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; Glb-cocontinuous)
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 $q : \mathbb{P}\,\mathsf{B}$:

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

```
infix 20 _↑ _↓ _↑↓ _↓↑ _↑₀ _↓₀ _↑↓₀ _↓↑₀
_↑ : {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 $\Omega\,\{\mathsf{A}\}$ and $\Omega\,\{\mathsf{B}\}$, 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 ˘) ˘
   ≈˘⟨ ⨾˘-˘ ⟩
     (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 (⨾˘-˘ ⟨≈˘≈⟩ ˘-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 ⨾˘-˘ ⟩
          (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 (⨾˘-˘ ⟨≈˘≈⟩ ˘-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 ⨾˘-˘ ⟩
          (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:

$$\forall\ Q \to \mathsf{Glb}\ Q \fatsemi_1 (R\uparrow) \approx_1 \mathsf{Lub}\ (Q \fatsemi \mathsf{Mapping.mor}\ (R\uparrow))$$

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 (⨾˘-˘ ⟨≈˘≈⟩ ˘-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-restrcocontinuous : {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-restrcocontinuous 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 (⨾˘-˘ ⟨≈˘≈⟩ ˘-cong Λ⨾∈˘ ⟨≈≈⟩ ˘\˘-˘) ⟩
      (∈ / Q) \ R
    ≈⟨ assumption ⟩
      (Q ⨾ (∈ \ R))
    □) ⟩
    Λ₀ (Q ⨾ (∈ \ R))
  ≈˘⟨ Λ-cong (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘) ⟩
    Λ₀ ((Q ⨾ Λ₀ (∈ \ R)) ⨾ ∈ ˘)
  □
```

Essentially the same condition is required for restricted Glb-cocontinuity of R ↓:

```
↓-Glb-restrcocontinuous : {A B : Obj} (R : Mor A B)
  → {X : Obj} (Q : Mor X (ℙ B))
  → (∈ / Q) \ R ˘ ⊑ Q ⨾ (∈ \ R ˘)
  → Glb Q ⨾₁ R ↓ ≈₁ Lub (Q ⨾ R ↓₀)
↓-Glb-restrcocontinuous R 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 (⨾˘-˘ ⟨≈˘≈⟩ ˘-cong Λ⨾∈˘) ⟨≈≈˘⟩ /-˘ ⟩
      (R / (Q ˘ \ ∈ ˘)) ˘
    ≈⟨ /-˘ ⟨≈≈⟩ \-cong₁ ˘\˘-˘ ⟩
      (∈ / Q) \ R ˘
    ≈⟨ ⊑-antisym
      assumption
      (\-universal (⊑-begin
          (∈ / Q) ⨾ Q ⨾ (∈ \ R ˘)
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-outer ⟩
          ∈ ⨾ (∈ \ R ˘)
        ⊑⟨ \-cancel-outer ⟩
          R ˘
        □)) ⟩
      Q ⨾ (∈ \ R ˘)
    ≈˘⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘ ⟩
      (Q ⨾ Λ₀ (∈ \ R ˘)) ⨾ ∈ ˘
    □) ⟩
    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 ⨾˘-˘ ⟩
      (Λ₀ (∈ \ R) ⨾ ∈ ˘) ˘ ⨾ ∈ ˘
    ≈⟨ ⨾-cong₁ (˘-cong Λ⨾∈˘) ⟩
      (∈ \ R) ˘ ⨾ ∈ ˘
    ≈˘⟨ ⨾-˘ ⟩
      (∈ ⨾ (∈ \ R)) ˘
    ⊑⟨ ˘-monotone \-cancel-outer ⟩
      R ˘
    □)
∈⨾↑ : {A B : Obj} {R : Mor A B} → ∈ ⨾ R ↑₀ ⊑ (∈ \ (R ˘)) ˘   -- ≈ ∈ ⨾ R ↓₀ ˘
∈⨾↑ {A} {B} {R} = ⊑-˘-swap (⨾-˘ ⟨≈⊑⟩ ↑˘⨾∈˘)

↑˘⨾∈˘⊑↓⨾∈˘ : {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 ⨾ ∈ ˘
  □
∈⊑∈⨾↑↓˘ : {A B : Obj} {R : Mor A B} → ∈ ⊑ ∈ ⨾ R ↑↓₀ ˘
∈⊑∈⨾↑↓˘ = ˘˘ ⟨≈˘⊑⟩ ˘-monotone ∈˘⊑↑↓⨾∈˘ ⟨⊑≈⟩ ⨾˘-˘
```

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 ⨾-˘) ⟨≈≈⟩ ⨾-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 ∈˘⊑↑↓⨾∈˘ ⟨⊑≈⟩ ⨾˘-˘) ⟩
    (∈ ⨾ (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 ∈˘⊑↓↑⨾∈˘ ⟨⊑≈⟩ ⨾˘-˘) ⟩
    (∈ ⨾ (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 ⨾˘-˘ ⟩
    (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)
∈⨾↑↓˘ = ⨾˘-˘ ⟨≈˘≈⟩ ≈-˘-swap ↑↓⨾∈˘

↑↓⨾∈˘′ : {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 ⨾˘-˘ ⟩
      (Λ₀ (∈ \ R ˘) ⨾ ∈ ˘) ˘ \ R
    ≈⟨ \-cong₁ (˘-cong Λ⨾∈˘) ⟩
      (∈ \ R ˘) ˘ \ R
    ≈⟨ \-cong₁ \˘-˘ ⟩
      (R / ∈ ˘) \ R
    □
```

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

We have antitonicity of R ↑ and R ↓:

```
Ω⨾↑ : {A B : Obj} {R : Mor A B} → Ω ⨾ R ↑₀ ⊑ R ↑₀ ⨾ Ω ˘
Ω⨾↑ {A} {B} {R} = ⊑-begin
    Ω ⨾ R ↑₀
  ⊑⟨ /-universal (\-universal (⊑-begin
      ∈ ⨾ (Ω ⨾ R ↑₀) ⨾ ∈ ˘
    ≈⟨ ⨾-cong₂ (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘) ⟩
      ∈ ⨾ (∈ \ ∈) ⨾ (∈ \ R)
    ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ \-cancel-outer ⟨⊑⊑⟩ \-cancel-outer ⟩
      R
    □)) ⟩
    (∈ \ R) / ∈ ˘
  ≈˘⟨ /-cong₁ Λ⨾∈˘ ⟩
    (Λ₀ (∈ \ R) ⨾ ∈ ˘) / ∈ ˘
  ≈˘⟨ /-outer-⨾-≈ Λ-mapping ⟩
    Λ₀ (∈ \ R) ⨾ (∈ ˘ / ∈ ˘)
  ≈˘⟨ ⨾-cong₂ \-˘ ⟩
    R ↑₀ ⨾ Ω ˘
  □
```

```
Ω⨾↓ : {A B : Obj} {R : Mor A B} → Ω ⨾ R ↓₀ ⊑ R ↓₀ ⨾ Ω ˘
Ω⨾↓ {A} {B} {R} = Ω⨾↑ {B} {A} {R ˘}
```

```
Ω˘⨾↑ : {A B : Obj} {R : Mor A B} → Ω ˘ ⨾ R ↑₀ ⊑ R ↑₀ ⨾ Ω
Ω˘⨾↑ = mappingHom₁-to-L Λ-mapping Ω⨾↑ ⟨⊑≈⟩ ⨾-cong₂ ˘˘
```

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

```
↑⨾↓↑ : {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 ⟨≈⊑⟩ ⨾-monotone₁ (/-universal (⊑-begin
            (∈ ⨾ Λ₀ (∈ \ R)) ⨾ ∈ ˘
          ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘ ⟩
            ∈ ⨾ (∈ \ R)
```

```
            ⊑⟨ \-cancel-outer ⟩
              R
            □)) ⟩
            (R / ∈ ˘) ⨾ ((R / ∈ ˘) \ R)
          ⊑⟨ \-cancel-outer ⟩
            R
          □))
        (⊑-begin
            ∈ \ R
          ≈˘⟨ Λ⨾∈˘ ⟩
            Λ₀ (∈ \ R) ⨾ ∈ ˘
          ⊑⟨ ⨾-monotone₂ ⊑-\S∘S/ ⟩
            Λ₀ (∈ \ R) ⨾ ((R / ∈ ˘) \ R)
          □) ⟩
        ∈ \ R
      □) ⟩
      Λ₀ (∈ \ R)
  ≈⟨⟩
      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
```

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

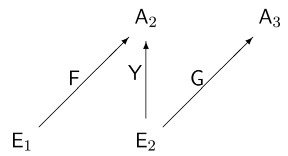
```
↓⨾↑↓′ : {A B₁ B₂ : Obj} {Q : Mor A B₁} {R : Mor A B₂}
    → R / (∈ \ R) ⊑ Q / (∈ \ Q) → Q ↓ ⨾₁ R ↑↓ ≈₁ Q ↓
↓⨾↑↓′ {A} {B₁} {B₂} {Q} {R} p = ≈-begin
    Q ↓₀ ⨾ R ↑↓₀
  ≈⟨ ∈⇒Λ {f = Q ↓ ⨾₁ R ↑↓} (≈-begin
      (Q ↓₀ ⨾ R ↑↓₀) ⨾ ∈ ˘
    ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ↑↓⨾∈˘ ⟩
      Λ₀ (∈ \ (Q ˘)) ⨾ (R / (∈ \ R)) ˘
```



```
    ≈⟨ ⊑-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
  □
```

```
ran↑-≈-ran↓↑ : {A B : Obj} {R : Mor A B} → R ↑₀ ˘ ⨾ R ↑₀ ≈ R ↓↑₀ ˘ ⨾ R ↓↑₀
ran↑-≈-ran↓↑ {A} {B} {R} = ⊑-antisym (⊑-begin
    R ↑₀ ˘ ⨾ R ↑₀
  ≈⟨ ⨾-cong (˘-cong ↑⨾↓↑ ⟨≈˘≈⟩ ⨾-˘) (≈-sym ↑⨾↓↑) ⟨≈≈⟩ ⨾-₂₂assoc₁₂₁ ⟩
    R ↓↑₀ ˘ ⨾ (R ↑₀ ˘ ⨾ R ↑₀) ⨾ R ↓↑₀
  ⊑⟨ ⨾-monotone₂ (proj₁ (Mapping.unival (R ↑))) ⟩
    R ↓↑₀ ˘ ⨾ R ↓↑₀
  □) (⨾-cong₁ ⨾-˘ ⟨≈≈⟩ ⨾-₂₂assoc₁₂₁ ⟨≈⊑⟩ ⨾-monotone₂ (proj₁ (Mapping.unival (R ↓))))
ran↓-≈-ran↑↓ : {A B : Obj} {R : Mor A B} → R ↓₀ ˘ ⨾ R ↓₀ ≈ R ↑↓₀ ˘ ⨾ R ↑↓₀
ran↓-≈-ran↑↓ {A} {B} {R} = ⊑-antisym (⊑-begin
    R ↓₀ ˘ ⨾ R ↓₀
  ≈⟨ ⨾-cong (˘-cong ↓⨾↑↓ ⟨≈˘≈⟩ ⨾-˘) (≈-sym ↓⨾↑↓) ⟨≈≈⟩ ⨾-₂₂assoc₁₂₁ ⟩
    R ↑↓₀ ˘ ⨾ (R ↓₀ ˘ ⨾ R ↓₀) ⨾ R ↑↓₀
  ⊑⟨ ⨾-monotone₂ (proj₁ (Mapping.unival (R ↓))) ⟩
    R ↑↓₀ ˘ ⨾ R ↑↓₀
  □) (⨾-cong₁ ⨾-˘ ⟨≈≈⟩ ⨾-₂₂assoc₁₂₁ ⟨≈⊑⟩ ⨾-monotone₂ (proj₁ (Mapping.unival (R ↑))))
```

For composition of context homomorphisms, we will require Lub-cocontinuity of $\mathsf{G} \downarrow ⨾_1 \mathsf{Y} \uparrow ⨾_1 \mathsf{F} \downarrow$ 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
```

Chapter 5

Internal Order Theory and Direct Powers without Meet

5.1 Categoric.OSGC.Preorder

```
module Categoric.OSGC.Preorder where
open import RATH.Level
open import RATH.Data.Product using (proj₁ ; proj₂ ; _,_)
open import Categoric.OSGC
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.Residuals
open import Relation.Binary.Poset.Galois
open import Relation.Binary.Poset.Dual using (dualPoset)
```

```
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 ˘
    ≈˘⟨ ⨾-˘ ⟩
      (E ⨾ E) ˘
    ⊑⟨ ˘-monotone trans ⟩
      E ˘
    □
idempot : IsIdempotent E
idempot = ⊑-antisym trans rightSupId
˘-idempot : IsIdempotent (E ˘)
˘-idempot = ⊑-antisym ˘-trans ˘-rightSupId
```

More explicitly:

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

Since from Categoric.OCC.Preorder (Sect. 5.3) on, the current OSGC-based IsPreorder concept will be known under the name $\mathsf{IsPreorder}_0$, we already added a subscript “$_0$” to the name here.

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

$$(\forall\, x \bullet f\,x \leqslant g\,x) \quad \Leftrightarrow \quad (\forall\, x, y \bullet g\,x \leqslant z \Rightarrow f\,x \leqslant z)$$

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) ⟨⊑≈⟩ ⨾˘-˘
```

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 ⨾˘-˘ ⟩
        (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 ⨾-˘ ⟩
    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

```
open import RATH.Level
open import RATH.Data.Product using (_,_; proj₁; proj₂)
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OSGC.SyQ.WithResiduals
open import Categoric.OSGC.Preorder
```

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 ⨾˘-˘)) ⟩
    (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 ⨾-˘) ⟩
    (E ˘ ⨾ (E ˘ ⨾ Q ˘)) ╳ E ˘
  ≈⟨ ╳-cong₁ (⨾-assocL ⟨≈≈⟩ ⨾-cong₁ ˘-idempot) ⟩
    lea Q
  □
```

5.2.2 lub and glb

Recall that for the greatest lower bound of a set S we have:

$$
\begin{aligned}
& \mathsf{glb}\ S = s \\
\Leftrightarrow\ & s \text{ is an upper bound of } S \text{ and the least such upper bound} \\
\Leftrightarrow\ & (\forall\ e \mid e \in S \bullet e \leq s) \wedge (\forall\ u \mid (\forall\ e \mid e \in S \bullet e \leq u) \bullet s \leq u) \\
\Leftrightarrow\ & \forall\ u \bullet u \leq s \ \Leftrightarrow\ (\forall\ e \mid e \in S \bullet e \leq u) \\
\Leftrightarrow\ & s\ (\leq \between (\geq / \ni))\ S
\end{aligned}
$$

Considering **glb** as relating sets with elements, we therefore have $S\ \mathsf{glb}\ s \ \Leftrightarrow\ s\ (\leq \between (\geq / \ni))\ S$, that is, $\mathsf{glb} = (\geq / \ni) \between \leq$.

Replacing the converse membership relation ∋ with an arbitrary relation Q, we obtain the following definitions:

```
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
```

Congruence is trivial:

```
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 (⨾-˘ ⟨≈˘≈⟩ ˘-cong ubd-upclosed) ⟩
      ubd Q ˘ ≬ E ˘
   □
```

The "greatest (lower bound)" is the "greatest lower bound":

```
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 (⨾˘-˘ ⟨≈˘≈⟩ ˘-cong lbd-downclosed) ⟩
      lbd Q ˘ ≬ E
   □
```

lub and glb commute with composition with mappings from the left:

```
lub-map⨾ : {I J : Obj} {F : Mor I J} {R : Mor J A} → isMapping F → lub (F ⨾ R) ≈ F ⨾ lub R
lub-map⨾ {I} {J} {F} {R} F-isMapping = ≈-begin
      lub (F ⨾ R)
   ≈⟨⟩
      ubd (F ⨾ R) ˘ ≬ E ˘
   ≈⟨ ≬-cong₁ (˘-cong (Mapping-⨾-ubd F-isMapping) ⟨≈˘≈⟩ ⨾-˘) ⟩
      (ubd R ˘ ⨾ F ˘) ≬ E ˘
   ≈˘⟨ ≬-in-left F-isMapping ⟩
      F ⨾ (ubd R ˘ ≬ E ˘)
   ≈⟨⟩
      F ⨾ lub R
   □
```

```
glb-map⨾ : {I J : Obj} {F : Mor I J} {R : Mor J A} → isMapping F → glb (F ⨾ R) ≈ F ⨾ glb R
glb-map⨾ {I} {J} {F} {R} F-isMapping = ≈-begin
      glb (F ⨾ R)
   ≈⟨⟩
      lbd (F ⨾ R) ˘ ≬ E
   ≈⟨ ≬-cong₁ (˘-cong (Mapping-⨾-lbd F-isMapping) ⟨≈˘≈⟩ ⨾-˘) ⟩
      (lbd R ˘ ⨾ F ˘) ≬ E
   ≈˘⟨ ≬-in-left F-isMapping ⟩
      F ⨾ (lbd R ˘ ≬ E)
   ≈⟨⟩
      F ⨾ glb R
   □
```

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) ˘
          ⊑⟨ ⨾-˘ ⟨≈˘⊑⟩ ˘-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)
   □
```

```
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) ˘
            ⊑⟨ ⨾-˘ ⟨≈˘⊑⟩ ˘-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.

```
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
   □
```

Whenever the least upper bound exists, below it (⨾ E ˘) lie the lower bound of all upper bounds:

```
total-lub-⨾-order˘′ : {I : Obj} {Q : Mor I A} → isTotal (lub Q) → lub Q ⨾ E ˘ ≈ (E ˘ / Q) \ E ˘
total-lub-⨾-order˘′ {I} {Q} lub-total = ≈-begin
      lub Q ⨾ E ˘
   ≈⟨⟩
      ((Q ˘ \ E) ˘ ╳ E ˘) ⨾ E ˘
   ≈⟨ ⊑-antisym (⊑-begin
           ((Q ˘ \ E) ˘ ╳ E ˘) ⨾ E ˘
      ⊑⟨ ⨾-monotone₁ (╳-⊑-\ ⟨⊑≈⟩ \-cong₁ ˘\-˘) ⟩
```



```
              ((E ˘ / Q) \ E ˘) ⨾ E ˘
           ⊑⟨ \-outer-⨾ ⟨⊑⊑⟩ \-monotone ˘-trans ⟩
              (E ˘ / Q) \ E ˘
           □) (⊑-begin
              (E ˘ / Q) \ E ˘
           ⊑⟨ proj₁ lub-total ⟨⊑≈⟩ ⨾-assoc ⟩
              lub Q ⨾ lub Q ˘ ⨾ ((E ˘ / Q) \ E ˘)
           ⊑⟨ ⨾-monotone₂₁ (╳-˘ ⟨≈≈⟩ ╳-cong₂ ˘\-˘ ⟨≈⊑⟩ ╳-⊑-\) ⟩
              lub Q ⨾ (E ˘ \ (E ˘ / Q)) ⨾ ((E ˘ / Q) \ E ˘)
           ⊑⟨ ⨾-monotone₂ \-cancel-middle ⟩
              lub Q ⨾ (E ˘ \ E ˘)
           ≈⟨ ⨾-cong₂ order˘-\ ⟩
              lub Q ⨾ E ˘
           □) ⟩
         (E ˘ / Q) \ E ˘
      □
total-lub-⨾-order˘ : {I : Obj} {Q : Mor I A} → isTotal (lub Q) → lub Q ⨾ E ˘ ≈ lbd (ubd Q)
total-lub-⨾-order˘ lub-total = total-lub-⨾-order˘′ lub-total ⟨≈≈˘⟩ lbd-ubd
order-⨾-total-lub˘ : {I : Obj} {Q : Mor I A} → isTotal (lub Q) → E ⨾ lub Q ˘ ≈ E / (Q ˘ \ E)
order-⨾-total-lub˘ {I} {Q} lub-total = ≈-begin
         E ⨾ lub Q ˘
      ≈⟨ ⨾˘-˘ ⟨≈˘≈⟩ ˘-cong (total-lub-⨾-order˘′ lub-total) ⟩
         ((E ˘ / Q) \ E ˘) ˘
      ≈⟨ \˘-˘ ⟩
         E / (E ˘ / Q) ˘
      ≈⟨ /-cong₂ ˘/-˘ ⟩
         E / (Q ˘ \ E)
      □
```

```
total-glb-⨾-order′ : {I : Obj} {Q : Mor I A} → isTotal (glb Q) → glb Q ⨾ E ≈ (E / Q) \ E
total-glb-⨾-order′ {I} {Q} glb-total = ≈-begin
         glb Q ⨾ E
      ≈⟨⟩
         (lbd Q ˘ ╳ E) ⨾ E
      ≈⟨⟩
         ((Q ˘ \ E ˘) ˘ ╳ E) ⨾ E
      ≈⟨ ⊑-antisym (⊑-begin
              ((Q ˘ \ E ˘) ˘ ╳ E) ⨾ E
           ⊑⟨ ⨾-monotone₁ (╳-⊑-\ ⟨⊑≈⟩ \-cong₁ ˘\˘-˘) ⟩
              ((E / Q) \ E) ⨾ E
           ⊑⟨ \-outer-⨾ ⟨⊑⊑⟩ \-monotone trans ⟩
              (E / Q) \ E
           □) (⊑-begin
              (E / Q) \ E
           ⊑⟨ proj₁ glb-total ⟨⊑≈⟩ ⨾-assoc ⟩
              glb Q ⨾ glb Q ˘ ⨾ ((E / Q) \ E)
           ⊑⟨ ⨾-monotone₂₁ (╳-˘ ⟨≈≈⟩ ╳-cong₂ ˘\˘-˘ ⟨≈⊑⟩ ╳-⊑-\) ⟩
              glb Q ⨾ (E \ (E / Q)) ⨾ ((E / Q) \ E)
           ⊑⟨ ⨾-monotone₂ \-cancel-middle ⟩
              glb Q ⨾ (E \ E)
           ≈⟨ ⨾-cong₂ order-\ ⟩
              glb Q ⨾ E
           □) ⟩
         (E / Q) \ E
      □
total-glb-⨾-order : {I : Obj} {Q : Mor I A} → isTotal (glb Q) → glb Q ⨾ E ≈ ubd (lbd Q)
```

```
total-glb-⨾-order glb-total = total-glb-⨾-order′ glb-total ⟨≈≈˘⟩ ubd-lbd
```

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 Dualisation Experiment

Many of the proofs of this section appear to be 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 — we show this here in module lea-lub, which instantiates the module IsPreorder″ from above with the opposite pre-order, to obtain glb as lub of E ˘, and then similarly obtains the properties of glb from the properties of the lub of E ˘:

```
private
  module lea-lub {A : Obj} {E : Mor A A} (E-isPreorder : IsPreorder osgc E) where
    open IsPreorder osgc E-isPreorder
    open IsPreorder″ E-isPreorder using
      (ubd; lbd
      ; gre; gre-cong; gre-downcone; lub; lub-cong; lea-ubd-≈-lub
      ; lub-map⨾; lub-≈-glb-ubd
      ; total-lub-⨾-order˘′; total-lub-⨾-order˘; total-glb→total-lub
      )
    private module E˘ = IsPreorder″ ˘-isPreorder₀
    lea : {I : Obj} → Mor I A → Mor I A
    lea = E˘.gre
    lea-cong : {I : Obj} {R S : Mor I A} → R ≈ S → lea R ≈ lea S
    lea-cong = E˘.gre-cong
    lea-upcone : {I : Obj} {Q : Mor I A} → lea (Q ⨾ E) ≈ lea Q
    lea-upcone = lea-cong (⨾-cong₂ ˘˘) ⟨≈˘≈⟩ E˘.gre-downcone
    glb : {I : Obj} → Mor I A → Mor I A
    glb = E˘.lub
    glb-cong : {I : Obj} {R S : Mor I A} → R ≈ S → glb R ≈ glb S
    glb-cong = E˘.lub-cong
    gre-lbd-≈-glb : {I : Obj} {Q : Mor I A} → gre (lbd Q) ≈ glb Q
    gre-lbd-≈-glb = ╳-cong (⨾-cong₁ ˘˘) ˘˘ ⟨≈˘≈⟩ E˘.lea-ubd-≈-lub
    glb-map⨾ : {I J : Obj} {F : Mor I J} {R : Mor J A} → isMapping F → glb (F ⨾ R) ≈ F ⨾ glb R
    glb-map⨾ = E˘.lub-map⨾
    glb-≈-lub-lbd : {I : Obj} {Q : Mor I A} → glb Q ≈ lub (lbd Q)
    glb-≈-lub-lbd = E˘.lub-≈-glb-ubd ⟨≈≈⟩ ╳-cong₁ (˘-cong (\-cong₂ ˘˘))
    total-glb-⨾-order˘ : {I : Obj} {Q : Mor I A} → isTotal (glb Q) → glb Q ⨾ E ˘ ≈ lbd Q
    total-glb-⨾-order˘ = E˘.total-lub-⨾-order
    total-glb-⨾-order′ : {I : Obj} {Q : Mor I A} → isTotal (glb Q) → glb Q ⨾ E ≈ (E / Q) \ E
    total-glb-⨾-order′ glbQ-isTotal = ⨾-cong₂ ˘˘ ⟨≈˘≈⟩ E˘.total-lub-⨾-order˘′ glbQ-isTotal ⟨≈≈⟩ \-cong (/-cong₁ ˘˘) ˘˘
    total-glb-⨾-order : {I : Obj} {Q : Mor I A} → isTotal (glb Q) → glb Q ⨾ E ≈ ubd (lbd Q)
```



```
    total-glb-⨾-order glbQ-isTotal = ⨾-cong₂ ˘˘ ⟨≈˘≈⟩ E˘.total-lub-⨾-order˘ glbQ-isTotal ⟨≈≈⟩ \-cong₂ ˘˘
    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 = E˘.total-glb→total-lub (≈-isTotal (╳-cong₁ (˘-cong (\-cong₂ ˘˘))) lub-total)
```

The expected result of dualisation is that occurrences of the order are changed into occurrences of the converse of the order and vice versa, and also that ubd is swapped with lbd, and similarly gre with lea and lub with glb. However, due to the fact that the involution property ˘˘ of converse is not a definitional equality, the dualised types of the original properties do not exactly match these expectations in about half of the cases here, but need explicit ˘˘-adaptations. Those that do not need adaptation could be handled via **renaming**, but we prefer to make their types explicit and checked here.

Even without this adaptation problem, dualisation is not completely straight-forward, since properties, like lub-≈-glb-ubd, that involve concepts that one wishes to introduce only via dualisation. need to be defined in a context where the relevant dualised concept (here glb) is already available. Where one wants to rely on dualisation in such circumstances, it is therefore necessary to define a hierarchy of modules, alternating definitions with dualisations, and importing both variants of earlier modules in later modules.

Besides the overhead introduced by these adaptations, using the proof components contained in ˘-isPreorder$_0$ occasionally produces "detours" that the direct proofs can avoid.

We counted the lines of the normalisations (as pretty-printed by interactive Agda) of the items defined in **module** lea-lub via dualisation and of the corresponding direct definitions in IsPreorder″, and just on these few examples observe an overhead of up to 40%. (These line counts appear to be a reasonable approximation of (proof) term size, see page 132 for more information.)

name	direct (IsPreorder″)	dualised (lea-lub)	overhead
lea-cong	19	19	
lea-upcone	169	239	41%
glb	12	16	33%
glb-cong	1058	1058	
gre-lbd-≈-glb	230	227	
glb-map⨾	787	787	
glb-≈-lub-lbd	2568	2983	16%
total-glb-⨾-order˘	347	370	7%
total-glb-⨾-order	1498	2101	40%
total-lub→total-glb	10408	12158	17%

5.3 Categoric.OCC.Preorder

```
module Categoric.OCC.Preorder where
open import RATH.Level
open import RATH.Data.Product using (proj₁; proj₂; _,_)
open import Categoric.OCC
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
```

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

Since we overload some of the names used in the OSGC context, but still need access to the original concepts here, we rename those by adding a subscript "$_0$":

```
open import Categoric.OSGC.Preorder using () renaming
(IsPreorder to IsPreorder₀
; module IsPreorder to IsPreorder₀
; module PreorderWithResiduals to OSGC-PreorderWithResiduals
)
```

In the remainde of this mooodule we assume an arbitrary, but fixed, OCC:

```
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}
```

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

```
    order-isTotalI : isTotalI E
    order-isTotalI = ˘-refl ⟨⊑⊑⟩ (leftId ⟨≈˘⊑⟩ ⨾-monotone₁ refl)
    order-isTotal : isTotal E
    order-isTotal = isTotal-from-I order-isTotalI
    order˘-isTotalI : isTotalI (E ˘)
    order˘-isTotalI = ˘-refl ⟨⊑⊑⟩ (rightId ⟨≈˘⊑⟩ ⨾-monotone₂ (refl ⟨⊑≈˘⟩ ˘˘))
    order˘-isTotal : isTotal (E ˘)
    order˘-isTotal = isTotal-from-I order˘-isTotalI
```

5.3.1 Retract Preorder and Preorder Invariance

If the relation $\_\preccurlyeq\_$ is a preorder, then so is, for any mapping f, the relation $\_\preccurlyeq'\_$ defined by $x \preccurlyeq' y \;=\; f(x) \preccurlyeq f(y)$. 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 ⨾-˘ ⟩
      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 reflexivity 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 reflexivity.

5.4 Categoric.OCC.Order

```
open import RATH.Level
open import RATH.Data.Product using (_,_; proj₁; proj₂)
open import Categoric.OCC
open import Categoric.OCC.Preorder using
  (IsPreorder; module IsPreorder; module IsPreorder₂; module PreorderWithResiduals
  ; retractPreorder
  )
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OSGC.SyQ.WithResiduals
open import Categoric.OCC.SyQ
```

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
    ; order-isTotalI; order-isTotal; order˘-isTotalI; order˘-isTotal
    )
  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:

```
private   -- preliminary formalisation
  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
    □
```

If we consider instead different indirect inclusions in the above motivating derivation, say the first from the left and the second from the right,

$$(\forall\ z \bullet z \leqslant f\,(x) \;\Leftrightarrow\; z \leqslant y) \;\Leftrightarrow\; (\forall\ z \bullet g\,(x) \leqslant z \;\Leftrightarrow\; y \leqslant z)\ ,$$

then the resulting formalization is as follows:

```
private   -- preliminary formalisation
  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:

```
private   -- preliminary formalisation
  ˘-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

$$(\forall\ x \bullet f\,x \approx g\,x) \;\Leftrightarrow\; (\forall\ x, z \bullet f\,(x) \leqslant z \;\Leftrightarrow\; g\,(x) \leqslant z)$$

— 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˘ = ⨾-˘ ⟨≈˘≈⟩ ˘-cong indir ⟨≈≈⟩ ⨾-˘
```

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
  (⨾˘-˘ ⟨≈˘≈⟩ (˘-cong indir ⟨≈≈⟩ ⨾˘-˘))
```


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) ⟨≈≈⟩ ⨾-˘) ⟩
    (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) ⟨≈≈⟩ ⨾˘-˘) ⟩
    (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, substitutivity 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 an injective mapping F and an order _≼_, we again have an order, which in the pointwise case would be defined by x ≼′ y :⇔ F x ≼ F y.

```
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 (⊓-M-in-right 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 ⨾-˘ ⨾˘-˘ ⟨≈≈⟩ ⨾-assoc) ⟩
            F0    ⨾ E ˘ ⨾ F0 ˘ ⨾ F0 ⨾ E ˘
          ⊑⟨ ⨾-monotone22 (⨾-assocL ⟨≈⊑⟩ proj1 F-unival) ⟩
            F0    ⨾ E ˘ ⨾ E ˘
          ⊑⟨ ⨾-monotone2 ˘-trans ⟩
            F0 ⨾ E ˘
          ≈˘⟨ ⨾˘-˘ ⟩
            (E ⨾ F0 ˘) ˘
          □)
        ⟩
        (E ⨾ F0 ˘) ⊓ (E ⨾ F0 ˘)
      ≈⟨ ⊓-in-left F-isM ⟨≈˘≈˘⟩ ⨾-cong2 (⊓-M-in-right 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 ˘
      ⊑⟨ ⨾-˘ ⟨≈˘⊑⟩ ˘-monotone \-cancel-outer ⟩
        R ˘
      □)
```

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

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

Formally, and generally,

```
  wrap : Mor A B
  wrap = Id ⨉ R
  wrap-isInjectiveI : isInjectiveI wrap
  wrap-isInjectiveI =  ⊑-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 = ⨾˘-˘ ⟨≈˘≈⟩ ˘-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’. For convenience, we also dualise it:

```
  /R˘-⨾R˘ : {C : Obj} {Q : Mor C A} → isTotal wrap → (Q / R ˘) ⨾ R ˘ ≈ Q
  /R˘-⨾R˘ {C} {Q} total = ≈-begin
      (Q / R ˘) ⨾ R ˘
    ≈˘⟨ ⨾˘-˘ ⟩
      (R ⨾ (Q / R ˘) ˘) ˘
    ≈⟨ ˘-cong (⨾-cong₂ /˘-˘) ⟩
      (R ⨾ (R \ Q ˘)) ˘
    ≈⟨ ˘-cong (R⨾-R\ total) ⟨≈≈⟩ ˘˘ ⟩
      Q
    □
```

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˘ = \-˘
  Ω˘⨾∊˘-⊑ : Ω ˘ ⨾ R ˘ ⊑ R ˘
  Ω˘⨾∊˘-⊑ = ⨾-˘ ⟨≈˘⊑⟩ ˘-monotone \-cancel-outer
  R⨾-R⨉ : {C : Obj} {Q : Mor A C} → R ⨾ (R ⨉ Q) ≈ Q
  R⨾-R⨉ = ⨉-surjective-cancel-left (isSurjectiveFromTotal (≈-isTotal ⨉-˘ comprehensive))
  ⨉R-⨾R˘ : {C : Obj} {Q : Mor A C} → (Q ⨉ R) ⨾ R ˘ ≈ Q ˘
  ⨉R-⨾R˘ = ⨉-total-cancel-right 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 ⨾˘-˘) \-˘ ⟩
    (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}
lub-⨾-Ω : {I : Obj} {Q : Mor I B} → lub Q ⨾ Ω ≈ ubd Q
lub-⨾-Ω = total-lub-⨾-order lubΩ-total
```

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

```
glbΩ≈⨡ : {I : Obj} {Q : Mor I B} → glb Q ≈ (lbd Q) ˘ ⨡ Ω
glbΩ≈⨡ = ≈-refl

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}
glb-⨾-Ω˘ : {I : Obj} {Q : Mor I B} → glb Q ⨾ Ω ˘ ≈ lbd Q
glb-⨾-Ω˘ = total-glb-⨾-order˘ 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 ⨾-˘) ⟩
    (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

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

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:

$$\forall\ \{Q\ f\} \to \mathsf{isMapping}\ f \to f_0 ⨾ R\,\breve{} \approx Q \;\Leftrightarrow\; f_0 \approx \Lambda_0\,Q$$

```
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 ⟨≈˘≈⟩ ⨾˘-˘) ⟩
    (R ⨾ f₀ ˘) ⨡ R
  ≈˘⟨ lubΩ≈⨡ ⟩
    lub f₀
  ≈⟨ lub-mapping (Mapping.prf f) ⟩
    f₀
  □) where f₀ = Mapping.mor f
```

5.5 Categoric.OCC.DirectPower

```
open import RATH.Level
open import RATH.Data.Product using (_,_; proj₁; proj₂)
open import Categoric.OCC
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OSGC.SyQ.WithResiduals
open import Categoric.OCC.SyQ
open import Categoric.OCC.Preorder using (module PreorderWithResiduals)
```

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-isInjectiveI to 𝒮-isInjectiveI
      )
    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          ; ˘-refl           to Ω˘-refl
      ; trans         to Ω-trans         ; ˘-trans          to Ω˘-trans
      ; idempot       to Ω-idempot       ; ˘-idempot        to Ω˘-idempot
      ; ˘-isOrder     to Ω˘-isOrder
      ; antisym≈      to Ω╳Ω             ; ˘-antisym≈       to Ω˘╳Ω˘
      ; order-\       to Ω\Ω≈Ω           ; order-/          to Ω/Ω≈Ω
      ; order-isTotalI to Ω-isTotalI     ; order˘-isTotalI  to Ω˘-isTotalI
      ; order-isTotal to Ω-isTotal       ; order˘-isTotal   to Ω˘-isTotal
      ; 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 ∈⨾-∈╳       -- {C : Obj} {Q : Mor A C} → ∈ ⨾ (∈ ╳ Q) ≈ Q
      ; ╳R-⨾R˘ to ╳∈-⨾∈˘    -- {C : Obj} {Q : Mor A C} → (Q ╳ ∈) ⨾ ∈ ˘ ≈ Q
      )
      hiding (Ω; lub-⨾wrap; lub-/R˘)
    ╳∈-univalentI : {A : Obj} {Q : Mor X A} → isUnivalentI (Q ╳ ∈)
    ╳∈-univalentI {A} {Q} = ⊑-begin
        (Q ╳ ∈) ˘ ⨾ (Q ╳ ∈)
      ≈⟨ ⨾-cong₁ ╳-˘ ⟩
        (∈ ╳ Q) ⨾ (Q ╳ ∈)
      ⊑⟨ ╳-cancel-middle ⟩
        ∈ ╳ ∈
      ⊑⟨ ∈-extensional ⟩
        Id
      □
    ╳∈-univalent : {A : Obj} {Q : Mor X A} → isUnivalent (Q ╳ ∈)
    ╳∈-univalent = isUnivalent-from-I ╳∈-univalentI
    ╳∈-isMapping : {A : Obj} {Q : Mor X A} → isMapping (Q ╳ ∈)
    ╳∈-isMapping    = ╳∈-univalent, ∈-comprehensive
    ╳∈-isMappingI : {A : Obj} {Q : Mor X A} → isMappingI (Q ╳ ∈)
    ╳∈-isMappingI   = ╳∈-univalentI, ∈-comprehensiveI
```

The following is a special case of (Furusawa and Kahl, 1998, Lemma 6.8):

```
    \∈⨾∈\-reflexive : {A : Obj} {Q : Mor X A} → Id ⊑ (Q \ ∈) ⨾ (∈ \ Q)
    \∈⨾∈\-reflexive {A} {Q} = ⊑-begin
        Id
      ⊑⟨ ∈-comprehensiveI ⟩
        (Q ╳ ∈) ⨾ (Q ╳ ∈) ˘
      ≈⟨ ⨾-cong₂ ╳-˘ ⟩
        (Q ╳ ∈) ⨾ (∈ ╳ Q)
      ⊑⟨ ⨾-monotone ╳-⊑-\ ╳-⊑-\ ⟩
        (Q \ ∈) ⨾ (∈ \ Q)
      □

    open \-Preorder′ ∈ using (wrap⨾R˘; R⨾wrap˘; wrap⨾E; R⨾-R\; /R˘-⨾R˘)
    𝒮 : Mapping X ℙX
    𝒮 = record {mor = 𝒮₀; prf = ╳∈-isMapping}
    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)
    ∈-isTotalI : isTotalI ∈
    ∈-isTotalI = ≈-isTotalI (≈-sym 𝒮⨾Ω) (⨾-isTotalI ∈-comprehensiveI Ω-isTotalI)
    ∈-isTotal : isTotal ∈
    ∈-isTotal = isTotal-from-I ∈-isTotalI
    ∈⨾-∈\ : {Y : Obj} {R : Mor X Y} → ∈ ⨾ (∈ \ R) ≈ R
    ∈⨾-∈\ = R⨾-R\ (mappingTotal 𝒮)
    /∈˘-⨾∈˘ : {Y : Obj} {S : Mor Y X} → (S / ∈ ˘) ⨾ ∈ ˘ ≈ S
    /∈˘-⨾∈˘ = /R˘-⨾R˘ (mappingTotal 𝒮)
    ∈\-\-∈\ : {Y Z : Obj} {R : Mor X Y} {S : Mor X Z} → (∈ \ R) \ (∈ \ S) ≈ R \ S
```

```
∈\-\-∈\ {Y} {Z} {R} {S}  =  ≈-begin
        (∈ \ R) \ (∈ \ S)
      ≈⟨ \\ ⟩
        (∈ ⨾ (∈ \ R)) \ S
      ≈⟨ \-cong₁ ∈⨾-∈\ ⟩
        R \ S
      □
/∈˘-/-/∈˘ : {Y Z : Obj} {R : Mor Y X} {S : Mor Z X} → (R / ∈ ˘) / (S / ∈ ˘) ≈ R / S
/∈˘-/-/∈˘ {Y} {Z} {R} {S}  =  ≈-begin
        (R / ∈ ˘) / (S / ∈ ˘)
      ≈⟨ // ⟩
        R / ((S / ∈ ˘) ⨾ ∈ ˘)
      ≈⟨ /-cong₂ /∈˘-⨾∈˘ ⟩
        R / S
      □
open \-OrderWithComprehension Ω-isOrder ∈-comprehensivel using
  (lub-⨾wrap; lub-/R˘; lubΩ≈≬)
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-map⨾ : {I J : Obj} {F : Mor I J} {R : Mor J X} → isMapping F → Λ₀ (F ⨾ R) ≈ F ⨾ Λ₀ R
Λ-map⨾ {I} {J} {F} {R} F-isMapping  =  ≈-begin
        Λ₀ (F ⨾ R)
      ≈⟨⟩
        (F ⨾ R) ˘ ≬ ∈
      ≈⟨ ≬-cong₁ ⨾-˘ ⟩
        (R ˘ ⨾ F ˘) ≬ ∈
      ≈˘⟨ ≬-in-left F-isMapping ⟩
        F ⨾ (R ˘ ≬ ∈)
      □


Λ⨾-∈\ : {I J : Obj} {R : Mor I X} {S : Mor X J} → Λ₀ R ⨾ (∈ \ S) ≈ R ˘ \ S
Λ⨾-∈\ {I} {J} {R} {S}  =  ≈-begin
        Λ₀ R ⨾ (∈ \ S)
      ≈⟨ \-inner-⨾ Λ-isMapping ⟨≈≈˘⟩ \-cong₁ ⨾˘-˘ ⟩
        (Λ₀ R ⨾ ∈ ˘) ˘ \ S
      ≈⟨ \-cong₁ (˘-cong ≬∈-⨾∈˘ ⟨≈≈⟩ ˘˘) ⟩
```



```
        R ˘ \ S
      □
Λ-˘-⨾-∈\ : {I J : Obj} {R : Mor X I} {S : Mor X J} → Λ₀ (R ˘) ⨾ (∈ \ S) ≈ R \ S
Λ-˘-⨾-∈\  =  Λ⨾-∈\ ⟨≈≈⟩ \-cong₁ ˘˘


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}}


∈˘\-/∈˘\     : {B C D : Obj} {S₁ : Mor B C} {S₂ : Mor B D}
                 → ∈ ˘ \ ((S₁ / ∈ ˘) \ S₂) ≈ S₁ \ S₂
∈˘\-/∈˘\     {B} {C} {D} {S₁} {S₂}  =  ≈-begin
      ∈ ˘ \ ((S₁ / ∈ ˘) \ S₂)
    ≈⟨ \\ ⟩
      ((S₁ / ∈ ˘) ⨾ ∈ ˘) \ S₂
    ≈⟨ \-cong₁ /∈˘-⨾∈˘ ⟩
      S₁ \ S₂
    □
/∈\-/∈     : {A B D : Obj} {S₁ : Mor A D} {S₂ : Mor B D}
               → (S₁ / (∈ \ S₂)) / ∈  ≈ S₁ / S₂
/∈\-/∈ {A} {B} {D} {S₁} {S₂}  =  ≈-begin
      (S₁ / (∈ \ S₂)) / ∈
    ≈⟨ // ⟩
      S₁ / (∈ ⨾ (∈ \ S₂))
    ≈⟨ /-cong₂ ∈⨾-∈\ ⟩
      S₁ / S₂
    □


/∈˘\-⊑-→-\-⊑     : {A B C D : Obj} {R₁ : Mor A C} {R₂ : Mor A D} {S₁ : Mor B C} {S₂ : Mor B D}
                     → (R₁ / ∈ ˘) \ R₂ ⊑ (S₁ / ∈ ˘) \ S₂
                     → R₁ \ R₂          ⊑ S₁ \ S₂
/∈˘\-⊑-→-\-⊑ {A} {B} {C} {D} {R₁} {R₂} {S₁} {S₂} assumption  =  ⊑-begin
      R₁ \ R₂
    ≈˘⟨ ∈˘\-/∈˘\ ⟩
      ∈ ˘ \ ((R₁ / ∈ ˘) \ R₂)
    ⊑⟨ \-monotone assumption ⟩
      ∈ ˘ \ ((S₁ / ∈ ˘) \ S₂)
    ≈⟨ ∈˘\-/∈˘\ ⟩
```

```
        S₁ \ S₂
      □

/∈\-⊑-→-/-⊑    : {A B C D : Obj} {R₁ : Mor A C} {R₂ : Mor B C} {S₁ : Mor A D} {S₂ : Mor B D}
                  → R₁ / (∈ \ R₂) ⊑ S₁ / (∈ \ S₂)
                  →       R₁ / R₂ ⊑ S₁ / S₂
/∈\-⊑-→-/-⊑ {A} {B} {C} {D} {R₁} {R₂} {S₁} {S₂} assumption = ⊑-begin
        R₁ / R₂
      ≈˘⟨ /∈\-/∈ ⟩
        (R₁ / (∈ \ R₂)) / ∈
      ⊑⟨ /-monotone assumption ⟩
        (S₁ / (∈ \ S₂)) / ∈
      ≈⟨ /∈\-/∈ ⟩
        S₁ / S₂
      □


∈˘\-⊑-∈⨾ : {A B : Obj} {R : Mor (ℙ A) B} → ∈ ˘ \ R ⊑ ∈ ⨾ R
∈˘\-⊑-∈⨾ = proj₁ ∈-isTotal ⟨⊑≈⟩ ⨾-assoc ⟨⊑⊑⟩ ⨾-monotone₂ \-cancel-outer
/∈-⊑-⨾∈˘ : {A B : Obj} {R : Mor A (ℙ B)} → R / ∈ ⊑ R ⨾ ∈ ˘
/∈-⊑-⨾∈˘ = proj₂ ∈-isTotal ⟨⊑≈⟩ ⨾-assocL ⟨⊑⊑⟩ ⨾-monotone₁ /-cancel-outer
\-⊑-∈⨾-/∈˘\    :    {B C D : Obj} {S₁ : Mor B C} {S₂ : Mor B D}
                  → S₁ \ S₂ ⊑ ∈ ⨾ ((S₁ / ∈ ˘) \ S₂)
\-⊑-∈⨾-/∈˘\     {B} {C} {D} {S₁} {S₂} = ⊑-begin
        S₁ \ S₂
      ≈˘⟨ ∈˘\-/∈˘\ ⟩
        ∈ ˘ \ ((S₁ / ∈ ˘) \ S₂)
      ⊑⟨ ∈˘\-⊑-∈⨾ ⟩
        ∈ ⨾ ((S₁ / ∈ ˘) \ S₂)
      □
/-⊑-/∈\-⨾∈˘    : {A B D : Obj} {S₁ : Mor A D} {S₂ : Mor B D}
                → S₁ / S₂ ⊑ (S₁ / (∈ \ S₂)) ⨾ ∈ ˘
/-⊑-/∈\-⨾∈˘ {A} {B} {D} {S₁} {S₂} = ⊑-begin
        S₁ / S₂
      ≈˘⟨ /∈\-/∈ ⟩
        (S₁ / (∈ \ S₂)) / ∈
      ⊑⟨ /∈-⊑-⨾∈˘ ⟩
        (S₁ / (∈ \ S₂)) ⨾ ∈ ˘
      □
```

In Categoric.OSGC.PowerOrder (Sect. 4.2), we have definitions of Lub and Glb with respect to the subset order Ω, defined directly using Λ; we can now show that these coincide with the Ω-specialisations of our order-theoretic lub and glb:

```
open import Categoric.OSGC.PowerOrder osgc        leftResOp rightResOp powerOp
    using (Lub₀; Glb₀)
Lub≈lub : {I X : Obj} {Q : Mor I (ℙ X)} → Lub₀ Q ≈ lub Q
Lub≈lub {I} {X} {Q} = ≬-cong₁ ⨾˘-˘ ⟨≈≈˘⟩ lubΩ≈≬


Glb≈glb : {I X : Obj} {Q : Mor I (ℙ X)} → Glb₀ Q ≈ glb Q
Glb≈glb {I} {X} {Q} = ⊑-antisym (≬-universal (\-universal (⊑-begin
          ∈ ⨾ (Q ˘ \ Ω ˘) ˘ ⨾ ((Q ˘ \ ∈ ˘) ˘ ≬ ∈)
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ (⨾˘-˘ ⟨≈˘⊑⟩ ˘-monotone \-outer-⨾) ⟩
          (Q ˘ \ (Ω ˘ ⨾ ∈ ˘)) ˘ ⨾ ((Q ˘ \ ∈ ˘) ˘ ≬ ∈)
        ⊑⟨ ⨾-monotone₁ (˘-monotone (\-monotone Ω˘⨾∈˘-⊑)) ⟨⊑⊑⟩ ≬-cancel-left ⟩
          ∈
```



```
□))(⊑-begin
    ((Q ˘ \ ∈ ˘) ˘ ≬ ∈) ⨾ (∈ \ ∈) ˘
⊑⟨ \-universal (/-universal (⊑-begin
      (Q ˘ ⨾ ((Q ˘ \ ∈ ˘) ˘ ≬ ∈) ⨾ (∈ \ ∈) ˘) ⨾ ∈ ˘
    ⊑⟨ ⨾-assoc₃₊₁ ⟨≈⊑⟩ ⨾-monotone₂₂ (⨾-˘ ⟨≈˘⊑⟩ ˘-monotone \-cancel-outer) ⟩
      Q ˘ ⨾ ((Q ˘ \ ∈ ˘) ˘ ≬ ∈) ⨾ ∈ ˘
    ⊑⟨ ⨾-monotone₂ ˘≬-cancel-right ⟩
      Q ˘ ⨾ (Q ˘ \ ∈ ˘)
    ⊑⟨ \-cancel-outer ⟩
      ∈ ˘
    □)) ⟩
    Q ˘ \ (∈ ˘ / ∈ ˘)
≈˘⟨ ˘˘ ⟨≈≈⟩ \-cong₂ \-˘ ⟩
    ((Q ˘ \ (∈ \ ∈) ˘) ˘) ˘
□)) (≬-universal (⊑-begin
    (Q ˘ \ ∈ ˘) ˘ ⨾ ((Q ˘ \ (∈ \ ∈) ˘) ˘ ≬ (∈ \ ∈))
⊑⟨ ⨾-monotone₁ (⊑-begin
      (Q ˘ \ ∈ ˘) ˘
    ≈⟨ ˘\˘-˘ ⟩
      ∈ / Q
    ≈˘⟨ ∈⨾-∈\ ⟩
      ∈ ⨾ (∈ \ (∈ / Q))
    ⊑⟨ ⨾-monotone₂ \/-⊑ ⟩
      ∈ ⨾ ((∈ \ ∈) / Q)
    ≈˘⟨ ⨾-cong₂ ˘\˘-˘ ⟩
      ∈ ⨾ (Q ˘ \ (∈ \ ∈) ˘) ˘
    □) ⟩
    (∈ ⨾ (Q ˘ \ (∈ \ ∈) ˘) ˘) ⨾ ((Q ˘ \ (∈ \ ∈) ˘) ˘ ≬ (∈ \ ∈))
⊑⟨ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ ≬-cancel-left ⟩
    ∈    ⨾ (∈ \ ∈)
⊑⟨ \-cancel-outer ⟩
    ∈
□)(⊑-begin
    ((Q ˘ \ (∈ \ ∈) ˘) ˘ ≬ (∈ \ ∈)) ⨾ ∈ ˘
≈⟨ ⨾-cong₂ (˘-cong ∈⨾-∈\ ⟨≈˘≈⟩ ⨾-˘) ⟩
    ((Q ˘ \ (∈ \ ∈) ˘) ˘ ≬ (∈ \ ∈)) ⨾ (∈ \ ∈) ˘ ⨾ ∈ ˘
⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ˘≬-cancel-right ⟩
    (Q ˘ \ (∈ \ ∈) ˘) ⨾ ∈ ˘
⊑⟨ \-outer-⨾ ⟨⊑⊑⟩ \-monotone Ω˘⨾∈˘-⊑ ⟩
    Q ˘ \ ∈ ˘
≈˘⟨ ˘˘ ⟩
    ((Q ˘ \ ∈ ˘) ˘) ˘
□))
```

Chapter 6

Internal Galois Connections

In the first two sections we define internal closure operators, that is, closure operators with respect to an OCC morphism E satisfying IsOrder E. 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.

In sections 6.3 to 6.5 we define internal Galois connections and prove some of their properties.

We then use the fact that the polarities _↑ and _↓ form Galois connections to derive a number of properties of polarities by instantiating our Galois modules appropiately in sections 6.6 to 6.9.

6.1 Categoric.OSGC.Preorder.Closure

```
open import RATH.Level
open import RATH.Data.Product using (proj₁ ; proj₂ ; _,_)
open import Categoric.OSGC
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.Preorder
module Categoric.OSGC.Preorder.Closure
  {i j k₁ k₂} {Obj : Set i} (osgc : OSGC j k₁ k₂ Obj) where
open OSGC osgc
```

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

$$\forall x, y \bullet x \leqslant C(x) \Leftrightarrow C(x) \leqslant C(y)$$

— 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 ˘
    ≈˘⟨ ⨾˘-˘ ⟩
      (E ⨾ C ˘) ˘
    ≈⟨ ˘-cong char ⟩
      (C ⨾ E ⨾ C ˘) ˘
    ≈⟨ ⨾-˘ ⟨≈≈⟩ (⨾-cong₁ ⨾˘-˘ ⟨≈≈⟩ ⨾-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 superidentities:

```
  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
    ⊑⟨ proj_2 EC˘-supId ⟩
      E ⨾ E ⨾ C ˘
    ≈⟨ ⨾-assoc ⟨≈˘≈⟩ ⨾-cong_1 A.idempot ⟩
      E ⨾ C ˘
    □
```

Consequently, we have the combinators,

```
  EC-⊑-E : E ⨾ C ⊑ E
  EC-⊑-E = ⨾-monotone_2 increasing ⟨⊑≈⟩ A.idempot
  CE-⊑-E : C ⨾ E ⊑ E
  CE-⊑-E = ⨾-monotone_1 increasing ⟨⊑≈⟩ A.idempot
```

6.1.2 Quasi-idempotency

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
    ⊑⟨ ⨾-monotone_2 (proj_1 EC˘-supId) ⟩
      C ⨾ (E ⨾ C ˘) ⨾ C ⨾ E
    ≈⟨ (⨾-assoc ⟨≈˘≈˘⟩ (⨾-cong_1 char)) ⟨≈≈⟩ ⨾-assoc      ⟩
      E ⨾ C ˘ ⨾ C ⨾ E
    ⊑⟨ ⨾-assoc_{3+1} ⟨≈˘⊑⟩ (⨾-monotone_1 EC˘C-⊑-CE ⟨⊑≈⟩ (⨾-assoc ⟨≈≈⟩ ⨾-cong_2 A.idempot)) ⟩
      C ⨾ E
    □) (⊑-begin
      C ⨾ E
    ⊑⟨ proj_1 CEC˘-supId ⟩
      (C ⨾ E ⨾ C ˘) ⨾ C ⨾ E
    ⊑⟨ (⨾-assoc ⟨≈≈⟩ ⨾-cong_2 (⨾-assoc ⟨≈≈˘⟩ ⨾-assoc_{3+1})) ⟨≈⊑⟩ ⨾-monotone_21 EC˘C-⊑-CE ⟩
      C ⨾ (C ⨾ E) ⨾ E
    ≈⟨ ⨾-cong_2 ⨾-assoc ⟨≈≈⟩ ⨾-cong_22 A.idempot ⟩
      C ⨾ C ⨾ E
    □)
```

6.1.3 Monotonicity

Finally, we show that the characterization yields monotonicity. Recall that monotocity can take a number of different forms — needless to say, the notions coincide since C is a mapping:

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



```
  ⇔  C ˘ ⨾ ≼  ⊑  ≼ ⨾ C ˘
  ⇔  ≼ ˘ ⨾ C  ⊑  C ⨾ ≼ ˘
```

The final form is the so called 'L-simulation'; its converse, i.e., the second-last form, will be our starting point. The third-last form, since it is the shortest, will be called just "monotone".

```
  monotoneL˘ : C ˘ ⨾ E ⊑ E ⨾ C ˘
  monotoneL˘ = ⊑-begin
      C ˘ ⨾ E
    ⊑⟨ ⨾-monotone_2 (proj_2 EC˘-supId) ⟩
      C ˘ ⨾ E ⨾ E ⨾ C ˘
    ≈⟨ ⨾-cong_2 (⨾-assoc ⟨≈˘≈⟩ ⨾-cong_1 A.idempot) ⟩
      C ˘ ⨾ E ⨾ C ˘
    ≈⟨ ⨾-cong_2 char ⟩
      C ˘ ⨾ C ⨾ E ⨾ C ˘
    ≈⟨ ⨾-cong_2 (⨾-assoc ⟨≈˘≈˘⟩ ⨾-cong_1 idempE) ⟩
      C ˘ ⨾ (C ⨾ C ⨾ E) ⨾ C ˘
    ≈⟨ ⨾-cong_2 ⨾-assoc_{3+1} ⟨≈≈⟩ ⨾-assocL ⟩
      (C ˘ ⨾ C) ⨾ C ⨾ E ⨾ C ˘
    ⊑⟨ proj_1 C.unival ⟩
      C ⨾ E ⨾ C ˘
    ≈˘⟨ char ⟩
      E ⨾ C ˘
    □
  monotoneL : E ˘ ⨾ C ⊑ C ⨾ E ˘
  monotoneL = ⊑-begin
      E ˘ ⨾ C
   ≈˘⟨ ˘⨾-˘ ⟩
      (C ˘ ⨾ E) ˘
   ⊑⟨ ˘-monotone monotoneL˘ ⟩
      (E ⨾ C ˘) ˘
   ≈⟨ ⨾˘-˘ ⟩
      C ⨾ E ˘
    □
  monotone : E ⨾ C ⊑ C ⨾ E
  monotone = ⊑-begin
      E ⨾ C
    ⊑⟨ proj_1 C.total ⟨⊑≈⟩ (⨾-assoc ⟨≈≈˘⟩ ⨾-cong_2 ⨾-assoc) ⟩
      C ⨾ (C ˘ ⨾ E) ⨾ C
    ⊑⟨ ⨾-monotone_21 monotoneL˘ ⟨⊑≈⟩ ⨾-cong_2 ⨾-assoc ⟩
      C ⨾ E ⨾ C ˘ ⨾ C
    ⊑⟨ ⨾-monotone_2 (proj_2 C.unival) ⟩
      C ⨾ E
    □

  monotone˘ : C ˘ ⨾ E ˘ ⊑ E ˘ ⨾ C ˘
  monotone˘ = ⊑-begin
      C ˘ ⨾ E ˘
    ≈˘⟨ ⨾-˘ ⟩
      (E ⨾ C) ˘
    ⊑⟨ ˘-monotone monotone ⟩
      (C ⨾ E) ˘
    ≈⟨ ⨾-˘ ⟩
      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 (⨾-˘ ⟨≈˘≈⟩ ˘-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 ˘
     ≈˘⟨ ⨾-˘ ⟩
      (C ⨾ E) ˘
     ≈⟨ ˘-cong char ⟩
      (C ⨾ E ⨾ C ˘) ˘
     ≈⟨ ⨾-˘ ⟨≈≈⟩ (⨾-cong₁ ⨾˘-˘ ⟨≈≈⟩ ⨾-assoc) ⟩
      C ⨾ E ˘ ⨾ C ˘
     □}
  open PreClosureOp COC hiding (monotone; monotoneL˘)
  contraction : E ⊑ C ⨾ E
  contraction = ⊑-˘-swap expansion ⟨⊑≈⟩ ˘⨾˘-˘
  monotone : E ⨾ C ⊑ C ⨾ E
  monotone = ⨾-cong₁ ˘˘ ⟨≈˘⊑⟩ (monotoneL ⟨⊑≈⟩ ⨾-cong₂ ˘˘)
  monotoneL˘ : C ˘ ⨾ E ⊑ E ⨾ C ˘
  monotoneL˘ = ⨾-cong₂ ˘˘ ⟨≈˘⊑⟩ (monotone˘ ⟨⊑≈⟩ ⨾-cong₁ ˘˘)
  open PreClosureOp COC public hiding (char˘; expansion; monotoneL; 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 ˘
    ; monotoneL˘   to monotone˘       -- : C ˘ ⨾ E ˘ ⊑ E ˘ ⨾ C ˘
    ; monotone     to monotoneL       -- : 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.

6.2 Categoric.OCC.Order.Closure

```
open import RATH.Level
open import RATH.Data.Product using (proj₁; proj₂; _,_)
open import Categoric.OCC
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OSGC.SyQ.WithResiduals
module Categoric.OCC.Order.Closure {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 SyQ-ResidualProps        osgc leftResOp rightResOp syqOp
open ResidualOps                   leftResOp rightResOp
open import Categoric.OSGC.Preorder.Extrema osgc leftResOp rightResOp
open import Categoric.OSGC.Preorder.Closure osgc
open import Categoric.OCC.Order occ leftResOp rightResOp syqOp
```

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 ⟨≈˘≈⟩ ⨾-˘) ⟨≈⊑⟩ ⨾-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

```
open import RATH.Level
open import Categoric.OSGC
module Categoric.OSGC.Preorder.Galois {i j k₁ k₂} {Obj : Set i} (osgc : OSGC j k₁ k₂ Obj) where
open OSGC osgc
open import RATH.Data.Product using (proj₁ ; proj₂ ; _,_)
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.Preorder
open import Categoric.OSGC.Preorder.Closure
```

We are now in a position to turn to internal (monotone) Galois Connections. The characterization that (L, U) constitute such a connection is precisely ∀ x, y • L (x) ≼′ y ⇔ x ≼ U (y), i.e. L ⨾ ≼′ ≈ ≼ ⨾ U ˘. 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 (⨾-˘ ⟨≈˘≈⟩ (˘-cong gc ⟨≈≈⟩ ⨾˘-˘))
```


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˘ = ⨾-˘ ⟨≈˘⊑⟩ (˘-⊑-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 ⨾˘-˘)) ⟩
      (R ˘ ⨾ (U ⨾ E₁ ˘) ˘ ⨾ L ˘) ˘
    ≈˘⟨ ˘-cong (⨾-cong₂ ⨾-˘) ⟩
      (R ˘ ⨾ (L ⨾ U ⨾ E₁ ˘) ˘) ˘
    ≈⟨ ˘⨾˘-˘ ⟩
        (L ⨾ U ⨾ E₁ ˘) ⨾ R
    □), (λ {B} {S} → ⊑-begin
        S
    ≈˘⟨ ˘˘ ⟩
        S ˘ ˘
    ⊑⟨ ˘-monotone (proj₁ EU˘L˘-supId) ⟩
      ((E₁ ⨾ U ˘ ⨾ L ˘) ⨾ S ˘) ˘
    ≈⟨       ˘-cong (⨾-cong₁ (⨾-assoc ⟨≈˘≈˘⟩ ⨾-cong₁ ⨾˘-˘)) ⟩
      (((U ⨾ E₁ ˘) ˘ ⨾ L ˘) ⨾ S ˘) ˘
    ≈˘⟨ ˘-cong (⨾-cong₁ ⨾-˘) ⟩
      ((L ⨾ U ⨾ E₁ ˘) ˘ ⨾ S ˘) ˘
    ≈⟨ ˘⨾˘-˘ ⟩
        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 = ⨾-˘ ⟨≈˘⊑⟩ (˘-⊑-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 (˘⨾˘-˘ ⟨≈⊑⟩ (L-monotone ⟨⊑≈˘⟩ ˘⨾˘-˘)) ⟨⊑≈⟩ ˘˘)
L-monotoneL : E₁ ˘ ⨾ L ⊑ L ⨾ E₂ ˘
```



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

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-monotoneL ⟨⊑≈⟩ ⨾-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 ⨾˘-˘ ⟩
          U ⨾ E₁ ⨾ (U ⨾ E₁ ˘) ˘
     ≈˘⟨ ⨾-cong₂₂ (˘-cong ULUE˘≈UE˘) ⟩
          U ⨾ E₁ ⨾ ((U ⨾ L ⨾ U) ⨾ E₁ ˘) ˘
     ≈⟨ ⨾-cong₂₂ ⨾˘-˘ ⟩
          U ⨾ E₁ ⨾ E₁ ⨾ (U ⨾ L ⨾ U) ˘
     ≈⟨ ⨾-cong₂ (⨾-assoc ⟨≈˘≈⟩ (⨾-cong₁ A₁.idempot ⟨≈≈⟩ ⨾-cong₂ ⨾-˘)) ⟩
          U ⨾ E₁ ⨾ ((L ⨾ U) ˘ ⨾ U ˘)
     ≈⟨ ⨾-cong₂₂ (⨾-cong₁ ⨾-˘ ⟨≈≈⟩ ⨾-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 (⨾-˘ ⟨≈≈⟩ ⨾-cong₁ ˘⨾˘-˘) ⟩
          (E₂ ⨾ L ˘ ⨾ U ˘) ˘ ⨾ L ˘ ⨾ U ˘
     ≈˘⟨ ⨾-˘ ⟨≈≈⟩ ⨾-cong₁ ⨾-˘ ⟨≈≈⟩ ⨾-assoc ⟩
          (U ⨾ L ⨾ E₂ ⨾ L ˘ ⨾ U ˘) ˘
     ≈⟨ ˘-cong L-isotone-on-U ⟩
          (U ⨾ E₁ ⨾ U ˘) ˘
     ≈⟨ ⨾-˘ ⟨≈≈⟩ (⨾-cong₁ ⨾˘-˘ ⟨≈≈⟩ ⨾-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 ⨾-˘ ⟩
          L ⨾ E₂ ˘ ⨾ (L ⨾ E₂) ˘
     ≈˘⟨ ⨾-cong₂₂ (˘-cong LULE≈LE) ⟩
          L ⨾ E₂ ˘ ⨾ ((L ⨾ U ⨾ L) ⨾ E₂) ˘
     ≈⟨ ⨾-cong₂₂ ⨾-˘ ⟩
          L ⨾ E₂ ˘ ⨾ E₂ ˘ ⨾ (L ⨾ U ⨾ L) ˘
     ≈⟨ ⨾-cong₂ (⨾-assoc ⟨≈˘≈⟩ (⨾-cong₁ A₂.˘-idempot ⟨≈≈⟩ ⨾-cong₂ ⨾-˘)) ⟩
          L ⨾ E₂ ˘ ⨾ ((U ⨾ L) ˘ ⨾ L ˘)
     ≈⟨ ⨾-cong₂₂ (⨾-cong₁ ⨾-˘ ⟨≈≈⟩ ⨾-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 (˘⨾-˘ ⟨≈≈⟩ ⨾-cong1 ˘⨾˘-˘) ⟩
      (E1 ˘ ⨾ U ˘ ⨾ L ˘) ˘ ⨾ U ˘ ⨾ L ˘
    ≈˘⟨ ⨾-˘ ⟨≈≈⟩ ⨾-cong1 ⨾-˘ ⟨≈≈⟩ ⨾-assoc ⟩
        (L ⨾ U ⨾ E1 ˘ ⨾ U ˘ ⨾ L ˘) ˘
    ≈⟨ ˘-cong U-coisotone-on-L ⟩
        (L ⨾ E2 ˘ ⨾ L ˘) ˘
    ≈⟨ ⨾-˘ ⟨≈≈⟩ ⨾-cong1 ˘⨾˘-˘ ⟨≈≈⟩ ⨾-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 ⨾-˘ ⟨≈≈⟩ ⨾-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
    ; monotoneL    to UL-monotoneL        -- E2 ˘ ⨾ (U ⨾ L) ⊑ (U ⨾ L) ⨾ E2 ˘
    ; monotoneL˘   to UL-monotoneL˘       -- (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 ⨾-˘ ⟨≈≈⟩ ⨾-assoc ⟩
        L ⨾ U ⨾ E1 ⨾ U ˘ ⨾ L ˘
    ≈⟨ U-isotone-on-L ⟩
        L ⨾ E2 ⨾ L ˘
    ≈⟨ ⨾-assoc ⟨≈˘≈⟩ ⨾-cong1 gc ⟩
        (E1 ⨾ U ˘) ⨾ L ˘
    ≈⟨ ⨾-assoc ⟨≈≈˘⟩ ⨾-cong2 ⨾-˘ ⟩
        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) ˘
    ; monotoneL˘   to LU-monotoneL˘       -- (L ⨾ U) ˘ ⨾ E1 ⊑ E1 ⨾ (L ⨾ U) ˘
    ; monotoneL    to LU-monotoneL        -- 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

```
module Categoric.OCC.Preorder.Galois where

open import RATH.Level
open import RATH.Data.Product using (proj1 ; proj2 ; _,_)

open import Categoric.OCC
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OCC.Preorder
```

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 ⨾-˘)
      ⟨≈˘≈⟩ ⨾˘-˘ ⟨≈≈⟩ ⨾-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 ⨾-˘)
      ⟨≈˘≈⟩ (˘⨾˘-˘ ⟨≈≈⟩ ⨾-assoc)))
```

This is nearly three times larger than the indirect proof.

So much for considering identities.

6.5 Categoric.OCC.Order.Galois

```
open import RATH.Level
open import RATH.Data.Product using (proj₁ ; proj₂ ; _,_)
open import Categoric.OCC
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OCC.Preorder
```



```
module Categoric.OCC.Order.Galois {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 import Categoric.OCC.Order occ leftResOp rightResOp syqOp
open import Categoric.OCC.Order.Closure occ leftResOp rightResOp syqOp
open import Categoric.OSGC.Preorder.Galois osgc
```

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 ClosureOp {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.Power.Polarities

```
open import RATH.Level
open import RATH.Data.Product using (proj₁; proj₂)
open import Categoric.OCC
open import Categoric.Category
open import Categoric.MapCat
open import Categoric.LESGraph
open import Categoric.OSGC.PowerOp
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
import Categoric.OSGC.PowerOrder
open import Categoric.ConvSemigroupoid
```

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



```
  (powerOp : PowerOp osgc)
  where
open ResidualOps                               leftResOp rightResOp
open OrdCat-Residual-Props orderedCategory    leftResOp rightResOp
open OSGC-Residuals         osgc               leftResOp rightResOp
open PowerOp osgc powerOp using (ℙ; Λ-⨾∈˘; Λ⨾∈˘; ∈⨾Λ˘; map-Λ)
open Category₁ (MapCat occ)
open import Categoric.OSGC.PowerOrder       osgc leftResOp rightResOp powerOp using (Ω; Ω-trans)
open import Categoric.OSGC.Power.Polarities osgc leftResOp rightResOp powerOp
open import Categoric.OCC.Preorder.Galois
open import Categoric.OCC.Preorder
```

[WK: *For the time being, we have local definitions of Ω-isPreorder and Ω˘-isPreorder here. Probably the overall organisation needs some adaptation to be able to get these from elsewhere.* **]**

```
Ω-isPreorder : {A : Obj} → IsPreorder occ (Ω {A})
Ω-isPreorder = record {refl = \-isReflexive; trans = Ω-trans}
Ω˘-isPreorder : {A : Obj} → IsPreorder occ (Ω {A} ˘)
Ω˘-isPreorder = record {refl = /-isReflexive ⟨⊑≈˘⟩ \-˘; trans = ⨾-˘ ⟨≈˘⊑⟩ ˘-monotone Ω-trans}
```

```
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-monotoneL   to ↑-antitoneL₀      -- Ω ˘ ⨾ R ↑₀ ⊑ R ↑₀ ⨾ Ω ˘ ˘
      ; L-monotoneL˘  to ↑-antitoneL˘      -- R ↑₀ ˘ ⨾ Ω ⊑ Ω ˘ ⨾ R ↑₀ ˘
      ; U-monotoneL   to ↓-antitoneL₀      -- Ω ˘ ˘ ⨾ R ↓₀ ⊑ R ↓₀ ⨾ Ω ˘
      ; U-monotoneL˘  to ↓-antitoneL˘      -- 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 ↓↑-monotoneL˘₀          -- R ↓↑₀ ˘ ⨾ Ω ˘ ˘ ⊑ Ω ˘ ˘ ⨾ R ↓↑₀ ˘
  ; UL-monotone               to ↓↑-monotoneL            -- Ω ˘ ⨾ R ↓↑₀ ⊑ R ↓↑₀ ⨾ Ω ˘
  ; UL-monotoneL              to ↓↑-monotone₀            -- Ω ˘ ˘ ⨾ R ↓↑₀ ⊑ R ↓↑₀ ⨾ Ω ˘ ˘
  ; UL-monotoneL˘             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-monotoneL˘             to ↑↓-monotoneL˘           -- R ↑↓₀ ˘ ⨾ Ω ⊑ Ω ⨾ R ↑↓₀ ˘
  ; LU-monotoneL              to ↑↓-monotoneL            -- 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 ˘˘

↑-antitoneL : Ω ˘ ⨾ R ↑₀ ⊑ R ↑₀ ⨾ Ω
↑-antitoneL = ↑-antitoneL₀ ⟨⊑≈⟩ ⨾-cong₂ ˘˘

↓-monotoneL : Ω ⨾ R ↓₀ ⊑ R ↓₀ ⨾ Ω ˘
↓-monotoneL = ⨾-cong₁ ˘˘ ⟨≈˘⊑⟩ ↓-antitoneL₀

↓-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 ˘˘)

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

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

[MA: *Perhaps also open with the polars switched; cf* ℙ*glb-preserves-↓↑. Then, e.g., you got closure for both ↑↓ and ↓↑ !* **]**

6.7 Categoric.OCC.DirectPower.Polarities

```
open import RATH.Level
open import RATH.Data.Product using (proj₁ ; proj₂)

open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.Category
open import Categoric.MapCat
open import Categoric.LESGraph
open import Categoric.OSGC.PowerOp

open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OSGC.SyQ.WithResiduals
```

```
open import Categoric.OCC.SyQ
import Categoric.OSGC.PowerOrder
open import Categoric.ConvSemigroupoid


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


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 C : Obj} {S : Mor A B} {R : Mor A C} → S ↓₀ ⨾ (∈ \ R) ≈ (S / ∈ ˘) \ R
↓⨾∈\ {A} {B} {C} {S} {R} = ≈-begin
    S ↓₀ ⨾ (∈ \ R)
  ≈⟨ \-inner-⨾ (Mapping.prf (S ↓)) ⟩
    (∈ ⨾ S ↓₀ ˘) \ R
  ≈˘⟨ \-cong₁ ⨾˘-˘ ⟩
    (S ↓₀ ⨾ ∈ ˘) ˘ \ R
  ≈⟨ \-cong₁ (˘-cong ↓⨾∈˘) ⟩
    (∈ \ S ˘) ˘ \ R
  ≈⟨ \-cong₁ \˘-˘ ⟩
    (S / ∈ ˘) \ R
  □


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


↑⨾∈\ : {A B C : Obj} {S : Mor A B} {R : Mor B C} → S ↑₀ ⨾ (∈ \ R) ≈ (∈ \ S) ˘ \ R
↑⨾∈\ {A} {B} {C} {S} {R} = ≈-begin
    S ↑₀ ⨾ (∈ \ R)
  ≈⟨ \-inner-⨾ (Mapping.prf (S ↑)) ⟩
    (∈ ⨾ S ↑₀ ˘) \ R
  ≈˘⟨ \-cong₁ ⨾˘-˘ ⟩
    (S ↑₀ ⨾ ∈ ˘) ˘ \ R
  ≈⟨ \-cong₁ (˘-cong ↑⨾∈˘) ⟩
    (∈ \ S) ˘ \ R
  □


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



```
      (S ˘ / (∈ \ R ˘)) ≬ ∈
  □
```

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

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

```
↑↓≈≬ : {A B : Obj} {S : Mor A B} → S ↑↓₀ ≈ (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 / ∈ ˘)
  □
```

```
↓⨾↑↓⨾∈˘ : {A B C : Obj} {S : Mor A B} {R : Mor A C} → S ↓₀ ⨾ R ↑↓₀ ⨾ ∈ ˘ ≈ (R ˘ / (∈ \ S ˘)) \ R ˘
↓⨾↑↓⨾∈˘ {A} {B} {C} {S} {R} = ≈-begin
      S ↓₀ ⨾ R ↑↓₀ ⨾ ∈ ˘
  ≈⟨ ⨾-cong₂ ↑↓⨾∈˘′ ⟩
      S ↓₀ ⨾ ((R ˘ / ∈ ˘) \ R ˘)
  ≈⟨ \-inner-⨾ (Mapping.prf (S ↓)) ⟨≈≈⟩ \-cong₁ (/-inner-⨾ (Mapping.prf (S ↓))) ⟩
      (R ˘ / (S ↓₀ ⨾ ∈ ˘)) \ R ˘
  ≈⟨ \-cong₁ (/-cong₂ ↓⨾∈˘) ⟩
      (R ˘ / (∈ \ S ˘)) \ R ˘
  □
```

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

```
  ≈⟨ ˘-cong (/-cong₂ ˘/-˘) ⟩
    (R / ((∊ \ S ˘) ˘ \ R)) ˘
  ≈⟨ ˘-cong (/-cong₂ (\-cong₁ \˘-˘)) ⟩
    (R / ((S / ∊ ˘) \ R)) ˘
  □
```

[MA: *The above pairs of proofs can be dualized . . . maybe later? or now for comparisions?* **]**

```
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 ⨾˘-˘ ⟨≈≈˘⟩ \-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 ⨾˘-˘ ⟨≈≈˘⟩ \-inner-⨾ (Mapping.prf (S ↓↑)) ⟩
    S ↓↑₀ ⨾ (∊ \ ∊)
  □
```

```
↓-lub-cocontinuous : {A B : Obj} (R : Mor A B)
                   → {X : Obj} (Q : Mor X (ℙ B)) → ℙ.lub Q ⨾ R ↓₀ ≈ ℙ.glb (Q ⨾ R ↓₀)
↓-lub-cocontinuous R Q = ⨾-cong₁ ℙ.Lub≈lub ⟨≈˘≈⟩ ↓-Lub-cocontinuous R Q ⟨≈≈⟩ ℙ.Glb≈glb
↑-lub-cocontinuous : {A B : Obj} (R : Mor A B)
                   → {X : Obj} (Q : Mor X (ℙ A)) → ℙ.lub Q ⨾ R ↑₀ ≈ ℙ.glb (Q ⨾ R ↑₀)
↑-lub-cocontinuous R Q = ⨾-cong₁ ℙ.Lub≈lub ⟨≈˘≈⟩ ↑-Lub-cocontinuous R Q ⟨≈≈⟩ ℙ.Glb≈glb
```

```
open import Categoric.OCC.Preorder.Galois using (module PreGaloisConnection)
open import Categoric.OCC.Power.Polarities occ leftResOp rightResOp powerOp
  using (Galois-↓-↑-˘; ↑↓-closure)
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})
```

```
ℙglb-preserves-↑↓ : {A B C : Obj} {R : Mor A (ℙ B)} {S : Mor B C}
  → 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 ClosureOp {ℙ B} {Ω} {Ω-isOrder} {S ↑↓} (record {char = ≈-sym ↑↓-closure})
```

6.8 Categoric.OCC.DirectPower.PolaritiesGC

This module reproduces much of the content of Categoric.OCC.DirectPower.Polarities (Sect. 6.7) 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. 6.7), it currently is recommended to base further developments on Categoric.OCC.DirectPower.Polarities instead of on this module.

```
open import RATH.Level
open import RATH.Data.Product using (proj1 ; proj2)
open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.OSGC.PowerOp
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.SyQ
```

```
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-monotoneL 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 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.

6.9 Categoric.OCC.DirectPower.OrderPolarities

```
open import RATH.Level
open import RATH.Data.Product using (proj1 ; proj2)
open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.Category
open import Categoric.MapCat
open import Categoric.LESGraph
open import Categoric.OSGC.PowerOp
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OSGC.SyQ.WithResiduals
open import Categoric.OCC.SyQ
import Categoric.OSGC.PowerOrder
open import Categoric.ConvSemigroupoid
```

Here we collect properties of the polarities ≼↑ and ≼↓ 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 ⨾-˘ ⟩
          (Λ₀ (ubd (lbd (∈ ˘))) ⨾ ∈ ˘) ˘ \ ≼ ˘
        ≈⟨ \-cong₁ (˘-cong Λ⨾∈˘) ⟩
          (ubd (lbd (∈ ˘))) ˘ \ ≼ ˘
        ≈⟨ lbd-ubd-lbd ⟩
          lbd (∈ ˘)
        □


  downset : Mapping A (ℙ A)
  downset = Λ (≼ ˘)
  downset₀ : Mor A (ℙ A)
  downset₀ = Mapping.mor downset   -- ≈ Λ₀ (≼ ˘)
  downset-isInjectiveI : isInjectiveI downset₀
  downset-isInjectiveI = ⨾-cong₂ ⊓-˘ ⟨≈⊑⟩ ⊓-cancel-middle ⟨⊑≈⟩ ⊓-cong ˘˘ ˘˘ ⟨⊑≈⟩ ≼⊓≼
  downset-isInjective : isInjective downset₀
  downset-isInjective = isInjective-from-I downset-isInjectiveI
  upset : Mor A (ℙ A)
  upset = Λ₀ ≼
  down-up-nat : downset₀ ⨾ ∈ ˘ ≈ ∈ ⨾ 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∈˘-⊑ : downset₀ ⨾ lub (∈ ˘) ⊑ Id
downset-⨾-lub∈˘-⊑ = ⊑-begin
      downset₀ ⨾ 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 ⨾-˘ ⟩
      (≼ ↓↑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 ⨾-˘ ⟩
      (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 : downset₀ ⨾ Ω ˘ ≈ ≼ ˘ ⨾ lub (∈ ˘) ˘
  downset-char = ≈-begin
        Λ₀ (≼ ˘) ⨾ Ω ˘
     ≈⟨ Λ⨾Ω˘ ⟩
        ≼ ˘ / ∈ ˘
     ≈˘⟨ ˘-cong (\-cong₁ ˘˘) ⟨≈≈⟩ \-˘ ⟩
        ubd (∈ ˘) ˘
     ≈˘⟨ ˘-cong (total-lub-⨾-order lub-total) ⟩
        (lub (∈ ˘) ⨾ ≼) ˘
     ≈⟨ ⨾-˘ ⟩
        ≼ ˘ ⨾ lub (∈ ˘) ˘
     □

  downset-⨾-Ω : downset₀ ⨾ Ω ≈ ≼ \ ∈
  downset-⨾-Ω = ≈-begin
        Λ₀ (≼ ˘) ⨾ (∈ \ ∈)
     ≈⟨ \-inner-⨾ Λ-isMapping ⟩
        (∈ ⨾ Λ₀ (≼ ˘) ˘) \ ∈
     ≈⟨ \-cong₁ (⨾˘-˘ ⟨≈˘≈⟩ ≈-˘-swap Λ⨾∈˘) ⟩
        ≼ \ ∈
     □

  downset-⨾-ubd∈˘ : downset₀ ⨾ ubd (∈ ˘) ≈ ≼
  downset-⨾-ubd∈˘ = ≈-begin
        downset₀ ⨾ ubd (∈ ˘)
     ≈⟨ ⨾-cong₂ (\-cong₁ ˘˘) ⟩
        Λ₀ (≼ ˘) ⨾ (∈ \ ≼)
     ≈⟨ \-inner-⨾ Λ-isMapping ⟩
        (∈ ⨾ Λ₀ (≼ ˘) ˘) \ ≼
```

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

```
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∈˘ : downset₀ ⨾ lub (∈ ˘) ≈ Id
downset-⨾-lub∈˘ = total⊑unival-≈ (⨾-isTotal ℙ.∈-comprehensive lub-total)
                                  idUnivalent
                                  downset-⨾-lub∈˘-⊑
downset-semi-inverse : {Z : Obj} {R : Mor A Z} → downset₀ ⨾ lub (∈ ˘) ⨾ R ≈ R
downset-semi-inverse = ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ downset-⨾-lub∈˘ ⟨≈≈⟩ leftId
```

A useful consequence of this is that lub (∈ ˘) is surjective:

```
lub∈˘-˘-⨾-lub∈˘ : lub (∈ ˘) ˘ ⨾ lub (∈ ˘) ≈ Id
lub∈˘-˘-⨾-lub∈˘ = ≈-begin
    lub (∈ ˘) ˘ ⨾ lub (∈ ˘)
  ≈⟨ leftId ⟨≈˘≈⟩ (⨾-cong₁ downset-⨾-lub∈˘ ⟨≈˘≈⟩ ⨾-assoc) ⟩
    downset₀ ⨾ lub (∈ ˘) ⨾ lub (∈ ˘) ˘ ⨾ lub (∈ ˘)
  ≈⟨ ⨾-cong₂ (mappingBiDifunctional (lub-Mapping (∈ ˘))) ⟩
    downset₀ ⨾ lub (∈ ˘)
  ≈⟨ downset-⨾-lub∈˘ ⟩
    Id
  □
```

```
glb⨾downset : {I : Obj} {R : Mor I A} → glb R ⨾ downset₀ ≈ Λ₀ (lbd R)
glb⨾downset {I} {R} = ≈-begin
    glb R ⨾ downset₀
  ≈⟨⟩
    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 ⨾˘-˘) ⟩
    Λ₀ (∈ \ (R ⨾ ≼ ˘))
  ≈⟨   Λ-cong (\-flip-M R-isMapping ⟨≈˘≈˘⟩ \-cong₁ ˘⨾-˘) ⟩
    Λ₀ (lbd (∈ ˘ ⨾ R))
  □
```

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

```
≼↑↓-≈-lub∋⨾downset : ≼ ↑↓₀ ≈ lub (∈ ˘) ⨾ downset₀
≼↑↓-≈-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 (∈ ˘) ⨾ downset₀
  □
```

At least downset-closed can probably also be obtained via the Galois connection based on downset-char.

```
downset-closed₀ : downset₀ ˘ ⨾ downset₀ ≈ ≼ ↑↓₀ ˘ ⨾ ≼ ↑↓₀
downset-closed₀ = ≈-begin
    downset₀ ˘ ⨾ downset₀
  ≈⟨ ⨾-cong₂ (leftId ⟨≈˘≈˘⟩ ⨾-cong₁ lub∈˘-˘-⨾-lub∈˘) ⟨≈≈⟩ ⨾-₁₂₁assoc₂₂ ⟩
    (downset₀ ˘ ⨾ lub (∈ ˘) ˘) ⨾ (lub (∈ ˘) ⨾ downset₀)
  ≈˘⟨ ⨾-cong (˘-cong ≼↑↓-≈-lub∋⨾downset ⟨≈≈⟩ ⨾-˘) ≼↑↓-≈-lub∋⨾downset ⟩
    ≼ ↑↓₀ ˘ ⨾ ≼ ↑↓₀
  □
```

```
downset-closed : downset₀ ⨾ ≼ ↑↓₀ ˘ ⨾ ≼ ↑↓₀ ≈ downset₀
downset-closed = ≈-begin
    downset₀ ⨾ ≼ ↑↓₀ ˘ ⨾ ≼ ↑↓₀
  ≈˘⟨ ⨾-cong₂ downset-closed₀ ⟩
    downset₀ ⨾ downset₀ ˘ ⨾ downset₀
  ≈⟨ mappingBiDifunctional downset ⟩
    downset₀
  □
```

Chapter 7

Abstract Contexts

This chapter contains the details of the formalisation of abstract contexts described in (Kahl, 2014a). Contexts themselves, with their polarities-based compatibility conditions, and context homomorphisms require for their definition, in Sect. 7.1, only an OSGC with a power operator. Defining a "singleton set constructor" requires in addition identity morphisms; we set this in an OCC with a power operator in Sect. 7.2. This singleton operator is required for being able to define composition of context homomorphisms and prove that these form a category, which is done in Sect. 7.3.

7.1 Data.AContext.InOSGC

```
open import RATH.Level
open import RATH.Data.Product
open import Categoric.OSGC
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.PowerOp
open import Categoric.Semigroupoid
open import Categoric.MapSG
open import Categoric.LESGraph
open import RATH.Data.Product using (proj₁; proj₂)
open import Function using (_∘_)

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 (⨾˘-˘ ⟨≈˘≈⟩ ˘-cong ↓↑⨾∈˘) ⟩
    ((Y / ∈ ˘) \ Y) ˘ \ R ˘
  ≈˘⟨ /-˘ ⟩
    (R / ((Y / ∈ ˘) \ Y)) ˘
  ≈⟨ ˘-cong (T/∘\S∘S/ (↓↑⨾∈˘ ⟨≈˘⊑⟩ Y▫R ⟨⊑≈⟩ ↓↑⨾∈˘)) ⟩
    (R / ∈ ˘) ˘
  ≈⟨ /˘-˘ ⟩
    ∈ \ R ˘
  □)
```

A "context" is simply a "typed relation", that is, a pair of "sets" together with a relation between them:

```
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 : mor ↓ ⨾₁ X.inc ↑↓ ≈₁ mor ↓
          trgCompat : Y.inc ↓↑ ⨾₁ mor ↓ ≈₁ mor ↓
  srcCompat′ : mor ↓₀ ⨾ X.inc ↑↓₀ ˘ ⨾ X.inc ↑↓₀ ≈ mor ↓₀
  srcCompat′ = ⊑-antisym (proj₂ (mappingUnivalent (X.inc ↑↓))) (↑↓-ranClosed-← srcCompat)
```

We add a number of useful consequences of the compatibility properties:

```
  srcCompat-⨾∈˘ : X.inc ↑↓₀ ⨾ ∈ ˘ ⊑ mor ↑↓₀ ⨾ ∈ ˘
  srcCompat-⨾∈˘ = SrcCompat-⇒ X.inc mor srcCompat

  trgCompat-⨾∈˘ : Y.inc ↓↑₀ ⨾ ∈ ˘ ⊑ mor ↓↑₀ ⨾ ∈ ˘
  trgCompat-⨾∈˘ = TrgCompat-⇒ mor Y.inc trgCompat

  srcCompat-/∈\ : X.inc / (∈ \ X.inc) ⊑ mor / (∈ \ mor)
  srcCompat-/∈\ = ˘˘ ⟨≈˘⊑⟩ ˘-monotone (↑↓⨾∈˘ ⟨≈˘⊑⟩ srcCompat-⨾∈˘ ⟨⊑≈⟩ ↑↓⨾∈˘) ⟨⊑≈⟩ ˘˘

  trgCompat-/∈˘\ : (Y.inc / ∈ ˘) \ Y.inc ⊑ (mor / ∈ ˘) \ mor
  trgCompat-/∈˘\ = ↓↑⨾∈˘ ⟨≈˘⊑⟩ trgCompat-⨾∈˘ ⟨⊑≈⟩ ↓↑⨾∈˘

  srcCompat-/\ : {Z : Obj} {R : Mor X.ent Z} → X.inc / (R \ X.inc) ⊑ mor / (R \ mor)
  srcCompat-/\ {Z} {R} = ⊑-begin
      X.inc / (R \ X.inc)
    ≈˘⟨ /-cong₂ (\-cong₁ (⨾˘-˘ ⟨≈˘≈⟩ ≈-˘-swap Λ⨾∈˘)) ⟩
      X.inc / ((∈ ⨾ Λ₀ (R ˘) ˘) \ X.inc)
    ≈⟨ /-cong₂ (\-inner-⨾ Λ-mapping) ⟨≈˘≈˘⟩ /-inner-⨾ Λ-mapping ⟩
      (X.inc / (∈ \ X.inc)) ⨾ Λ₀ (R ˘) ˘
    ⊑⟨ ⨾-monotone₁ srcCompat-/∈\ ⟩
      (mor / (∈ \ mor)) ⨾ Λ₀ (R ˘) ˘
    ≈⟨ /-inner-⨾ Λ-mapping ⟨≈≈⟩ /-cong₂ (\-inner-⨾ Λ-mapping) ⟩
      mor / ((∈ ⨾ Λ₀ (R ˘) ˘) \ mor)
    ≈⟨ /-cong₂ (\-cong₁ (⨾˘-˘ ⟨≈˘≈⟩ ≈-˘-swap Λ⨾∈˘)) ⟩
      mor / (R \ mor)
    □

  trgCompat-/\ : {Z : Obj} {R : Mor Z Y.att} → (Y.inc / R) \ Y.inc ⊑ (mor / R) \ mor
  trgCompat-/\ {Z} {R} = ⊑-begin
      (Y.inc / R) \ Y.inc
    ≈˘⟨ \-cong₁ (/-cong₂ Λ⨾∈˘) ⟩
      (Y.inc / (Λ₀ R ⨾ ∈ ˘)) \ Y.inc
    ≈⟨ \-cong₁ (/-inner-⨾ Λ-mapping) ⟨≈˘≈˘⟩ \-inner-⨾ Λ-mapping ⟩
      Λ₀ R ⨾ ((Y.inc / ∈ ˘) \ Y.inc)
    ⊑⟨ ⨾-monotone₂ trgCompat-/∈˘\ ⟩
      Λ₀ R ⨾ ((mor / ∈ ˘) \ mor)
    ≈⟨ \-inner-⨾ Λ-mapping ⟨≈≈⟩ \-cong₁ (/-inner-⨾ Λ-mapping) ⟩
      (mor / (Λ₀ R ⨾ ∈ ˘)) \ mor
    ≈⟨ \-cong₁ (/-cong₂ Λ⨾∈˘) ⟩
      (mor / R) \ 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

```
open import RATH.Level
open import RATH.Data.Product

open import Categoric.OCC
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.PowerOp
open import Categoric.Category
open import Categoric.MapCat
open import Categoric.LESGraph

open import RATH.Data.Product using (proj₁; proj₂)
open import Function using (_∘_)
```

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 $a \in \mathsf{f}\{b\}$:

```
[_] : {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:

```
ΛId-⨾-↑ : {A B : Obj} {R : Mor A B} → Λ₀ Id ⨾ R ↑₀ ≈ Λ₀ R
ΛId-⨾-↑ {A} {B} {R} = ≈-begin
    Λ₀ Id ⨾ Λ₀ (∈ \ R)
  ≈⟨ map-Λ {f = Λ Id} ⟩
    Λ₀ (Λ₀ Id ⨾ (∈ \ R))
  ≈⟨ Λ-cong (\-inner-⨾ (Mapping.prf (Λ Id))) ⟩
    Λ₀ ((∈ ⨾ Λ₀ Id ˘) \ R)
  ≈⟨ Λ-cong (\-cong₁ (⨾˘-˘ ⟨≈˘≈⟩ ˘-cong Λ⨾∈˘ ⟨≈≈⟩ Id˘) ⟨≈≈⟩ Id-\) ⟩
    Λ₀ R
  □

ΛId-⨾-↓ : {A B : Obj} {R : Mor A B} → Λ₀ Id ⨾ R ↓₀ ≈ Λ₀ (R ˘)
ΛId-⨾-↓ = ΛId-⨾-↑

[↓] : {A B : Obj} (R : Mor A B) → [R↓] ≈ R
[↓] R = ≈-begin
    (Λ₀ Id ⨾ R ↓₀ ⨾ ∈ ˘) ˘
  ≈⟨ ˘-cong (⨾-assocL ⟨≈≈⟩ ⨾-cong₁ ΛId-⨾-↓) ⟩
    (Λ₀ (R ˘) ⨾ ∈ ˘) ˘
  ≈⟨ ˘-cong Λ⨾∈˘ ⟨≈≈⟩ ˘˘ ⟩
    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 ˘⨾-˘ ⟨≈≈⟩ \-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

```
open import RATH.Level
open import Categoric.OCC
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.PowerOp
open import Categoric.Category
open import Categoric.MapCat
open import Categoric.LESGraph
open import RATH.Data.Product using (proj₁ ; proj₂)
```

```
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↓ G↓⍮Y↑⍮F↓-Lub-cocontinuous

  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
```

$$X_1 \xrightarrow{F} X_2 \xrightarrow{G} X_3 \xrightarrow{H} X_4$$


```
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}
    }
  }
```

Chapter 8

Abstract Complete Semilattices in OCCs

In Sect. 8.1 we define complete lower semilattices and their meet-preserving homomorphisms, and show that these form the category ACSL-Category. As first step toward formalising the duality outlined by Moshier (2013) between this category and the category of abstract contexts from Sect. 7.3, we then define the two functors between the latter and the opposite of ACSL-Category in sections 8.2 and 8.3.

8.1 Categoric.OCC.CSL

```
open import RATH.Level
open import RATH.Data.Product using (_,_; proj₁; proj₂)
open import Categoric.OCC
open import Categoric.MapCat
open import Categoric.Category
open import Categoric.LESGraph
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OSGC.SyQ.WithResiduals
open import Categoric.OCC.SyQ
```

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
  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)
```

We derive the "L-similation" shape of the monotonicity condition:

```
  monotone-L : A.≼ ˘ ⨾ map0 ⊑ map0 ⨾ B.≼ ˘
  monotone-L = mappingHom1-to-L (Mapping.prf map) monotone
```

From that, we obtain another useful lemma:

```
  ≼⨾map˘/≼ : (B.≼ ⨾ map0 ˘) / A.≼ ≈ B.≼ ⨾ map0 ˘
  ≼⨾map˘/≼ = ⊑-antisym
    (⊑-begin
       (B.≼ ⨾ map0 ˘) / A.≼
    ⊑⟨ /-antitone A.≼-refl ⟩
       (B.≼ ⨾ map0 ˘) / Id
    ≈⟨ /-Id ⟩
       B.≼ ⨾ map0 ˘
    □)
    (/-universal (⊑-begin
       (B.≼ ⨾ map0 ˘) ⨾ A.≼
    ≈⟨ ⨾-assoc ⟨≈≈˘⟩ ⨾-cong2 ˘⨾-˘ ⟩
       B.≼ ⨾ (A.≼ ˘ ⨾ map0) ˘
    ⊑⟨ ⨾-monotone2 (˘-monotone monotone-L ⟨⊑≈⟩ ⨾˘-˘) ⟩
```



```
       B.≼ ⨾ B.≼ ⨾ map0 ˘
    ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone1 B.≼-trans ⟩
       B.≼ ⨾ 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 that corresponds to the fact that 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.≼
      □
    }
```

```
infix 4 _≋_
_≋_ : {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
  }

infixr 9 _⨾⨾_
_⨾⨾_ : {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}
ACSL-OpCategory : Category (i ⊍ j ⊍ k₁ ⊍ k₂) k₁ ACSL
ACSL-OpCategory = oppositeCategory ACSL-Category
```

8.2 Categoric.OCC.CSL.ToAContext

We define the contravariant functor from the complete lower semilattices of Categoric.OCC.CSL (Sect. 8.1) to the "abstract contexts" of Data.AContext.Category (Sect. 7.3) as suggested by Moshier (2013) for showing the duality between these two categories.

```
open import RATH.Level
open import RATH.Data.Product using (proj₁; proj₂)
open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.Category using (module Category₁; oppositeCategory)
open import Categoric.Functor
open import Categoric.MapCat using (MapCat)
open import Categoric.OSGC.PowerOp using (module PowerOp)
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OSGC.SyQ.WithResiduals
open import Categoric.OCC.SyQ

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)
  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.OCC.CSL                 occ leftResOp rightResOp syqOp
  renaming (_≋_ to _≋₁_; _⨾⨾_ to _⨾⨾₁_)
open import Categoric.OCC.DirectPower.OrderPolarities occ leftResOp rightResOp syqOp directPower
open import Categoric.OCC.DirectPower.Polarities occ leftResOp rightResOp syqOp directPower
open import Categoric.OSGC.Power.Polarities osgc leftResOp rightResOp powerOp
```

```
open import Data.AContext.InOSGC          osgc leftResOp rightResOp powerOp
  renaming (_≋_ to _≋₂_)
```

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 = mor; srcCompat = srcCompat; trgCompat = trgCompat}
  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)
    mor = ≼B ⨾ map₀ ˘
    srcCompat = ≈-begin
        mor ↓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 ↓⨾∊\) ⟩
        (≼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 ˘) / ∊ ˘) ≬ ∊
      ≈˘⟨ ↓≈≬ ⟩
        mor ↓₀
      □
```

The following lemma₁ is used to show lemma below, which is used in the proof of trgCompat.

Due to continuity, the lower bounds of the map-image of some set are exactly the members of the downset₀ of the map-image of the glb of that set:

```
    lemma₁ : Λ₀ (B.lbd (∊ ˘ ⨾ map₀)) ≈ A.glb (∊ ˘) ⨾ map₀ ⨾ B.downset₀
    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.downset₀
      □


    lemma : (≼B ⨾ map₀ ˘) ↓₀ ≈ A.glb (∊ ˘) ⨾ map₀ ⨾ B.downset₀
    lemma = ≈-begin
        (≼B ⨾ map₀ ˘) ↓₀
      ≈⟨ B.lemma₀ (Mapping.prf map) ⟩
        Λ₀ (B.lbd (∊ ˘ ⨾ map₀))
      ≈⟨ lemma₁ ⟩
        A.glb (∊ ˘) ⨾ map₀ ⨾ B.downset₀
      □


    trgCompat = ≈-begin
        ≼A ↓↑₀ ⨾ (≼B ⨾ map₀ ˘) ↓₀
      ≈⟨ ⨾-cong₂ lemma ⟩
        ≼A ↓↑₀ ⨾ A.meet ⨾ map₀ ⨾ B.downset₀
      ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ A.≼↓↑-⨾-glb∊˘ ⟩
        A.meet ⨾ map₀ ⨾ B.downset₀
      ≈˘⟨ lemma ⟩
        (≼B ⨾ map₀ ˘) ↓₀
      □
```

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 : mor ↓₀ ⨾ ≼B ↑↓₀ ≈ mor ↓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
      □
```

Proving that these pieces extend to a functor is relatively straight-forward:

```
open import Data.AContext.InOCC     occ leftResOp rightResOp powerOp
open import Data.AContext.Category occ leftResOp rightResOp powerOp
  renaming (_⨾⨾_ to _⨾⨾₂_)
CSL→Ctx : Functor ACSL-OpCategory ACH-Category
CSL→Ctx = record
  {obj      = fromACSL
  ; mor      = fromACSLHom
```

```
; mor-cong = λ {A} {B} {F} {G}       → mor-cong {A} {B} {F} {G}
; mor-⨾     = λ {A} {B} {C} {F} {G} → mor-⨾ {A} {B} {C} {F} {G}
; mor-Id    = λ {A}                  → mor-Id {A}
}
where
  mor-cong : {A B : ACSL} {F G : ACSLHom B A} → F ≋₁ G → fromACSLHom F ≋₂ fromACSLHom G
  mor-cong {A} {B} {F} {G} F≈G = ⨾-cong₂ (˘-cong F≈G)
  mor-Id : {A : ACSL} → fromACSLHom (ACSL-Id {A}) ≋₂ AContext-Id
  mor-Id {A} = ⨾-cong₂ Id˘ ⟨≈≈⟩ rightId
  mor-⨾ : {A B C : ACSL} {F : ACSLHom B A} {G : ACSLHom C B}
       → fromACSLHom (G ⨾⨾₁ F) ≋₂ fromACSLHom F ⨾⨾₂ fromACSLHom G
  mor-⨾ {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
    in ≈-sym (≈-begin
      [ (B.≼ ⨾ G.map₀ ˘) ↓ ⨾₁ B.≼ ↑ ⨾₁ (A.≼ ⨾ F.map₀ ˘) ↓ ]
    ≈⟨⟩
      (Λ₀ Id ⨾ ((B.≼ ⨾ G.map₀ ˘) ↓₀ ⨾ B.≼ ↑₀ ⨾ (A.≼ ⨾ F.map₀ ˘) ↓₀) ⨾ ∈ ˘) ˘
    ≈⟨ ˘-cong (⨾-cong₂ (⨾-assoc₃₊₁ ⟨≈≈⟩ ⨾-cong₂₂ ↓⨾∈˘)) ⟩
      (Λ₀ Id ⨾ (B.≼ ⨾ G.map₀ ˘) ↓₀ ⨾ B.≼ ↑₀ ⨾ (∈ \ (A.≼ ⨾ F.map₀ ˘) ˘)) ˘
    ≈⟨ ˘-cong (⨾-assocL₃₊₁ ⟨≈≈⟩ \˘-inner-⨾ (Mapping.prf (Λ Id ⨾₁ (B.≼ ⨾ G.map₀ ˘) ↓ ⨾₁ B.≼ ↑)))) ⟩
      (((A.≼ ⨾ F.map₀ ˘) / ((Λ₀ Id ⨾ (B.≼ ⨾ G.map₀ ˘) ↓₀ ⨾ B.≼ ↑₀) ⨾ ∈ ˘)) ˘) ˘
    ≈⟨ ˘˘ ⟨≈≈⟩ /-cong₂ (⨾-assoc₃₊₁ ⟨≈≈⟩ ⨾-cong₂₂ ↑⨾∈˘) ⟩
      (A.≼ ⨾ F.map₀ ˘) / (Λ₀ Id ⨾ (B.≼ ⨾ G.map₀ ˘) ↓₀ ⨾ (∈ \ B.≼))
    ≈⟨ /-cong₂ (⨾-cong₂ ↓⨾∈\) ⟩
      (A.≼ ⨾ F.map₀ ˘) / (Λ₀ Id ⨾ (((B.≼ ⨾ G.map₀ ˘) / ∈ ˘) \ B.≼))
    ≈⟨ /-cong₂ (\-inner-⨾ Λ-isMapping) ⟩
      (A.≼ ⨾ F.map₀ ˘) / ((((B.≼ ⨾ G.map₀ ˘) / ∈ ˘) ⨾ Λ₀ Id ˘) \ B.≼)
    ≈⟨ /-cong₂ (\-cong₁ (/-inner-⨾ Λ-isMapping)) ⟩
      (A.≼ ⨾ F.map₀ ˘) / (((B.≼ ⨾ G.map₀ ˘) / (Λ₀ Id ⨾ ∈ ˘)) \ B.≼)
    ≈⟨ /-cong₂ (\-cong₁ (/-cong₂ Λ⨾∈˘ ⟨≈≈⟩ /-Id)) ⟩
      (A.≼ ⨾ F.map₀ ˘) / ((B.≼ ⨾ G.map₀ ˘) \ B.≼)
    ≈˘⟨ /-cong₂ (\-inner-⨾ (Mapping.prf G.map)) ⟩
      (A.≼ ⨾ F.map₀ ˘) / (G.map₀ ⨾ (B.≼ \ B.≼))
    ≈˘⟨ /-inner-⨾ (Mapping.prf G.map) ⟩
      ((A.≼ ⨾ F.map₀ ˘) / (B.≼ \ B.≼)) ⨾ G.map₀ ˘
    ≈⟨ ⨾-cong₁ (/-cong₂ B.order-\) ⟩
      ((A.≼ ⨾ F.map₀ ˘) / B.≼) ⨾ G.map₀ ˘
    ≈⟨ ⨾-cong₁ F.≼⨾map˘/≼ ⟨≈≈⟩ ⨾-assoc ⟨≈≈˘⟩ ⨾-cong₂ ⨾-˘ ⟩
      A.≼ ⨾ (G.map₀ ⨾ F.map₀) ˘
    □)
```

8.3 Categoric.OCC.CSL.FromAContext

```
open import RATH.Level
open import RATH.Data.Product using (Σ:•; proj₁; proj₂; _,_)

open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.OSGC.PowerOp

open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OSGC.SyQ.WithResiduals
open import Categoric.OCC.SyQ
```



```
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)
private
  module ℙ = DirectPower directPower
open ℙ using (powerOp; ℙ; ∈; ∈⨾-∈\; /∈\-⊑-→-/-⊑; /∈˘\-⊑-→-\-⊑
  ; Ω; Ω-isOrder; Ω-trans; Ω-refl; Ω⨾Ω; Ω˘-trans; Ω\Ω≈Ω)
open PowerOp                                   osgc powerOp
  using (Λ⨾∈˘)
open import Categoric.OSGC.PowerOrder          osgc leftResOp rightResOp powerOp
  using (Lub₀; Glb₀; Lub-cong; Glb-cong)
open import Categoric.OSGC.Power.Polarities osgc leftResOp rightResOp powerOp
open import Data.AContext.InOSGC               osgc leftResOp rightResOp powerOp
  renaming (_≋_ to _≋₂_)
open import Categoric.OCC.CSL                  occ  leftResOp rightResOp syqOp
  renaming (_≋_ to _≋₁_; _⨾⨾_ to _⨾⨾₁_)
open import Categoric.OCC.DirectPower.Polarities occ leftResOp rightResOp syqOp directPower
  using (ℙglb-preserves-↑↓; ↓-lub-cocontinuous; Ω-⨾-↑↓˘)

toACSL : AContext → ACSL
toACSL A = record
    {Carrier = ↑↓-image
    ; ≼ = ≼      -- = Q˘ ⨾ Ω ⨾ Q – leave, subset ordering, then come back
    ; ≼-isOrder = ≼-isOrder
    ; glb-total = λ {I} R → glb-total {I} {R}
    }
  module ToACSL where
    open AContext A public
    inc↑↓ : Mapping (ℙ ent) (ℙ ent)
    inc↑↓ = inc ↑↓
    inc↑↓₀ : Mor (ℙ ent) (ℙ ent)
    inc↑↓₀ = Mapping.mor inc↑↓
    inc↑↓-⨾-∈\inc : inc↑↓₀ ⨾ (∈ \ inc) ≈ ∈ \ inc
    inc↑↓-⨾-∈\inc = ≈-begin
        inc↑↓₀ ⨾ (∈ \ inc)
      ≈⟨ \-inner-⨾ (Mapping.prf inc↑↓) ⟩
        (∈ ⨾ inc↑↓₀ ˘) \ inc
      ≈⟨ \-cong₁ (⨾˘-˘ ⟨≈˘≈⟩ ˘-cong ↑↓⨾∈˘ ⟨≈≈⟩ ˘˘) ⟨≈≈⟩ \S∘S/∘\S ⟩
        ∈ \ inc
      □
```

```
inc↑↓-⨾-Ω-⨾-inc↑↓˘ : inc↑↓₀ ⨾ Ω ⨾ inc↑↓₀ ˘ ≈ Ω ⨾ inc↑↓₀ ˘
inc↑↓-⨾-Ω-⨾-inc↑↓˘ = ≈-begin
    inc↑↓₀ ⨾ Ω ⨾ inc↑↓₀ ˘
  ≈⟨ ⨾-cong₂ Ω-⨾-↑↓˘ ⟩
    inc↑↓₀ ⨾ ((∈ \ inc) / (∈ \ inc))
  ≈⟨ /-outer-⨾-≈ (Mapping.prf inc↑↓) ⟩
    (inc↑↓₀ ⨾ (∈ \ inc)) / (∈ \ inc)
  ≈⟨ /-cong₁ inc↑↓-⨾-∈\inc ⟩
    (∈ \ inc) / (∈ \ inc)
  ≈˘⟨ Ω-⨾-↑↓˘ ⟩
    Ω ⨾ inc↑↓₀ ˘
  □
Ξ : Mor (ℙ ent) (ℙ ent)
Ξ = inc↑↓₀ ˘ ⨾ inc↑↓₀
Ξ-isSubIdentity : isSubidentity Ξ
Ξ-isSubIdentity = mappingUnivalent inc↑↓
↓⨾Ξ : inc ↓₀ ⨾ Ξ ≈ inc ↓₀
↓⨾Ξ = ⊑-antisym (proj₂ Ξ-isSubIdentity) (↑↓-ranClosed-← ↓⨾↑↓)
Ξ-isSymIdempot : IsSymIdempot Ξ
Ξ-isSymIdempot = record
  {symmetric = ˘⨾-˘
  ; idempotent = ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ (mappingBiDifunctional inc↑↓)
  }
split : SymSplitting Ξ
split = splitSymIdempot {ℙ ent} {Ξ} Ξ-isSymIdempot
module Q = SymSplitting split
open Q public renaming (obj to ↑↓-image; mor to Q; proof to isSymSplitting)
Q˘ : Mor ↑↓-image (ℙ ent)
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 ent) ↑↓-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)
Q-isSurjective : isSurjective Q
Q-isSurjective = isSurjectiveFromTotal (proj₂ Q˘-isMapping)
Q˘⨾Q≈Id : Q˘ ⨾ Q ≈ Id
Q˘⨾Q≈Id = isIdentity-≈Id splitId
Q˘⨾Ξ≈Q˘ : Q˘ ⨾ Ξ ≈ Q˘
Q˘⨾Ξ≈Q˘ = ≈-begin
    Q˘ ⨾ Ξ
  ≈˘⟨ ⨾-cong₂ factors ⟩
    Q˘ ⨾ Q ⨾ Q˘
  ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ Q˘⨾Q≈Id ⟨≈≈⟩ leftId ⟩
    Q˘
  □
inc↑↓˘-⨾-Q : inc↑↓₀ ˘ ⨾ Q ≈ Q
inc↑↓˘-⨾-Q = ≈-begin
    inc↑↓₀ ˘ ⨾ Q
  ≈⟨ ⨾-cong₂ (leftClosed ⟨≈˘≈⟩ ⨾-assoc) ⟨≈≈⟩ ⨾-assocL ⟩
    (inc↑↓₀ ˘ ⨾ inc↑↓₀ ˘) ⨾ inc↑↓₀ ⨾ Q
```



```
  ≈⟨ ⨾-cong₁ (⨾-˘ ⟨≈˘≈⟩ ˘-cong ↑↓-idempotent) ⟩
    inc↑↓₀ ˘ ⨾ inc↑↓₀ ⨾ Q
  ≈⟨ ⨾-assocL ⟨≈≈⟩ leftClosed ⟩
    Q
  □
Q˘-⨾-inc↑↓ : Q˘ ⨾ inc↑↓₀ ≈ Q ˘
Q˘-⨾-inc↑↓ = ˘⨾-˘ ⟨≈˘≈⟩ ˘-cong inc↑↓˘-⨾-Q
inc/-∈⨾Q\inc : inc / ((∈ ⨾ Q) \ inc) ≈ ∈ ⨾ Q
inc/-∈⨾Q\inc = ≈-begin
    inc / ((∈ ⨾ Q) \ inc)
  ≈˘⟨ /-cong₂ (\-cong₁ (⨾-cong₂ ˘˘)) ⟩
    inc / ((∈ ⨾ Q˘ ˘) \ inc)
  ≈˘⟨ /-cong₂ (\-inner-⨾ Q˘-isMapping) ⟩
    inc / (Q˘ ⨾ (∈ \ inc))
  ≈˘⟨ /-inner-⨾ Q˘-isMapping ⟩
    (inc / (∈ \ inc)) ⨾ Q˘ ˘
  ≈⟨ ⨾-cong (≈-sym ∈⨾↑↓˘) ˘˘ ⟨≈≈⟩ ⨾-assoc ⟩
    ∈ ⨾ inc↑↓₀ ˘ ⨾ Q
  ≈⟨ ⨾-cong₂ inc↑↓˘-⨾-Q ⟩
    ∈ ⨾ Q
  □
inc↑↓-⨾-Ω-⨾-Q : inc↑↓₀ ⨾ Ω ⨾ Q ≈ Ω ⨾ Q
inc↑↓-⨾-Ω-⨾-Q = ≈-begin
    inc↑↓₀ ⨾ Ω ⨾ Q
  ≈⟨ ⨾-cong₂₂ (leftClosed ⟨≈˘≈⟩ ⨾-assoc) ⟩
    inc↑↓₀ ⨾ Ω ⨾ inc↑↓₀ ˘ ⨾ inc↑↓₀ ⨾ Q
  ≈⟨  ⨾-assocL₃₊₁ ⟨≈≈⟩ ⨾-cong₁ inc↑↓-⨾-Ω-⨾-inc↑↓˘ ⟩
    (Ω ⨾ inc↑↓₀ ˘) ⨾ inc↑↓₀ ⨾ Q
  ≈⟨ ⨾-₂₂assoc₁₂₁ ⟨≈≈⟩ ⨾-cong₂ leftClosed ⟩
    Ω ⨾ Q
  □
Ω⨾Q : Ω ⨾ Q ≈ ∈ \ (∈ ⨾ Q)
Ω⨾Q = ⨾-cong₂ ˘˘ ⟨≈˘≈⟩ \-outer-⨾-≈ Q˘-isMapping ⟨≈≈⟩ \-cong₂ (⨾-cong₂ ˘˘)
Q˘⨾Ω : Q˘ ⨾ Ω ≈ (∈ ⨾ Q) \ ∈
Q˘⨾Ω = \-inner-⨾ Q˘-isMapping ⟨≈≈⟩ \-cong₁ (⨾-cong₂ ˘˘)
F : Mapping ↑↓-image (ℙ ent)
F = record {mor = Q ˘; prf = Q˘-isMapping}
Q˘⨾-\-Q˘⨾ : {Z₁ Z₂ : Obj} {R : Mor (ℙ ent) Z₁} {S : Mor (ℙ ent) Z₂}
           → (Q˘ ⨾ R) \ (Q˘ ⨾ S) ≈ (inc↑↓₀ ⨾ R) \ (inc↑↓₀ ⨾ S)
Q˘⨾-\-Q˘⨾ {Z₁} {Z₂} {R} {S} = ≈-begin
      (Q˘ ⨾ R) \ (Q˘ ⨾ S)
    ≈˘⟨ \-flip-M Q˘-isMapping ⟩
      (Q˘ ˘ ⨾ Q˘ ⨾ R) \ S
    ≈⟨ \-cong₁ (⨾-cong₁ ˘˘ ⟨≈≈⟩ ⨾-cong₁&₂₁ factors) ⟩
      (inc↑↓₀ ˘ ⨾ inc↑↓₀ ⨾ R) \ S
    ≈⟨ \-flip-M (Mapping.prf inc↑↓) ⟩
      (inc↑↓₀ ⨾ R) \ (inc↑↓₀ ⨾ S)
    □
Q˘⨾/-\-Q˘⨾ : {Z₁ Z₂ Z₃ : Obj} {R : Mor (ℙ ent) Z₂} {S : Mor (ℙ ent) Z₃} {T : Mor Z₁ Z₂}
             → ((Q˘ ⨾ R) / T) \ (Q˘ ⨾ S) ≈ ((inc↑↓₀ ⨾ R) / T) \ (inc↑↓₀ ⨾ S)
Q˘⨾/-\-Q˘⨾ {Z₁} {Z₂} {Z₃} {R} {S} {T} = ≈-begin
      ((Q˘ ⨾ R) / T) \ (Q˘ ⨾ S)
    ≈˘⟨ \-cong₁ (/-outer-⨾-≈ Q˘-isMapping) ⟩
      (Q˘ ⨾ (R / T)) \ (Q˘ ⨾ S)
```

```
    ≈⟨ Q˘⨾-\-Q˘⨾ ⟩
      (inc↑↓₀ ⨾ (R / T)) \ (inc↑↓₀ ⨾ S)
    ≈⟨ \-cong₁ (/-outer-⨾-≈ (Mapping.prf inc↑↓)) ⟩
      ((inc↑↓₀ ⨾ R) / T) \ (inc↑↓₀ ⨾ S)
    □

⨾Q-/-⨾Q : {Z₁ Z₂ : Obj} {R : Mor Z₁ (ℙ ent)} {S : Mor Z₂ (ℙ ent)}
          → (R ⨾ Q) / (S ⨾ Q) ≈ (R ⨾ inc↑↓₀ ˘) / (S ⨾ inc↑↓₀ ˘)
⨾Q-/-⨾Q {Z₁} {Z₂} {R} {S} = ≈-begin
      (R ⨾ Q) / (S ⨾ Q)
    ≈˘⟨ /-flip-˘ Q˘-isMapping ⟩
      R / ((S ⨾ Q) ⨾ Q˘)
    ≈⟨ /-cong₂ (⨾-cong₁₂&₂ factors) ⟩
      R / ((S ⨾ inc↑↓₀ ˘) ⨾ inc↑↓₀)
    ≈⟨ /-flip (Mapping.prf inc↑↓) ⟩
      (R ⨾ inc↑↓₀ ˘) / (S ⨾ inc↑↓₀ ˘)
    □

⨾Q-/-\⨾Q : {Z₁ Z₂ Z₃ : Obj} {R : Mor Z₁ (ℙ ent)} {S : Mor Z₂ (ℙ ent)} {T : Mor Z₂ Z₃}
          → (R ⨾ Q) / (T \ (S ⨾ Q)) ≈ (R ⨾ inc↑↓₀ ˘) / (T \ (S ⨾ inc↑↓₀ ˘))
⨾Q-/-\⨾Q {Z₁} {Z₂} {Z₃} {R} {S} {T} = ≈-begin
      (R ⨾ Q) / (T \ (S ⨾ Q))
    ≈˘⟨ /-cong₂ (\-outer-⨾-≈-˘M Q˘-isMapping) ⟩
      (R ⨾ Q) / ((T \ S) ⨾ Q)
    ≈⟨ ⨾Q-/-⨾Q ⟩
      (R ⨾ inc↑↓₀ ˘) / ((T \ S) ⨾ inc↑↓₀ ˘)
    ≈⟨ /-cong₂ (\-outer-⨾-≈ (Mapping.prf inc↑↓)) ⟩
      (R ⨾ inc↑↓₀ ˘) / (T \ (S ⨾ inc↑↓₀ ˘))
    □

≼ : Mor ↑↓-image ↑↓-image
≼ = Q˘ ⨾ Ω ⨾ Q   -- leave, subset ordering, then come back

≼≈∈⨾Q\∈⨾Q : ≼ ≈ (∈ ⨾ Q) \ (∈ ⨾ Q)
≼≈∈⨾Q\∈⨾Q = ≈-begin
    Q˘ ⨾ Ω ⨾ Q
  ≈⟨ ⨾-cong₂ Ω⨾Q ⟩
    Q˘ ⨾ (∈ \ (∈ ⨾ Q))
  ≈⟨ \-inner-⨾ Q˘-isMapping ⟩
    (∈ ⨾ Q˘ ˘) \ (∈ ⨾ Q)
  ≈⟨ \-cong₁ (⨾-cong₂ ˘˘) ⟩
    (∈ ⨾ Q) \ (∈ ⨾ Q)
  □

≼-˘ : ≼ ˘ ≈ Q˘ ⨾ Ω ˘ ⨾ Q
≼-˘ = ≈-begin
    (Q˘ ⨾ Ω ⨾ Q) ˘
  ≈⟨ ⨾-˘ ⟩
    (Ω ⨾ Q) ˘ ⨾ Q˘ ˘
  ≈⟨ ⨾-cong ⨾-˘ ˘˘ ⟨≈≈⟩ ⨾-assoc ⟩
    Q˘ ⨾ Ω ˘ ⨾ Q
  □
```

open SubOrder {ℙ ent} {Ω} Ω-isOrder {↑↓-image} F Q˘-isInjective **public 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-closed-⊑ : ℙ.glb (R ⨾ Q˘) ⊑ ℙ.glb (R ⨾ Q˘) ⨾ Ξ
glb-closed-⊑ = ℙglb-preserves-↑↓ R⨾Q˘-closed
glb-closed : ℙ.glb (R ⨾ Q˘) ⨾ Ξ ≈ ℙ.glb (R ⨾ Q˘)
glb-closed = ⊑-antisym (proj₂ Ξ-isSubIdentity) glb-closed-⊑
≼-glb-R-simpl₁ : ≼.glb R ⨾ Q˘ ≈ ((Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘) ⨿ (Q˘ ⨾ Ω)) ⨾ Q ⨾ Q˘
≼-glb-R-simpl₁ = ≈-begin
      ≼.glb R ⨾ Q˘
    ≈⟨⟩
      (≼.lbd R ˘ ⨿ ≼) ⨾ Q˘
    ≈⟨ ⨾-cong₁ (⨿-cong ≼-lbd-˘ (⨾-assocL ⟨≈≈˘⟩ ⨾-cong₂ ˘˘)) ⟩
      ((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˘
    ≈⟨ ⨾-cong₂₁ ˘˘ ⟩
      ((Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘) ⨿ (Q˘ ⨾ Ω)) ⨾ Q ⨾ Q˘
    □

≼-glb-R-⊒ : ℙ.glb (R ⨾ Q˘) ⊑ ≼.glb R ⨾ Q˘
≼-glb-R-⊒ = ⊑-begin
      ℙ.glb (R ⨾ Q˘)
    ⊑⟨ glb-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˘
    ≈˘⟨ ≼-glb-R-simpl₁ ⟩
      ≼.glb R ⨾ Q˘
    □

≼-glb-R-⊑ : ≼.glb R ⊑ ℙ.glb (R ⨾ Q˘) ⨾ Q
≼-glb-R-⊑ = ⊑-begin
      ≼.glb R
    ≈⟨ proj₂ splitId ⟨≈˘≈⟩ ⨾-assocL ⟩
      (≼.glb R ⨾ Q˘) ⨾ Q
    ≈⟨ ⨾-cong₁ ≼-glb-R-simpl₁ ⟨≈≈⟩ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ (⨾-assoc ⟨≈≈⟩ proj₂ splitId) ⟩
      ((Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘) ⨿ (Q˘ ⨾ Ω)) ⨾ Q
    ⊑⟨ ˘⨿-universal (⊑-begin
        ℙ.lbd (R ⨾ Q˘) ˘ ⨾ (((Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘) ⨿ (Q˘ ⨾ Ω)) ⨾ Q)
      ⊑⟨ ⨾-monotone₁ (⊑-begin
          ℙ.lbd (R ⨾ Q˘) ˘
        ≈⟨ ˘-cong ℙ.glb-⨾-Ω˘ ⟨≈˘≈⟩ ⨾˘-˘ ⟩
```

```
          Ω ⨾ ℙ.glb (R ⨾ Q˘) ˘
        ⊑⟨ ⨾-monotone₂ (˘-monotone glb-closed-⊑ ⟨⊑≈⟩ ⨾-˘) ⟩
          Ω ⨾ Ξ ˘ ⨾ ℙ.glb (R ⨾ Q˘) ˘
        ⊑⟨ ⨾-monotone₂₂ (˘-monotone ℙ.˘-rightSupId) ⟩
          Ω ⨾ Ξ ˘ ⨾ (ℙ.glb (R ⨾ Q˘) ⨾ Ω ˘) ˘
        ≈⟨ ⨾-cong₂ (⨾-cong (˘⨾-˘ ⟨≈≈˘⟩ factors) (˘-cong ℙ.glb-⨾-Ω˘) ⟨≈≈⟩ ⨾-assoc) ⟩
          Ω ⨾ Q ⨾ Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘
        □) ⟩
        (Ω ⨾ Q ⨾ Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘) ⨾ (((Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘) ⨉ (Q˘ ⨾ Ω)) ⨾ Q)
      ⊑⟨ ⨾-cong₁ ⨾-assocL ⟨≈≈⟩ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ (⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ ⨉-cancel-left) ⟩
        (Ω ⨾ Q) ⨾ (Q˘ ⨾ Ω) ⨾ Q
      ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong (⨾-₂₂assoc₁₂₁ ⟨≈≈⟩ ⨾-cong₂₁ factors) (≈-sym ˘˘) ⟩
        (Ω ⨾ Ξ ⨾ Ω) ⨾ Q˘ ˘
      ⊑⟨ ⨾-monotone₁ (⨾-monotone₂ (proj₁ Ξ-isSubIdentity) ⟨⊑⊑⟩ Ω-trans) ⟩
        Ω ⨾ Q˘ ˘
      □) (⊑-begin
        (((Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘) ⨉ (Q˘ ⨾ Ω)) ⨾ Q) ⨾ (Ω ⨾ Q˘ ˘) ˘
      ≈⟨ ⨾-cong (⨾-cong₂ (≈-sym ˘˘)) ⨾˘-˘ ⟩
        (((Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘) ⨉ (Q˘ ⨾ Ω)) ⨾ Q˘ ˘) ⨾ Q˘ ⨾ Ω ˘
      ⊑⟨ ⨾-monotone₁₂ (˘-monotone ℙ.rightSupId) ⟩
        (((Q˘ ⨾ ℙ.lbd (R ⨾ Q˘) ˘) ⨉ (Q˘ ⨾ Ω)) ⨾ (Q˘ ⨾ Ω) ˘) ⨾ Q˘ ⨾ Ω ˘
      ⊑⟨ ⨾-monotone₁ (⨉-cancel-right ⟨⊑≈⟩ ˘⨾˘-˘) ⟨⊑≈⟩ ⨾-assoc ⟩
        ℙ.lbd (R ⨾ Q˘) ⨾ Q ⨾ Q˘ ⨾ Ω ˘
      ⊑⟨ ⨾-monotone₂ (⨾-assocL ⟨≈⊑⟩ proj₁ Q-isInjective) ⟨⊑≈⟩ ℙ.lbd-downclosed ⟩
        ℙ.lbd (R ⨾ Q˘)
      □) ⟩
      (ℙ.lbd (R ⨾ Q˘)) ˘ ⨉ (Ω ⨾ Q˘ ˘)
    ≈˘⟨ ⨉-M-in-right Q˘-isMapping ⟩
      ((ℙ.lbd (R ⨾ Q˘)) ˘ ⨉ Ω) ⨾ Q˘ ˘
    ≈⟨ ⨾-cong₂ ˘˘ ⟩
      ℙ.glb (R ⨾ Q˘) ⨾ Q
    □
  ≼-glb-R-≈ : ≼.glb R ≈ ℙ.glb (R ⨾ Q˘) ⨾ Q
  ≼-glb-R-≈ = ⊑-antisym ≼-glb-R-⊑ (⊑-begin
      ℙ.glb (R ⨾ Q˘) ⨾ Q
    ⊑⟨ ⨾-monotone₁ ≼-glb-R-⊒ ⟨⊑≈⟩ ⨾-assoc ⟩
      ≼.glb R ⨾ Q˘ ⨾ Q
    ≈⟨ ⨾-cong₂ Q˘⨾Q≈Id ⟨≈≈⟩ rightId ⟩
      ≼.glb R
    □)
  glb-total : isTotal (≼.glb R)
  glb-total = isTotal-from-I (⊑-begin
      Id
    ⊑⟨ ℙ.glbΩ-totalI ⟩
      ℙ.glb (R ⨾ Q˘) ⨾ ℙ.glb (R ⨾ Q˘) ˘
    ⊑⟨ ⨾-monotone ≼-glb-R-⊒ (˘-monotone ≼-glb-R-⊒ ⟨⊑≈⟩ ⨾-˘) ⟩
      (≼.glb R ⨾ Q˘) ⨾ Q˘ ˘ ⨾ ≼.glb R ˘
    ⊑⟨ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ (⨾-assocL ⟨≈⊑⟩ proj₁ Q˘-isInjective) ⟩
      ≼.glb R ⨾ ≼.glb R ˘
    □)


open import Categoric.Category using (module Category₁)
open import Categoric.MapCat using (MapCat)
open Category₁ (MapCat occ)
open import Categoric.Functor
open import Data.AContext.InOCC    occ leftResOp rightResOp powerOp
```



```
open import Data.AContext.Category occ leftResOp rightResOp powerOp
  renaming (_⨾⨾_ to _⨾⨾₂_)
module AContextHomToACSLHom {A B : AContext} (R : AContextHom A B) where
    module A = ToACSL A
    module B = ToACSL B
    module R = AContextHom R
    srcCompat-/ : A.inc / A.inc ⊑ R.mor / R.mor
    srcCompat-/ = /∊\-⊑-→-/-⊑ R.srcCompat-/∊\
    trgCompat-\ : B.inc \ B.inc ⊑ R.mor \ R.mor
    trgCompat-\ = /∊˘\-⊑-→-\-⊑ R.trgCompat-/∊˘\
    srcCompat-/′ : (A.inc / A.inc) \ R.mor ≈ R.mor
    srcCompat-/′ = ⊑-antisym (\-antitone /-isReflexive ⟨⊑≈⟩ Id-\) (⊑-begin
        R.mor
      ≈˘⟨ \S∘S/-S ⟩
        (R.mor / R.mor) \ R.mor
      ⊑⟨ \-antitone srcCompat-/ ⟩
        (A.inc / A.inc) \ R.mor
      □)
    trgCompat-\′ : R.mor / (B.inc \ B.inc) ≈ R.mor
    trgCompat-\′ = ⊑-antisym (/-antitone \-isReflexive ⟨⊑≈⟩ /-Id) (⊑-begin
        R.mor
      ≈˘⟨ S/∘\S-S ⟩
        R.mor / (R.mor \ R.mor)
      ⊑⟨ /-antitone trgCompat-\ ⟩
        R.mor / (B.inc \ B.inc)
      □)
    Φ₀ : Mor B.↑↓-image A.↑↓-image
    Φ₀ = B.Q˘ ⨾ B.inc ↑₀ ⨾ R.mor ↓₀ ⨾ A.Q
    Φ-isUnivalent : isUnivalent Φ₀
    Φ-isUnivalent = ≈-isUnivalent ⨾-assocL₃₊₁ (⨾-isUnivalent (Mapping.unival (B.F ⨾₁ B.inc ↑ ⨾₁ R.mor ↓)) A.Q-isUnivalent)
    Φ-isTotal : isTotal Φ₀
    Φ-isTotal = ≈-isTotal ⨾-assocL₃₊₁ (⨾-isTotal-local (Mapping.total (B.F ⨾₁ B.inc ↑ ⨾₁ R.mor ↓))
      (⊑-begin
        B.Q˘ ⨾ B.inc ↑₀ ⨾ R.mor ↓₀
      ≈˘⟨ ⨾-cong₂₂ R.srcCompat′ ⟩
        B.Q˘ ⨾ B.inc ↑₀ ⨾ R.mor ↓₀ ⨾ A.Ξ
      ≈⟨ ⨾-assocL₃₊₁ ⟨≈≈˘⟩ ⨾-cong₂ A.factors ⟩
        (B.Q˘ ⨾ B.inc ↑₀ ⨾ R.mor ↓₀) ⨾ A.Q ⨾ A.Q˘
      □))
    Φ : Mapping B.↑↓-image A.↑↓-image
    Φ = record {mor = Φ₀; prf = Φ-isUnivalent, Φ-isTotal}
    Φ-momotone : B.≼ ⨾ Φ₀ ⊑ Φ₀ ⨾ A.≼
    Φ-momotone = mappingHom₁-from-L (Φ-isUnivalent, Φ-isTotal) (⊑-begin
        B.≼ ˘ ⨾ B.Q˘ ⨾ B.inc ↑₀ ⨾ R.mor ↓₀ ⨾ A.Q
      ≈⟨ ⨾-cong₁ B.≼-˘ ⟨≈≈⟩ ⨾-assoc₃₊₁ ⟨≈≈⟩ ⨾-cong₂₂ ⨾-assocL ⟩
        B.Q˘ ⨾ Ω ˘ ⨾ (B.Q ⨾ B.Q˘) ⨾ B.inc ↑₀ ⨾ R.mor ↓₀ ⨾ A.Q
      ≈⟨ ⨾-cong₂₂ (⨾-cong₁ B.factors ⟨≈≈⟩ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ (⨾-assocL ⟨≈≈⟩ ⨾-cong₁ ↑↓⨾↑)) ⟩
        B.Q˘ ⨾ Ω ˘ ⨾ B.inc↑↓₀ ˘ ⨾ B.inc ↑₀ ⨾ R.mor ↓₀ ⨾ A.Q
      ⊑⟨ ⨾-monotone₂ (⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ (⨾-monotone₂ (˘-monotone ↑↓⊑Ω) ⟨⊑⊑⟩ Ω˘-trans)) ⟩
        B.Q˘ ⨾ Ω ˘ ⨾ B.inc ↑₀ ⨾ R.mor ↓₀ ⨾ A.Q
      ⊑⟨ ⨾-monotone₂ (⨾-monotone₁&₂₁ Ω˘⨾↑) ⟩
        B.Q˘ ⨾ B.inc ↑₀ ⨾ Ω ⨾ R.mor ↓₀ ⨾ A.Q
      ⊑⟨ ⨾-monotone₂₂ (⨾-monotone₁&₂₁ Ω⨾↓) ⟩
        B.Q˘ ⨾ B.inc ↑₀ ⨾ R.mor ↓₀ ⨾ Ω ˘ ⨾ A.Q
      ≈˘⟨ ⨾-cong₂₂₁ (⨾-cong₂ A.factors ⟨≈≈⟩ R.srcCompat′) ⟩
        B.Q˘ ⨾ B.inc ↑₀ ⨾ (R.mor ↓₀ ⨾ A.Q ⨾ A.Q˘) ⨾ Ω ˘ ⨾ A.Q
```

```
        ≈⟨ ⨾-cong22 (⨾-assoc3+1 ⟨≈≈⟩ ⨾-assocL) ⟨≈≈⟩ ⨾-assocL3+1 ⟨≈≈˘⟩ ⨾-cong2 A.≼-˘ ⟩
          (B.Q˘ ⨾ B.inc ↑0 ⨾ R.mor ↓0 ⨾ A.Q) ⨾ A.≼ ˘
        □)
    Φ-continuous : {I : Obj} {S : Mor I B.↑↓-image} → B.≼.glb S ⨾ Φ0 ≈ A.≼.glb (S ⨾ Φ0)
    Φ-continuous {I} {S} = ≈-begin
          B.≼.glb S ⨾ B.Q˘ ⨾ B.inc ↑0 ⨾ R.mor ↓0 ⨾ A.Q
        ≈⟨ ⨾-cong1 B.≼-glb-R-≈ ⟨≈≈⟩ ⨾-22assoc121 ⟨≈≈⟩ ⨾-cong21 B.factors ⟩
          ℙ.glb (S ⨾ B.Q˘) ⨾ B.Ξ ⨾ B.inc ↑0 ⨾ R.mor ↓0 ⨾ A.Q
        ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong1 B.glb-closed ⟩
          ℙ.glb (S ⨾ B.Q˘) ⨾ B.inc ↑0 ⨾ R.mor ↓0 ⨾ A.Q
        ≈⟨ ⨾-cong1 (ℙ.glb-cong (B.R⨾Q˘-closed ⟨≈˘≈⟩ ⨾-assocL ⟨≈≈⟩ ⨾-cong1 ⨾-assoc)) ⟩
          ℙ.glb ((S ⨾ B.Q˘ ⨾ B.inc ↑0) ⨾ B.inc ↓0) ⨾ B.inc ↑0 ⨾ R.mor ↓0 ⨾ A.Q
        ≈⟨ ⨾-cong1 (↓-lub-cocontinuous B.inc (S ⨾ B.Q˘ ⨾ B.inc ↑0)) ⟨≈˘≈⟩ ⨾-assoc ⟩
          ℙ.lub (S ⨾ B.Q˘ ⨾ B.inc ↑0) ⨾ B.inc ↓0 ⨾ B.inc ↑0 ⨾ R.mor ↓0 ⨾ A.Q
        ≈⟨ ⨾-cong2 (⨾-assocL ⟨≈≈⟩ ⨾-assocL ⟨≈≈⟩ ⨾-cong1 R.trgCompat) ⟩
          ℙ.lub (S ⨾ B.Q˘ ⨾ B.inc ↑0) ⨾ R.mor ↓0 ⨾ A.Q
        ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong1 (↓-lub-cocontinuous R.mor (S ⨾ B.Q˘ ⨾ B.inc ↑0) ⟨≈≈⟩ ℙ.glb-cong ⨾-assoc3+1) ⟩
          ℙ.glb (S ⨾ B.Q˘ ⨾ B.inc ↑0 ⨾ R.mor ↓0) ⨾ A.Q
        ≈˘⟨ ⨾-cong1 (ℙ.glb-cong (⨾-cong222 R.srcCompat′)) ⟩
          ℙ.glb (S ⨾ B.Q˘ ⨾ B.inc ↑0 ⨾ R.mor ↓0 ⨾ A.Ξ) ⨾ A.Q
        ≈˘⟨ A.≼-glb-R-≈ ⟨≈≈⟩ ⨾-cong1 (ℙ.glb-cong (⨾-assoc3+1 ⟨≈≈⟩ ⨾-cong22 (⨾-assoc3+1 ⟨≈≈⟩ ⨾-cong22 A.factors))) ⟩
          A.≼.glb (S ⨾ B.Q˘ ⨾ B.inc ↑0 ⨾ R.mor ↓0 ⨾ A.Q)
        □

    Φ1 : ACSLHom (toACSL B) (toACSL A)
    Φ1 = record {map = Φ; monotone = Φ-momotone; continuous = Φ-continuous}

Ctx→CSL : Functor ACH-Category ACSL-OpCategory
Ctx→CSL = record
  {obj        = toACSL
  ; mor       = mor
  ; mor-cong  = λ {A} {B} {R} {S} → mor-cong {A} {B} {R} {S}
  ; mor-⨾     = λ {A} {B} {C} {R} {S} → mor-⨾ {A} {B} {C} {R} {S}
  ; mor-Id    = mor-Id
  }
  where
    mor = AContextHomToACSLHom.Φ1
    mor-cong : {A B : AContext} {R S : AContextHom A B} → R ≋2 S → mor R ≋1 mor S
    mor-cong {A} {B} {R} {S} R≈S = let
        module A = ToACSL A
        module B = ToACSL B
        module R = AContextHom R
        module S = AContextHom S
      in ≈-begin
        B.Q˘ ⨾ B.inc ↑0 ⨾ R.mor ↓0 ⨾ A.Q
      ≈⟨ ⨾-cong221 (↓-cong R≈S) ⟩
        B.Q˘ ⨾ B.inc ↑0 ⨾ S.mor ↓0 ⨾ A.Q
      □
    mor-⨾ : {A B C : AContext} {R : AContextHom A B} {S : AContextHom B C}
          → mor (R ⨾⨾2 S) ≋1 mor S ⨾⨾1 mor R
    mor-⨾ {A} {B} {C} {R} {S} = let
        module A = ToACSL A
        module B = ToACSL B
        module C = ToACSL C
        module R = AContextHom R
        module S = AContextHom S
        module RS = AContextHom-Comp R S
      in ≈-begin
```



```
        C.Q˘ ⨾ C.inc ↑0 ⨾ [S.mor ↓ ⨾1 B.inc ↑ ⨾1 R.mor ↓] ↓0 ⨾ A.Q
      ≈⟨ ⨾-cong22 (⨾-cong1 RS. [⨾⨾] ↓ ⟨≈≈⟩ ⨾-assoc3+1) ⟩
        C.Q˘ ⨾ C.inc ↑0 ⨾ S.mor ↓0 ⨾ B.inc ↑0 ⨾ R.mor ↓0 ⨾ A.Q
      ≈⟨ ⨾-cong22 (⨾-cong1 (S.srcCompat′ ⟨≈˘≈˘⟩ ⨾-cong2 B.factors) ⟨≈≈⟩ ⨾-assoc3+1) ⟩
        C.Q˘ ⨾ C.inc ↑0 ⨾ S.mor ↓0 ⨾ B.Q ⨾ B.Q˘ ⨾ B.inc ↑0 ⨾ R.mor ↓0 ⨾ A.Q
      ≈⟨ ⨾-cong22 ⨾-assocL ⟨≈≈⟩ ⨾-assocL3+1 ⟩
        (C.Q˘ ⨾ C.inc ↑0 ⨾ S.mor ↓0 ⨾ B.Q) ⨾ (B.Q˘ ⨾ B.inc ↑0 ⨾ R.mor ↓0 ⨾ A.Q)
      □
    mor-Id : {A : AContext} → mor (AContext-Id {A}) ≋1 ACSL-Id
    mor-Id = λ {A} → let module A = ToACSL A in ≈-begin
        A.Q˘ ⨾ A.inc ↑0 ⨾ A.inc ↓0 ⨾ A.Q
      ≈⟨ ⨾-cong (≈-sym leftId) ⨾-assocL ⟨≈≈⟩ ⨾-assocL ⟩
        ((Id ⨾ A.Q˘) ⨾ (A.inc ↑0 ⨾ A.inc ↓0)) ⨾ A.Q
      ≈⟨ ⨾-cong1 A.R⨾Q˘-closed ⟩
        (Id ⨾ A.Q˘) ⨾ A.Q
      ≈⟨ ⨾-cong1 leftId ⟨≈≈⟩ A.Q˘⨾Q≈Id ⟩
        Id
      □
```

Chapter 9

Duality between Contexts and Complete Lower Semilattices

Since typechecking the categoric duality induced by the two functors CSL→Ctx and Ctx→CSL from the previous two sections Categoric.OCC.CSL.ToAContext (Sect. 8.2) and Categoric.OCC.CSL.FromAContext (Sect. 8.3) requires significant heap space, stack space, and time, we split this into several modules. For each of the two functor compositions, the *NatIsoPieces modules contains the definition of the indexed morphism of the respective natural transformation between the composed functor and the identity functor, as well as its inverse morphism together with the inverse proofs, which all require only a single base object, while *NatIsoNaturality contains the morphisms naturality, which affects two base objects. In Categoric.OCC.CSL.ContextDualityPieces (Sect. 9.5), all these are collected and re-exported; this is currently our machine-checked proof of the duality between the abstract contexts and abstract complete lower semilattices, and is referenced in the top-level module AContext2 included in Chapter 1, which type-checks together with the whole development it references in less than an hour.

9.1 Categoric.OCC.CSL.ToFromAContextNatIsoPieces

```
open import RATH.Level
open import RATH.Data.Product using (Σ:•; proj₁; proj₂; _,_)
open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.MapCat using (MapCat)
open import Categoric.OSGC.PowerOp
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OSGC.SyQ.WithResiduals
open import Categoric.OCC.SyQ
open import Categoric.Category
open import Categoric.Functor

module Categoric.OCC.CSL.ToFromAContextNatIsoPieces {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)
private
  module ℙ = DirectPower directPower
open ℙ using (powerOp; ℙ
  ; ∈; ∈⨾-∈\
  ; Λ₀; Λ-cong; Λ-isMapping    -- ; Λ-map⨾
  ; Glb≈glb
  ; Ω; Ω-isOrder; Ω-trans; Ω-refl; Ω⟋Ω; Ω˘-trans; Ω\Ω≈Ω)
open PowerOp                                  osgc powerOp
  using (Λ⨾∈˘)
open import Categoric.OSGC.PowerOrder         osgc leftResOp rightResOp powerOp
  using (Lub₀; Glb₀; Glb; Lub-cong; Glb-cong)
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-↑↓; ↓-lub-cocontinuous)
open import Data.AContext.InOCC     occ leftResOp rightResOp powerOp
open import Data.AContext.Category occ leftResOp rightResOp powerOp
open import Categoric.OCC.CSL.FromAContext occ leftResOp rightResOp syqOp directPower splitSymIdempot
open import Categoric.OCC.CSL.ToAContext     occ leftResOp rightResOp syqOp directPower

open Category₁ (MapCat occ)
  -- module 𝒞₂ = Category ACH-Category
  -- open Category₂ ACH-Category
private
  module To = Functor CSL→Ctx
  module From = Functor Ctx→CSL
  module 𝒞₃ = Category ACSL-OpCategory
open Category₃ ACSL-OpCategory

module ACSL-ToFromAContext {A : ACSL} where
  module A = ACSL′ A
  C : AContext
  C = To.obj A
  module C = ToACSL C
  L₀ : Mor A.Carrier C.↑↓-image
  L₀ = A.downset₀ ⨾ C.Q
  downset-closed′ : A.downset₀ ⨾ C.Q ⨾ C.Q ˘ ≈ A.downset₀
  downset-closed′ = ⨾-cong₂ C.factors ⟨≈≈⟩ A.downset-closed
  L-isMapping : isMapping L₀
  L-isMapping = ⨾-isUnivalent (Mapping.unival A.downset) C.Q-isUnivalent
                  , isTotal-from-I (⊑-begin
                       Id
                     ⊑⟨ mappingTotalI A.downset ⟩
                       A.downset₀ ⨾ A.downset₀ ˘
                     ≈⟨ ⨾-cong₁ downset-closed′ ⟨≈˘≈⟩ ⨾-assoc₃₊₁ ⟩
```

```
                A.downset₀ ⨾ C.Q ⨾ C.Q ˘ ⨾ A.downset₀ ˘
              ≈⟨ ⨾-assocL ⟨≈≈˘⟩ ⨾-cong₂ ⨾-˘ ⟩
                (A.downset₀ ⨾ C.Q) ⨾ (A.downset₀ ⨾ C.Q) ˘
              □)
L : Mapping A.Carrier C.↑↓-image
L = record {mor = L₀; prf = L-isMapping}
R : Mapping C.↑↓-image A.Carrier
R = A.lub-Mapping (C.Q˘ ⨾ ∈ ˘)
R₀ : Mor C.↑↓-image A.Carrier
R₀ = Mapping.mor R
R-isMapping = Mapping.prf R


L-continuous : {I : Obj} {S : Mor I A.Carrier} → A.glb S ⨾ L₀ ≈ C.≼.glb (S ⨾ L₀)
L-continuous {I} {S} = ≈-begin
      A.glb S ⨾ L₀
    ≈⟨ ⨾-assocL ⟩
      (A.glb S ⨾ A.downset₀) ⨾ C.Q
    ≈⟨ ⨾-cong₁ (≈-begin
          A.glb S ⨾ A.downset₀
        ≈⟨ A.glb⨾downset ⟩
          Λ₀ (A.lbd S)
        ≈⟨ Λ-cong (≈-begin
              A.lbd S
            ≈˘⟨ /-˘ ⟩
              (A.≼ / S) ˘
            ≈⟨ ˘-cong (⊑-antisym (/-universal (⊑-begin
                (A.≼ / S) ⨾ (S ⨾ A.downset₀)
              ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-outer ⟩
                A.≼ ⨾ ((A.≼ ˘) ˘ )( ∈)
              ⊑⟨ ⨾-cong₂ ()(-cong₁ ˘˘) ⟨≈⊑⟩ )(-cancel-left ⟩
                ∈
              □))(/-universal (⊑-begin
                (∈ / (S ⨾ A.downset₀)) ⨾ S
              ≈˘⟨ ⨾-cong₁ // ⟩
                ((∈ / A.downset₀) / S) ⨾ S
              ⊑⟨ /-cancel-outer ⟩
                ∈ / A.downset₀
              ≈⟨ /-cong₂ rightId ⟨≈˘≈⟩ (/-inner-⨾ Λ-isMapping ⟨≈˘≈⟩ ⨾-cong₁ /-Id) ⟩
                ∈ ⨾ A.downset₀ ˘
              ≈⟨ ⨾˘-˘ ⟨≈˘≈⟩ ≈-˘-swap Λ⨾∈˘ ⟩
                A.≼
              □))) ⟩
              (∈ / (S ⨾ A.downset₀)) ˘
            ≈⟨ /-˘ ⟩
              (S ⨾ A.downset₀) ˘ \ ∈ ˘
            □) ⟩
          Glb₀ (S ⨾ A.downset₀)
        ≈⟨ Glb≈glb ⟩
          ℙ.glb (S ⨾ A.downset₀)
        ≈˘⟨ ℙ.glb-cong (⨾-assoc₃₊₁ ⟨≈≈⟩ ⨾-cong₂ downset-closed′) ⟩
          ℙ.glb ((S ⨾ A.downset₀ ⨾ C.Q) ⨾ C.Q˘)
        □) ⟩
      ℙ.glb ((S ⨾ A.downset₀ ⨾ C.Q) ⨾ C.Q˘) ⨾ C.Q
    ≈˘⟨ C.≼-glb-R-≈ ⟩
      C.≼.glb (S ⨾ A.downset₀ ⨾ C.Q)
    ≈⟨⟩
```



```
      C.≼.glb (S ⨾ L₀)
    □

R-continuous : {I : Obj} {S : Mor I C.↑↓-image} → C.≼.glb S ⨾ R₀ ≈ A.glb (S ⨾ R₀)
R-continuous {I} {S} = ≈-begin
      C.≼.glb S ⨾ A.lub (C.Q˘ ⨾ ∈ ˘)
    ≈⟨ ⨾-cong₂ (A.lub-map⨾ C.Q˘-isMapping) ⟩
      C.≼.glb S ⨾ C.Q˘ ⨾ A.lub (∈ ˘)
    ≈⟨ ⨾-cong₁ C.≼-glb-R-≈ ⟨≈≈⟩ ⨾-₂₂assoc₁₂₁ ⟩
      ℙ.glb (S ⨾ C.Q˘) ⨾ (C.Q ⨾ C.Q˘) ⨾ A.lub (∈ ˘)
    ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ (⨾-cong₂ C.factors ⟨≈≈⟩ C.glb-closed) ⟩
      ℙ.glb (S ⨾ C.Q˘) ⨾ A.lub (∈ ˘)
    ≈˘⟨ ⨾-cong₁ Glb≈glb ⟩
      Glb₀ (S ⨾ C.Q˘) ⨾ A.lub (∈ ˘)
    ≈⟨ ⨾-cong₁ ()(-cong₁ ˘\˘-˘) ⟩
      ((∈ / (S ⨾ C.Q˘)) )( ∈) ⨾ A.lub (∈ ˘)
    ≈⟨ ⨾-cong₁ ()(-cong₁ (⊑-antisym (/-universal (⊑-begin
            (∈ / (S ⨾ C.Q˘)) ⨾ ((S ⨾ C.Q˘) ⨾ (∈ \ A.≼))
          ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-outer ⟩
            ∈ ⨾ (∈ \ A.≼)
          ⊑⟨ \-cancel-outer ⟩
            A.≼
          □))(/-universal (⊑-begin
            (A.≼ / ((S ⨾ C.Q˘) ⨾ (∈ \ A.≼))) ⨾ (S ⨾ C.Q˘)
          ≈˘⟨ ⨾-cong₁ // ⟩
            ((A.≼ / (∈ \ A.≼)) / (S ⨾ C.Q˘)) ⨾ (S ⨾ C.Q˘)
          ≈⟨ ⨾-cong₂ (⨾-cong₂ C.Q˘-⨾-inc↑↓ ⟨≈˘≈⟩ ⨾-assocL) ⟩
            ((A.≼ / (∈ \ A.≼)) / (S ⨾ C.Q˘)) ⨾ (S ⨾ C.Q˘) ⨾ A.≼ ↑↓₀
          ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-outer ⟩
            (A.≼ / (∈ \ A.≼)) ⨾ A.≼ ↑↓₀
          ≈⟨ ⨾-cong₁ ∈⨾↑↓˘ ⟨≈˘≈⟩ ⨾-assoc ⟩
            ∈ ⨾ A.≼ ↑↓₀ ˘ ⨾ A.≼ ↑↓₀
          ⊑⟨ proj₂ (Mapping.unival (A.≼ ↑↓)) ⟩
            ∈
          □)))) ⟩
      ((A.≼ / ((S ⨾ C.Q˘) ⨾ (∈     \ A.≼))) )( ∈) ⨾ A.lub (∈ ˘)
    ≈˘⟨ ⨾-cong₁ ()(-cong₁ (// ⟨≈≈⟩ /-cong₂ (⨾-cong₂ (\-cong₁ ˘˘)))) ⟩
      (((A.≼ / ((∈ ˘) ˘ \ A.≼)) / (S ⨾ C.Q˘)) )( ∈) ⨾ A.lub (∈ ˘)
    ≈˘⟨ ⨾-cong₁ ()(-cong₁ (/-cong₁ (A.order-⨾-total-lub˘ (A.lub-total _)))) ⟩
      (((A.≼ ⨾ A.lub (∈ ˘) ˘) / (S ⨾ C.Q˘)) )( ∈) ⨾ A.lub (∈ ˘)
    ≈˘⟨ ⨾-cong₁ ()(-cong₁ (/-cong₂ ⨾-assocL ⟨≈≈⟩ /-flip (A.lub-isMapping _))) ⟩
      ((A.≼ / ((S ⨾ C.Q˘ ⨾ A.lub (∈ ˘)))) )( ∈) ⨾ A.lub (∈ ˘)
    ≈˘⟨ ⨾-cong₁ ()(-cong₁ ˘\˘-˘) ⟩
      Λ₀ ((S ⨾ C.Q˘ ⨾ A.lub (∈ ˘)) ˘ \ A.≼ ˘) ⨾ A.lub (∈ ˘)
    ≈⟨⟩
      Λ₀ (A.lbd (S ⨾ C.Q˘ ⨾ A.lub (∈ ˘))) ⨾ A.lub (∈ ˘)
    ≈⟨ ⨾-cong₁ A.glb⨾downset ⟨≈˘≈⟩ ⨾-assoc ⟩
      A.glb (S ⨾ C.Q˘ ⨾ A.lub (∈ ˘)) ⨾ A.downset₀ ⨾ A.lub (∈ ˘)
    ≈⟨ ⨾-cong₂ A.downset-⨾-lub∈˘ ⟨≈≈⟩ rightId ⟩
      A.glb (S ⨾ C.Q˘ ⨾ A.lub (∈ ˘))
    ≈˘⟨ A.glb-cong (⨾-cong₂ (A.lub-map⨾ C.Q˘-isMapping)) ⟩
      A.glb (S ⨾ A.lub (C.Q˘ ⨾ ∈ ˘))
    □


L : ACSLHom A (From.obj (To.obj A))
L = record
```

```
    {map = L
    ; monotone = ⊑-begin
          A.≼ ⨾ L₀
        ≈⟨⟩
          A.≼ ⨾ A.downset₀ ⨾ C.Q
        ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ (\-universal (⊑-begin
                  A.≼ ⨾ A.≼ ⨾ A.downset₀
                ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong A.idempot (╳-cong₁ ˘˘) ⟩
                  A.≼ ⨾ (A.≼ ╳ ∈)
                ⊑⟨ ╳-cancel-left ⟩
                  ∈
                □)) ⟩
          (A.≼ \ ∈) ⨾ C.Q
        ≈⟨ ⨾-cong₁ A.downset-⨾-Ω ⟨≈˘≈⟩ ⨾-assoc ⟩
          A.downset₀ ⨾ Ω ⨾ C.Q
        ≈⟨ ⨾-cong₁ A.downset-closed ⟨≈˘≈⟩ ⨾-assoc ⟩
          A.downset₀ ⨾ (A.≼ ↑↓₀ ˘ ⨾ A.≼ ↑↓₀) ⨾ Ω ⨾ C.Q
        ≈⟨ ⨾-cong₂₁ C.factors ⟨≈˘≈⟩ ⨾-₁₂₁assoc₂₂ ⟩
          (A.downset₀ ⨾ C.Q) ⨾ C.Q˘ ⨾ Ω ⨾ C.Q
        ≈⟨⟩
          L₀ ⨾ C.≼
        □
    ; continuous = L-continuous
    }


L⁻¹ : ACSLHom (From.obj (To.obj A)) A
L⁻¹ = record
  {map = R
  ; monotone = ⊑-begin
        C.≼ ⨾ R₀
      ≈⟨ ⨾-assoc ⟩
        C.Q˘ ⨾ (Ω ⨾ C.Q) ⨾ A.lub (C.Q˘ ⨾ ∈ ˘)
      ⊑⟨ ⨾-monotone₂ (⊑-begin
          (Ω ⨾ C.Q) ⨾ A.lub (C.Q˘ ⨾ ∈ ˘)
        ≈⟨ ⨾-cong₂ (A.lub-map⨾ C.Q˘-isMapping) ⟨≈≈⟩ (⨾-cong₁₂&₂₁ (C.factors ⟨≈≈˘⟩ A.downset-closed₀) ⟨≈≈⟩ ⨾-₁₂₁assoc₂₂) ⟩
          (Ω ⨾ A.downset₀ ˘) ⨾ A.downset₀ ⨾ A.lub (∈ ˘)
        ≈⟨ ⨾-cong₂ A.downset-⨾-lub∈˘ ⟨≈≈⟩ rightId ⟩
          Ω ⨾ A.downset₀ ˘
        ⊑⟨ \-universal (⊑-begin
              ∈ ⨾ Ω ⨾ A.downset₀ ˘
            ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ \-cancel-outer ⟩
              ∈ ⨾ A.downset₀ ˘
            ≈⟨ ⨾-cong₂ ╳-˘ ⟩
              ∈ ⨾ (∈ ╳ A.≼ ˘ ˘)
            ⊑⟨ ╳-cancel-left ⟨⊑≈⟩ ˘˘ ⟩
              A.≼
            □) ⟩
          ∈ \ A.≼
        ≈˘⟨ \-cong₁ ˘˘ ⟩
          A.ubd (∈ ˘)
        ≈˘⟨ A.total-lub-⨾-order (A.lub-total _) ⟩
          A.lub (∈ ˘) ⨾ A.≼
        □) ⟩
        C.Q˘ ⨾ A.lub (∈ ˘) ⨾ A.≼
      ≈⟨ ⨾-assocL ⟨≈≈˘⟩ ⨾-cong₁ (A.lub-map⨾ C.Q˘-isMapping) ⟩
        A.lub (C.Q˘ ⨾ ∈ ˘) ⨾ A.≼
```



```
      ≈⟨⟩
        R₀ ⨾ A.≼
      □
  ; continuous = R-continuous
  }


L-⨾-L⁻¹ : L ⨾₃ L⁻¹ ≈₃ Id₃ {From.obj (To.obj A)}
L-⨾-L⁻¹ = ≈-begin
    A.lub (C.Q˘ ⨾ ∈ ˘) ⨾ (A.downset₀ ⨾ C.Q)
  ≈⟨ ⨾-cong₁ (A.lub-map⨾ C.Q˘-isMapping) ⟨≈≈⟩ ⨾-₂₂assoc₁₂₁ ⟩
    C.Q˘ ⨾ (A.lub (∈ ˘) ⨾ A.downset₀) ⨾ C.Q
  ≈⟨ ⨾-cong₂₁ A.≼↑↓-≈-lub∋⨾downset ⟨≈˘≈⟩ ⨾-assocL ⟩
    (C.Q˘ ⨾ A.≼ ↑↓₀) ⨾ C.Q
  ≈⟨ ⨾-cong₁ (˘⨾-˘ ⟨≈˘≈⟩ ˘-cong C.inc↑↓˘-⨾-Q) ⟨≈≈⟩ C.Q˘⨾Q≈Id ⟩
    Id
  □


L⁻¹-⨾-L : L⁻¹ ⨾₃ L ≈₃ Id₃ {A}
L⁻¹-⨾-L = ≈-begin
    (A.downset₀ ⨾ C.Q) ⨾ A.lub (C.Q˘ ⨾ ∈ ˘)
  ≈⟨ ⨾-cong₂ (A.lub-map⨾ C.Q˘-isMapping) ⟨≈≈⟩ ⨾-₂₂assoc₁₂₁ ⟩
    A.downset₀ ⨾ (C.Q ⨾ C.Q˘) ⨾ A.lub (∈ ˘)
  ≈⟨ ⨾-cong₂ (⨾-cong₁ C.factors) ⟨≈≈⟩ ⨾-₁₂₁assoc₂₂ ⟩
    (A.downset₀ ⨾ A.≼ ↑↓₀ ˘) ⨾ (A.≼ ↑↓₀ ⨾ A.lub (∈ ˘))
  ≈⟨ ⨾-cong (⨾-cong₂ (˘-cong A.≼↑↓-≈-lub∋⨾downset ⟨≈≈⟩ ⨾-˘) ⟨≈≈⟩ ⨾-assocL)
    (⨾-cong₁ A.≼↑↓-≈-lub∋⨾downset ⟨≈≈⟩ ⨾-assoc) ⟩
    ((A.downset₀ ⨾ A.downset₀ ˘) ⨾ A.lub (∈ ˘) ˘) ⨾ (A.lub (∈ ˘) ⨾ A.downset₀ ⨾ A.lub (∈ ˘))
  ≈⟨ ⨾-cong₂ (⨾-cong₂ A.downset-⨾-lub∈˘ ⟨≈≈⟩ rightId) ⟩
    ((A.downset₀ ⨾ A.downset₀ ˘) ⨾ A.lub (∈ ˘) ˘) ⨾ A.lub (∈ ˘)
  ≈⟨ ⨾-cong₁ (⊑-antisym (proj₁ A.downset-isInjective) (proj₁ (Mapping.total A.downset))) ⟩
    A.lub (∈ ˘) ˘ ⨾ A.lub (∈ ˘)
  ≈⟨ A.lub∈˘-˘-⨾-lub∈˘ ⟩
    Id
  □
```

open ACSL-ToFromAContext **public using** $(\mathbf{L}; \mathbf{L}^{-1}; \mathbf{L}\text{-⨾-}\mathbf{L}^{-1}; \mathbf{L}^{-1}\text{-⨾-}\mathbf{L})$

9.2 Categoric.OCC.CSL.ToFromAContextNatIsoNaturality

```
open import RATH.Level

open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
  -- open import Categoric.MapCat using (MapCat)

open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OSGC.SyQ.WithResiduals
open import Categoric.OCC.SyQ

open import Categoric.Category
open import Categoric.Functor


module Categoric.OCC.CSL.ToFromAContextNatIsoNaturality {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)
open DirectPower directPower using (powerOp; ∈)
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 (↓≈╳)
open import Data.AContext.InOCC    occ leftResOp rightResOp powerOp
open import Data.AContext.Category occ leftResOp rightResOp powerOp
open import Categoric.OCC.CSL.FromAContext occ leftResOp rightResOp syqOp directPower splitSymIdempot
open import Categoric.OCC.CSL.ToAContext     occ leftResOp rightResOp syqOp directPower
open import Categoric.OCC.CSL.ToFromAContextNatIsoPieces
                                             occ leftResOp rightResOp syqOp directPower splitSymIdempot


  -- open Category₁ (MapCat occ)
  -- module 𝒞₂ = Category ACH-Category
  -- open Category₂ ACH-Category
private
  module To = Functor CSL→Ctx
  module From = Functor Ctx→CSL
    -- module 𝒞₃ = Category ACSL-OpCategory
  module 𝒞₄ = Category ACSL-Category
  -- open Category₃ ACSL-OpCategory
open Category₄ ACSL-Category


module _ {A B : ACSL} {F : ACSLHom B A} where
  private
    module F = ACSLHom F
    module B where
      open ACSL-ToFromAContext {B} public
      open A public
    module A where
      open ACSL-ToFromAContext {A} public
      open A public
  L-naturality :
                  -- From.mor (To.mor F) ⨾₃ L {B} ≈₃ L {A} ⨾₃ F – desired statement in ACSL-OpCategory
                  -- L {B} ⨾₄ From.mor (To.mor F) ≈₄ F ⨾₄ L {A} – translated into ACSL-Category
                  (B.downset₀ ⨾ B.C.Q) ⨾ (B.C.Q˘ ⨾ B.≼ ↑₀ ⨾ (A.≼ ⨾ F.map₀ ˘) ↓₀ ⨾ A.C.Q)
```



```
                  ≈ F.map₀ ⨾ A.downset₀ ⨾ A.C.Q   -- expanded into underlying SPOCC
  L-naturality = ≈-begin
        (B.downset₀ ⨾ B.C.Q) ⨾ (B.C.Q˘ ⨾ B.≼ ↑₀ ⨾ (A.≼ ⨾ F.map₀ ˘) ↓₀ ⨾ A.C.Q)
      ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ B.C.factors ⟨≈≈⟩ B.downset-closed) ⟩
        B.downset₀ ⨾ B.≼ ↑₀ ⨾ (A.≼ ⨾ F.map₀ ˘) ↓₀ ⨾ A.C.Q
      ≈⟨ ⨾-cong₂ (⨾-assocL ⟨≈≈⟩ ⨾-cong₁ (⨾-cong₂ ↓≈╳ ⟨≈≈⟩ ╳-in-left (Mapping.prf (B.≼ ↑)))) ⟩
        B.downset₀ ⨾ ((((A.≼ ⨾ F.map₀ ˘) / ∈ ˘) ⨾ B.≼ ↑₀ ˘) ╳ ∈) ⨾ A.C.Q
      ≈⟨ ⨾-cong₂₁ (╳-cong₁ (/-inner-⨾ (Mapping.prf (B.≼ ↑)))) ⟩
        B.downset₀ ⨾ (((A.≼ ⨾ F.map₀ ˘) / (B.≼ ↑₀ ⨾ ∈ ˘)) ╳ ∈) ⨾ A.C.Q
      ≈⟨ ⨾-cong₂₁ (╳-cong₁ (/-cong₂ B.≼↑⨾∈˘)) ⟩
        B.downset₀ ⨾ (((A.≼ ⨾ F.map₀ ˘) / (B.ubd (∈ ˘))) ╳ ∈) ⨾ A.C.Q
      ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ (╳-in-left (Mapping.prf B.downset) ⟨≈≈⟩ ╳-cong₁ (/-inner-⨾ (Mapping.prf B.downset))) ⟩
        (((A.≼ ⨾ F.map₀ ˘) / (B.downset₀ ⨾ B.ubd (∈ ˘))) ╳ ∈) ⨾ A.C.Q
      ≈⟨ ⨾-cong₁ (╳-cong₁ (/-cong₂ B.downset-⨾-ubd∈˘)) ⟩
        (((A.≼ ⨾ F.map₀ ˘) / B.≼) ╳ ∈) ⨾ A.C.Q
      ≈⟨ ⨾-cong₁ (╳-cong₁ F.≼⨾map˘/≼) ⟩
        ((A.≼ ⨾ F.map₀ ˘) ╳ ∈) ⨾ A.C.Q
      ≈⟨ ⨾-cong₁ (╳-in-left (Mapping.prf F.map) ⟨≈˘≈˘⟩ ⨾-cong₂ (╳-cong₁ ˘˘)) ⟨≈≈⟩ ⨾-assoc ⟩
        F.map₀ ⨾ A.downset₀ ⨾ A.C.Q
      □
```

9.3 Categoric.OCC.CSL.FromToAContextNatIsoPieces

```
open import RATH.Level
open import RATH.Data.Product using (Σ;•; proj₁; proj₂; _,_)

open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.MapCat using (MapCat)
open import Categoric.OSGC.PowerOp

open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OSGC.SyQ.WithResiduals
open import Categoric.OCC.SyQ

open import Categoric.Category
open import Categoric.Functor


module Categoric.OCC.CSL.FromToAContextNatIsoPieces {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
  (powerOp; ℙ; ∈; ∈-comprehensive
  ; ∈⨾-∈╳; ∈\-\-∈\; ╳∈-⨾∈˘; ∈⨾-∈\; /∈˘-⨾∈˘; \∈⨾∈\-reflexive; ╳∈-isMapping
```

```
  ; Ω; Ω-trans; Ω-refl; Ω⨾̆Ω; Ω˘-trans; Ω\Ω≈Ω
  ; Λ; Λ₀; Λ-isMapping
  )
open PowerOp                                        osgc powerOp
  using (Λ⨾∈˘)
open import Categoric.OSGC.PowerOrder        osgc leftResOp rightResOp powerOp
  using (Lub₀; Glb₀; Lub-cong; Glb-cong)
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 (↑⨾∈˘; ↑⨾∈\; ↓⨾∈˘; ↓⨾∈\; ↓≈⨾̆; ↑↓≈⨾̆; ↓↑≈⨾̆; Ω-⨾-↑↓˘)
open import Data.AContext.InOCC     occ leftResOp rightResOp powerOp
open import Data.AContext.Category occ leftResOp rightResOp powerOp
open import Categoric.OCC.CSL.FromAContext occ leftResOp rightResOp syqOp directPower splitSymIdempot
open import Categoric.OCC.CSL.ToAContext     occ leftResOp rightResOp syqOp directPower

open Category₁ (MapCat occ)
private
  module To = Functor CSL→Ctx
  module From = Functor Ctx→CSL
  module 𝒞₂ = Category ACH-Category
open Category₂ ACH-Category

module ACSL-FromToAContext {A : AContext} where
  module A = ToACSL A
  module A₁ = ACSL′ (toACSL A)
  R : Mor A.↑↓-image A.att
  R = A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘
  S : Mor A.ent A.↑↓-image
  S = ∈ ⨾ A.Q
  R-simpl : R ≈ S \ A.inc
  R-simpl = ≈-begin
      A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘
    ≈⟨ ⨾-cong₂ ↑⨾∈˘ ⟩
      A.Q˘ ⨾ (∈ \ A.inc)
    ≈⟨ \-inner-⨾ A.Q˘-isMapping ⟨≈≈⟩ \-cong₁ (⨾-cong₂ ˘˘) ⟩
      (∈ ⨾ A.Q) \ A.inc
    □
  S\-simpl : {Y : Obj} {T : Mor _ Y} → S \ T ≈ A.Q˘ ⨾ ((A.inc / (∈ \ A.inc)) \ T)
  S\-simpl {Y} {T} = ≈-begin
      (∈ ⨾ A.Q) \ T
    ≈⟨ \-cong₁ (⨾-cong₂ A.inc↑↓˘-⨾-Q ⟨≈˘≈⟩ ⨾-assocL) ⟨≈≈˘⟩ \-inner-⨾-˘M A.Q˘-isMapping ⟩
      A.Q˘ ⨾ ((∈ ⨾ A.inc↑↓₀ ˘) \ T)
    ≈⟨ ⨾-cong₂ (\-cong₁ ∈⨾↑↓˘) ⟩
      A.Q˘ ⨾ ((A.inc / (∈ \ A.inc)) \ T)
    □

C : AContextHom (To.obj (From.obj A)) A
C = record
  {mor = R
  ;srcCompat = ≈-begin
      R ↓₀ ⨾ A.≼ ↑↓₀
```



```
    ≈⟨ ⨾-cong ↓≈⨾̆ ↑↓≈⨾̆ ⟩
      ((R / ∈ ˘) ⨾̆ ∈) ⨾ ((A.≼ / (∈ \ A.≼)) ⨾̆ ∈)
    ≈⟨ ⨾̆-in-left ⨾̆∈-isMapping ⟨≈≈⟩ ⨾̆-cong₁ (≈-begin
        (A.≼ / (∈ \ A.≼)) ⨾ ((R / ∈ ˘) ⨾̆ ∈) ˘
      ≈⟨ /-inner-⨾ ⨾̆∈-isMapping ⟩
        A.≼ / (((R / ∈ ˘) ⨾̆ ∈) ⨾ (∈ \ A.≼))
      ≈⟨ /-cong₂ (\-inner-⨾ ⨾̆∈-isMapping ⟨≈≈⟩ \-cong₁ (⨾-cong₂ ⨾̆-˘ ⟨≈≈⟩ ∈⨾-∈⨾̆)) ⟩
        A.≼ / ((R / ∈ ˘) \ A.≼)
      ≈⟨ /-cong₂ (\-cong₁ (/-cong₁ R-simpl)) ⟩
        A.≼ / (((S \ A.inc) / ∈ ˘) \ A.≼)
      ≈˘⟨ /-cong₂ (\-cong₁ \/-≈) ⟩
        A.≼ / ((S \ (A.inc / ∈ ˘)) \ A.≼)
      ≈⟨ /-cong A.≼≈∈⨾Q\∈⨾Q (\-cong₂ A.≼≈∈⨾Q\∈⨾Q) ⟩
        (S \ S) / ((S \ (A.inc / ∈ ˘)) \ (S \ S))
      ≈⟨ ⊑-antisym (⊑-begin
          (S \ S) / ((S \ (A.inc / ∈ ˘)) \ (S \ S))
        ≈⟨ /-cong₂ \\ ⟩
          (S \ S) / ((S ⨾ (S \ (A.inc / ∈ ˘))) \ S)
        ⊑⟨ /-antitone (\-antitone \-cancel-outer) ⟩
          (S \ S) / ((A.inc / ∈ ˘) \ S)
        ≈˘⟨ \/-≈ ⟩
          S \ (S / ((A.inc / ∈ ˘) \ S))
        ⊑⟨ \-universal (⊑-begin
            S ⨾ (S \ (S / ((A.inc / ∈ ˘) \ S)))
          ⊑⟨ \-cancel-outer ⟩
            S / ((A.inc / ∈ ˘) \ S)
          ⊑⟨ /-universal (⊑-begin
              (S / ((A.inc / ∈ ˘) \ S)) ⨾ ∈ ˘
            ⊑⟨ /-below-S/∘\S-cancel-outer ⟩
              S / (A.inc \ S)
            ≈⟨⟩
              (∈ ⨾ A.Q) / (A.inc \ (∈ ⨾ A.Q))
            ≈⟨ A.⨾Q-/-\⨾Q ⟩
              (∈ ⨾ A.inc↑↓₀ ˘) / (A.inc \ (∈ ⨾ A.inc↑↓₀ ˘))
            ≈⟨ /-cong ∈⨾↑↓˘ (\-cong₂ ∈⨾↑↓˘ ⟨≈≈⟩ \/-≈) ⟩
              (A.inc / (∈ \ A.inc)) / ((A.inc \ A.inc) / (∈ \ A.inc))
            ⊑⟨ /-antitone \-twist ⟩
              (A.inc / (∈ \ A.inc)) / (A.inc \ ∈)
            ≈⟨ // ⟩
              A.inc / ((A.inc \ ∈) ⨾ (∈ \ A.inc))
            ⊑⟨ /-antitone \∈⨾∈\-reflexive ⟩
              A.inc / Id
            ≈⟨ /-Id ⟩
              A.inc
            □) ⟩
            A.inc / ∈ ˘
          □) ⟩
          S \ (A.inc / ∈ ˘)
        □) ⊑-S/∘\S ⟩
        S \ (A.inc / ∈ ˘)
      ≈⟨ \/-≈ ⟩
        (S \ A.inc) / ∈ ˘
      ≈˘⟨ /-cong₁ R-simpl ⟩
        R / ∈ ˘
      □) ⟩
      (R / ∈ ˘) ⨾̆ ∈
```

```
      ≈˘⟨ ↓≈⨉ ⟩
        R ↓₀
      □
   ; trgCompat  =  ≈-begin
        A.inc ↓↑₀ ⨾ R ↓₀
      ≈⟨ ⨾-cong ↓↑≈⨉ ↓≈⨉ ⟩
        ((A.inc ˘ / (∈ \ A.inc ˘)) ⨉ ∈) ⨾ ((R / ∈ ˘) ⨉ ∈)
      ≈⟨ ⨉-in-left ⨉∈-isMapping ⟨≈≈⟩ ⨉-cong₁ (≈-begin
          (R / ∈ ˘) ⨾ ((A.inc ˘ / (∈ \ A.inc ˘)) ⨉ ∈) ˘
        ≈⟨ /-inner-⨾ ⨉∈-isMapping ⟩
          R / (((A.inc ˘ / (∈ \ A.inc ˘)) ⨉ ∈) ⨾ ∈ ˘)
        ≈⟨ /-cong₂ (⨉-total-cancel-right ∈-comprehensive) ⟩
          R / ((A.inc ˘ / (∈ \ A.inc ˘)) ˘)
        ≈⟨ /-cong (⨾-cong₂ ↑⨾∈˘) (˘/-˘ ⟨≈≈⟩ \-cong₁ \˘-˘) ⟩
          (A.Q˘ ⨾ (∈ \ A.inc)) / ((A.inc / ∈ ˘) \ A.inc)
        ≈˘⟨ /-outer-⨾-≈ A.Q˘-isMapping ⟩
          A.Q˘ ⨾ ((∈ \ A.inc) / ((A.inc / ∈ ˘) \ A.inc))
        ≈⟨ ⨾-cong₂ (≈-begin
            (∈ \ A.inc) / ((A.inc / ∈ ˘) \ A.inc)
          ≈˘⟨ \/-≈ ⟩
            ∈ \ (A.inc / ((A.inc / ∈ ˘) \ A.inc))
          ≈⟨ \-cong₂ S/∘\S∘S/ ⟩
            ∈ \ (A.inc / ∈ ˘)
          ≈⟨ \/-≈ ⟩
            (∈ \ A.inc) / ∈ ˘
          □) ⟩
          A.Q˘ ⨾ ((∈ \ A.inc) / ∈ ˘)
        ≈⟨ /-outer-⨾-≈ A.Q˘-isMapping ⟩
          (A.Q˘ ⨾ (∈ \ A.inc)) / ∈ ˘
        ≈˘⟨ /-cong₁ (⨾-cong₂ ↑⨾∈˘) ⟩
          R / ∈ ˘
        □) ⟩
        (R / ∈ ˘) ⨉ ∈
      ≈˘⟨ ↓≈⨉ ⟩
        R ↓₀
      □
   }

C⁻¹ : AContextHom A (To.obj (From.obj A))
C⁻¹ = record
   {mor = S
   ; srcCompat  =  ≈-begin
        S ↓₀ ⨾ A.inc ↑↓₀
      ≈⟨ ⊑-antisym (⊑-begin
            S ↓₀ ⨾ A.inc ↑↓₀
          ⊑⟨ ⨾-monotone₁ (⊑-begin
              S ↓₀
            ⊑⟨ ⨉-universal (⊑-begin
                (S / ∈ ˘) ⨾ S ↓₀
              ⊑⟨ /-universal (⊑-begin
                  ((S / ∈ ˘) ⨾ S ↓₀) ⨾ (∈ \ A.inc)
                ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ↓⨾∈\ ⟩
                  (S / ∈ ˘) ⨾ ((S / ∈ ˘) \ A.inc)
                ⊑⟨ \-cancel-outer ⟩
                  A.inc
                □) ⟩
```



```
                  A.inc / (∈ \ A.inc)
                □) (⊑-begin
                  S ↓₀ ⨾ (A.inc / (∈ \ A.inc)) ˘
                ⊑⟨ \-universal (⊑-begin
                    ∈ ⨾ S ↓₀ ⨾ (A.inc / (∈ \ A.inc)) ˘
                  ≈⟨ ⨾-cong₂₂ /-˘ ⟩
                    ∈ ⨾ S ↓₀ ⨾ ((∈ \ A.inc) ˘ \ A.inc ˘)
                  ≈⟨ ⨾-cong₂ (\-inner-⨾ (Mapping.prf (S ↓))) ⟩
                    ∈ ⨾ (((∈ \ A.inc) ˘ ⨾ S ↓₀ ˘) \ A.inc ˘)
                  ≈⟨ ⨾-cong₂ (\-cong₁ (⨾-˘ ⟨≈˘≈⟩ ˘-cong ↓⨾∈\)) ⟩
                    ∈ ⨾ (((S / ∈ ˘) \ A.inc) ˘ \ A.inc ˘)
                  ≈⟨ ⨾-cong₂ /-˘ ⟨≈˘≈⟩ ⨾˘-˘ ⟩
                    ((A.inc / ((S / ∈ ˘) \ A.inc)) ⨾ ∈ ˘) ˘
                  ⊑⟨ ˘-monotone /-below-S/∘\S-cancel-outer ⟩
                    (A.inc / (S \ A.inc)) ˘
                  ≈⟨⟩
                    (A.inc / ((∈ ⨾ A.Q) \ A.inc)) ˘
                  ≈⟨ ˘-cong A.inc/-∈⨾Q\inc ⟩
                    (∈ ⨾ A.Q) ˘
                  ≈⟨⟩
                    S ˘
                  □) ⟩
                  ∈ \ S ˘
                ≈˘⟨ /˘-˘ ⟩
                  (S / ∈ ˘) ˘
                □) ⟩
              (S / ∈ ˘) ⨉ (A.inc / (∈ \ A.inc))
            ≈˘⟨ ⨉-cong₁ (⨾˘-˘ ⟨≈˘≈⟩ ˘-cong Λ⨾∈˘ ⟨≈≈⟩ \˘-˘) ⟩
              (∈ ⨾ S ↓₀ ˘) ⨉ (A.inc / (∈ \ A.inc))
            ≈˘⟨ ⨉-in-left (Mapping.prf (S ↓)) ⟩
              S ↓₀ ⨾ (∈ ⨉ (A.inc / (∈ \ A.inc)))
            ≈˘⟨ ⨾-cong₂ (˘-cong ↑↓≈⨉ ⟨≈≈⟩ ⨉-˘) ⟩
              S ↓₀ ⨾ A.inc↑↓₀ ˘
            □) ⟩
            (S ↓₀ ⨾ A.inc↑↓₀ ˘) ⨾ A.inc↑↓₀
          ⊑⟨ ⨾-assoc ⟨≈⊑⟩ proj₂ (Mapping.unival A.inc↑↓) ⟩
            S ↓₀
          □) (⊑-begin
            S ↓₀
          ⊑⟨ ⨉-universal (⊑-begin
              ((A.inc / (∈ \ A.inc)) ⨾ S ↓₀ ˘) ⨾ S ↓₀
            ≈⟨ ⨾-cong₁ (/-inner-⨾ (Mapping.prf (S ↓))) ⟩
              (A.inc / (S ↓₀ ⨾ (∈ \ A.inc))) ⨾ S ↓₀
            ≈⟨ ⨾-cong₁ (/-cong₂ ↓⨾∈\) ⟩
              (A.inc / ((S / ∈ ˘) \ A.inc)) ⨾ S ↓₀
            ⊑⟨ ⨾-monotone₁ S/∘\S-into-/-⊑ ⟩
              ((A.inc / (S \ A.inc)) / ∈ ˘) ⨾ S ↓₀
            ≈⟨ ⨾-cong₂ ↓≈⨉ ⟩
              ((A.inc / (S \ A.inc)) / ∈ ˘) ⨾ ((S / ∈ ˘) ⨉ ∈)
            ⊑⟨ ⨾-cong₂ (⨉-cong₁ (/-cong₁ A.inc/-∈⨾Q\inc)) ⟨≈˘⊑⟩ ⨉-cancel-left ⟩
              ∈
            □) (⊑-begin
              S ↓₀ ⨾ ∈ ˘
            ≈⟨ ↓⨾∈˘ ⟩
              ∈ \ S ˘
            ≈˘⟨ /˘-˘ ⟩
```

```
          (S / ∈ ˘) ˘
        ⊑⟨ ˘-monotone (⊑-begin
            S / ∈ ˘
          ⊑⟨ ⊑-S/○\S ⟩
            A.inc / ((S / ∈ ˘) \ A.inc)
          ≈˘⟨ /-cong₂ ↓⨾∈\ ⟩
            A.inc / (S ↓₀ ⨾ (∈ \ A.inc))
          ≈˘⟨ /-inner-⨾ (Mapping.prf (S ↓)) ⟩
            (A.inc / (∈ \ A.inc)) ⨾ S ↓₀ ˘
          □) ⟩
          ((A.inc / (∈ \ A.inc)) ⨾ S ↓₀ ˘) ˘
        □) ⟩
        ((A.inc / (∈ \ A.inc)) ⨾ S ↓₀ ˘) ⍀ ∈
      ≈˘⟨ ⍀-in-left (Mapping.prf (S ↓)) ⟩
        S ↓₀ ⨾ ((A.inc / (∈ \ A.inc)) ⍀ ∈)
      ≈˘⟨ ⨾-cong₂ ↑↓≈⍀ ⟩
        S ↓₀ ⨾ A.inc ↑↓₀
      □) ⟩
    S ↓₀
  □
; trgCompat = ≈-begin
    A.≼ ↓↑₀ ⨾ S ↓₀
  ≈⟨ ⨾-cong ↓↑≈⍀ ↓≈⍀ ⟩
    ((A.≼ ˘ / (∈ \ A.≼ ˘)) ⍀ ∈) ⨾ ((S / ∈ ˘) ⍀ ∈)
  ≈⟨ ⍀-in-left ⍀∈-isMapping ⟨≈≈⟩ ⍀-cong₁ (≈-begin
      ((∈ ⨾ A.Q) / ∈ ˘) ⨾ ((A.≼ ˘ / (∈ \ A.≼ ˘)) ⍀ ∈) ˘
    ≈⟨ /-inner-⨾ ⍀∈-isMapping ⟩
      (∈ ⨾ A.Q) / (((A.≼ ˘ / (∈ \ A.≼ ˘)) ⍀ ∈) ⨾ ∈ ˘)
    ≈⟨ /-cong₂ (⍀∈-⨾∈˘ ⟨≈≈⟩ ˘/-˘ ⟨≈≈⟩ \-cong₁ \˘-˘) ⟩
      (∈ ⨾ A.Q) / ((A.≼ / ∈ ˘) \ A.≼)
    ≈⟨ ⊑-antisym (/-antitone ⊑-\S○S/) (⊑-begin
          (∈ ⨾ A.Q) / ∈ ˘
        ⊑⟨ /-universal (⊑-begin
              ((∈ ⨾ A.Q) / ∈ ˘) ⨾ (((A.Q˘ ⨾ Ω ⨾ A.Q) / ∈ ˘) \ (A.Q˘ ⨾ Ω ⨾ A.Q))
            ≈⟨ ⨾-cong₂ A.Q˘⨾/-\-Q˘⨾ ⟩
              ((∈ ⨾ A.Q) / ∈ ˘) ⨾ (((A.inc↑↓₀ ⨾ Ω ⨾ A.Q) / ∈ ˘) \ (A.inc↑↓₀ ⨾ Ω ⨾ A.Q))
            ≈⟨ ⨾-cong₂ (\-cong (/-cong₁ A.inc↑↓-⨾-Ω-⨾-Q) A.inc↑↓-⨾-Ω-⨾-Q) ⟩
              ((∈ ⨾ A.Q) / ∈ ˘) ⨾ (((Ω ⨾ A.Q) / ∈ ˘) \ (Ω ⨾ A.Q))
            ≈⟨ ⨾-cong₂ (\-cong (/-cong₁ A.Ω⨾Q ⟨≈≈˘⟩ \/-≈) A.Ω⨾Q) ⟩
              ((∈ ⨾ A.Q) / ∈ ˘) ⨾ ((∈ \ ((∈ ⨾ A.Q) / ∈ ˘)) \ (∈ \ (∈ ⨾ A.Q)))
            ≈⟨ ⨾-cong₂ ∈\-\-∈\ ⟩
              ((∈ ⨾ A.Q) / ∈ ˘) ⨾ (((∈ ⨾ A.Q) / ∈ ˘) \ (∈ ⨾ A.Q))
            ⊑⟨ \-cancel-outer ⟩
              ∈ ⨾ A.Q
            □) ⟩
          (∈ ⨾ A.Q) / ((A.≼ / ∈ ˘) \ A.≼)
        □) ⟩
      (∈ ⨾ A.Q) / ∈ ˘
    □) ⟩
    (S / ∈ ˘) ⍀ ∈
  ≈˘⟨ ↓≈⍀ ⟩
    S ↓₀
  □
}
```



```
C-⨾-C⁻¹ : C ⨾₂ C⁻¹ ≈₂ Id₂ {To.obj (From.obj A)}
C-⨾-C⁻¹ = ≈-begin
    [ (∈ ⨾ A.Q) ↓ ⨾₁ A.inc ↑ ⨾₁ (A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘) ↓ ]
  ≈⟨ ˘-cong (⨾-cong₂ ⨾-assoc₃₊₁) ⟩
    (Λ₀ Id ⨾ (∈ ⨾ A.Q) ↓₀ ⨾ A.inc ↑₀ ⨾ (A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘) ↓₀ ⨾ ∈ ˘) ˘
  ≈⟨ ˘-cong (⨾-cong₂₂₂ ↓⨾∈˘) ⟩
    (Λ₀ Id ⨾ (∈ ⨾ A.Q) ↓₀ ⨾ A.inc ↑₀ ⨾ (∈ \ (A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘) ˘)) ˘
  ≈⟨ ˘-cong (⨾-cong₂₂ ↑⨾∈\) ⟩
    (Λ₀ Id ⨾ (∈ ⨾ A.Q) ↓₀ ⨾ ((∈ \ A.inc) ˘ \ (A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘) ˘)) ˘
  ≈⟨ ˘-cong (⨾-cong₂₂ (\-cong₂ (⨾-˘ ⟨≈≈⟩ ⨾-cong₁ (˘-cong ↑⨾∈˘)))) ⟩
    (Λ₀ Id ⨾ (∈ ⨾ A.Q) ↓₀ ⨾ ((∈ \ A.inc) ˘ \ ((∈ \ A.inc) ˘ ⨾ A.Q˘ ˘))) ˘
  ≈˘⟨ ˘-cong (⨾-cong₂₂ (\-outer-⨾-≈ A.Q˘-isMapping)) ⟩
    (Λ₀ Id ⨾ (∈ ⨾ A.Q) ↓₀ ⨾ ((∈ \ A.inc) ˘ \ (∈ \ A.inc) ˘) ⨾ A.Q˘ ˘) ˘
  ≈⟨ ˘-cong (⨾-assocL ⟨≈≈⟩ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ (\-inner-⨾ (Mapping.prf (Λ Id ⨾₁ (∈ ⨾ A.Q) ↓))))) ⟩
    ((((∈ \ A.inc) ˘ ⨾ (Λ₀ Id ⨾ (∈ ⨾ A.Q) ↓₀) ˘) \ (∈ \ A.inc) ˘) ⨾ A.Q˘ ˘) ˘
  ≈⟨ ⨾˘-˘ ⟨≈≈⟩ ⨾-cong₂ (\˘-˘ ⟨≈≈⟩ /-cong₂ ˘⨾˘-˘) ⟩
    A.Q˘ ⨾ ((∈ \ A.inc) / ((Λ₀ Id ⨾ (∈ ⨾ A.Q) ↓₀) ⨾ (∈ \ A.inc)))
  ≈⟨ ⨾-cong₂ (/-cong₂ (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ↓⨾∈\)) ⟩
    A.Q˘ ⨾ ((∈ \ A.inc) / (Λ₀ Id ⨾ (((∈ ⨾ A.Q) / ∈ ˘) \ A.inc)))
  ≈⟨ ⨾-cong₂ (/-cong₂ (\-inner-⨾ Λ-isMapping)) ⟩
    A.Q˘ ⨾ ((∈ \ A.inc) / ((((∈ ⨾ A.Q) / ∈ ˘) ⨾ Λ₀ Id ˘) \ A.inc))
  ≈⟨ ⨾-cong₂ (/-cong₂ (\-cong₁ (/-inner-⨾ Λ-isMapping))) ⟩
    A.Q˘ ⨾ ((∈ \ A.inc) / (((∈ ⨾ A.Q) / (Λ₀ Id ⨾ ∈ ˘)) \ A.inc))
  ≈⟨ ⨾-cong₂ (/-cong₂ (\-cong₁ (/-cong₂ Λ⨾∈˘))) ⟩
    A.Q˘ ⨾ ((∈ \ A.inc) / (((∈ ⨾ A.Q) / Id) \ A.inc))
  ≈⟨ ⨾-cong₂ (/-cong₂ (\-cong₁ /-Id)) ⟩
    A.Q˘ ⨾ ((∈ \ A.inc) / ((∈ ⨾ A.Q) \ A.inc))
  ≈⟨ /-outer-⨾-≈ A.Q˘-isMapping ⟩
    (A.Q˘ ⨾ (∈ \ A.inc)) / ((∈ ⨾ A.Q) \ A.inc)
  ≈⟨ /-cong₁ (\-inner-⨾ A.Q˘-isMapping) ⟩
    ((∈ ⨾ A.Q˘ ˘) \ A.inc) / ((∈ ⨾ A.Q) \ A.inc)    -- to old calc
  ≈⟨ /-cong₁ (\-cong₁ (⨾-cong₂ ˘˘)) ⟩
    (S \ A.inc) / (S \ A.inc)
  ≈⟨ ⊑-antisym (\-universal (⊑-begin
        S ⨾ ((S \ A.inc) / (S \ A.inc))
      ≈˘⟨ ⨾-cong₂ \/-≈ ⟩
        S ⨾ (S \ (A.inc / (S \ A.inc)))
      ⊑⟨ \-cancel-outer ⟩
        A.inc / (S \ A.inc)
      ≈⟨ /-cong₂ (\-inner-⨾-˘M A.Q˘-isMapping) ⟨≈˘≈˘⟩ /-inner-⨾ A.Q˘-isMapping ⟩
        (A.inc / (∈ \ A.inc)) ⨾ A.Q˘ ˘
      ≈⟨ ⨾-cong (≈-sym ∈⨾↑↓˘) ˘˘ ⟨≈≈⟩ ⨾-assoc ⟩
        ∈ ⨾ A.inc↑↓₀ ˘ ⨾ A.Q
      ≈⟨ ⨾-cong₂ A.inc↑↓˘-⨾-Q ⟩
        S
      □)) \-twist ⟩
    S \ S
  ≈˘⟨ A.≼≈∈⨾Q\∈⨾Q ⟩
    A.≼
  □


C⁻¹-⨾-C : C⁻¹ ⨾₂ C ≈₂ Id₂ {A}
C⁻¹-⨾-C = ≈-begin
    [ (A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘) ↓ ⨾₁ A.≼ ↑ ⨾₁ (∈ ⨾ A.Q) ↓ ]
  ≈⟨ ˘-cong (⨾-cong₂ ⨾-assoc₃₊₁) ⟩
    (Λ₀ Id ⨾ (A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘) ↓₀ ⨾ A.≼ ↑₀ ⨾ (∈ ⨾ A.Q) ↓₀ ⨾ ∈ ˘) ˘
```

```
≈⟨ ˘-cong (⨾-cong₂₂₂ ↓⨾∈˘) ⟩
  (Λ₀ Id ⨾ (A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘) ↓₀ ⨾ A.≼ ↑₀ ⨾ (∈ \ (∈ ⨾ A.Q) ˘)) ˘
≈⟨ ˘-cong (⨾-assocL ⟨≈≈⟩ ⨾-cong₂ (↑⨾∈\ ⟨≈≈˘⟩ /-˘)) ⟩
  ((Λ₀ Id ⨾ (A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘) ↓₀) ⨾ ((∈ ⨾ A.Q) / (∈ \ A.≼)) ˘) ˘
≈⟨ ⨾˘-˘ ⟨≈≈⟩ /-inner-⨾ (Mapping.prf (Λ Id ⨾₁ (A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘) ↓)) ⟩
  (∈ ⨾ A.Q) / ((Λ₀ Id ⨾ (A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘) ↓₀) ⨾ (∈ \ A.≼))
≈⟨ /-cong₂ (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ↓⨾∈\) ⟩
  (∈ ⨾ A.Q) / (Λ₀ Id ⨾ (((A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘) / ∈ ˘) \ A.≼))
≈⟨ /-cong₂ (\-inner-⨾ Λ-isMapping ⟨≈≈⟩ \-cong₁ (/-inner-⨾ Λ-isMapping ⟨≈≈⟩ /-cong₂ Λ⨾∈˘ ⟨≈≈⟩ /-Id)) ⟩
  (∈ ⨾ A.Q) / ((A.Q˘ ⨾ A.inc ↑₀ ⨾ ∈ ˘) \ A.≼)
≈⟨ /-cong₂ (\-cong₁ (⨾-cong₂ ↑⨾∈˘)) ⟩
  (∈ ⨾ A.Q) / ((A.Q˘ ⨾ (∈ \ A.inc)) \ (A.Q˘ ⨾ Ω ⨾ A.Q))
≈⟨ /-cong₂ (\-cong₂ (⨾-assocL ⟨≈≈˘⟩ ⨾-cong₂ ˘˘) ⟨≈≈˘⟩ \-outer-⨾-≈ A.Q˘-isMapping) ⟩
  (∈ ⨾ A.Q) / (((A.Q˘ ⨾ (∈ \ A.inc)) \ (A.Q˘ ⨾ Ω)) ⨾ A.Q˘ ˘)
≈⟨ /-flip-˘ A.Q˘-isMapping ⟨≈˘≈⟩ /-cong₂ (⨾-assoc ⟨≈≈⟩ ⨾-cong₂₁ ˘˘) ⟩
  ∈ / (((A.Q˘ ⨾ (∈ \ A.inc)) \ (A.Q˘ ⨾ Ω)) ⨾ A.Q ⨾ A.Q˘)
≈⟨ /-cong₂ (⨾-cong₁ (\-flip-M A.Q˘-isMapping ⟨≈˘≈⟩ \-cong₁ (⨾-cong₁ ˘˘ ⟨≈≈⟩ ⨾-assocL))) ⟩
  ∈ / ((((A.Q ⨾ A.Q˘) ⨾ (∈ \ A.inc)) \ Ω) ⨾ A.Q ⨾ A.Q˘)
≈⟨ /-cong₂ (⨾-cong (\-cong₁ (⨾-cong₁ A.factors ⟨≈≈⟩ ⨾-assoc)) A.factors ⟨≈≈⟩ ⨾-assocL) ⟩
  ∈ / ((((A.inc↑↓₀ ˘ ⨾ A.inc↑↓₀ ⨾ (∈ \ A.inc)) \ Ω) ⨾ A.inc↑↓₀ ˘) ⨾ A.inc↑↓₀)
≈⟨ /-cong₂ (⨾-cong₁ (\-outer-⨾-≈ (Mapping.prf A.inc↑↓))) ⟩
  ∈ / (((A.inc↑↓₀ ˘ ⨾ A.inc↑↓₀ ⨾ (∈ \ A.inc)) \ (Ω ⨾ A.inc↑↓₀ ˘)) ⨾ A.inc↑↓₀)
≈⟨ /-cong₂ (⨾-cong₁ (\-flip-M (Mapping.prf A.inc↑↓))) ⟩
  ∈ / (((A.inc↑↓₀ ⨾ (∈ \ A.inc)) \ (A.inc↑↓₀ ⨾ Ω ⨾ A.inc↑↓₀ ˘)) ⨾ A.inc↑↓₀)
≈⟨ /-flip (Mapping.prf A.inc↑↓) ⟨≈≈⟩ /-cong₁ (⨾˘-˘ ⟨≈˘≈⟩ ˘-cong ↑↓⨾∈˘ ⟨≈≈⟩ ˘˘) ⟩
  (A.inc / (∈ \ A.inc)) / ((A.inc↑↓₀ ⨾ (∈ \ A.inc)) \ (A.inc↑↓₀ ⨾ Ω ⨾ A.inc↑↓₀ ˘))
≈⟨ /-cong₂ (\-cong₁ A.inc↑↓-⨾-∈\inc) ⟩
  (A.inc / (∈ \ A.inc)) / ((∈ \ A.inc) \ (A.inc↑↓₀ ⨾ Ω ⨾ A.inc↑↓₀ ˘))
≈⟨ /-cong₂ (\-cong₂ (⨾-cong₂ Ω-⨾-↑↓˘)) ⟩
  (A.inc / (∈ \ A.inc)) / ((∈ \ A.inc) \ (A.inc↑↓₀ ⨾ ((∈ \ A.inc) / (∈ \ A.inc))))
≈⟨ /-cong₂ (\-cong₂ (/-outer-⨾-≈ (Mapping.prf A.inc↑↓) ⟨≈≈⟩ /-cong₁ A.inc↑↓-⨾-∈\inc)) ⟩
  (A.inc / (∈ \ A.inc)) / ((∈ \ A.inc) \ ((∈ \ A.inc) / (∈ \ A.inc)))
≈⟨ /-cong₂ \/-≈ ⟩
  (A.inc / (∈ \ A.inc)) / (((∈ \ A.inc) \ (∈ \ A.inc)) / (∈ \ A.inc))
≈⟨ ⊑-antisym (⊑-begin
    (A.inc / (∈ \ A.inc)) / (((∈ \ A.inc) \ (∈ \ A.inc)) / (∈ \ A.inc))
  ≈⟨ /-cong₂ (/-cong₁ (\\ ⟨≈≈⟩ \-cong₁ ∈⨾-∈\)) ⟩
    (A.inc / (∈ \ A.inc)) / ((A.inc \ A.inc) / (∈ \ A.inc))
  ⊑⟨ /-antitone \-twist-down ⟩
    (A.inc / (∈ \ A.inc)) / (A.inc \ ∈)
  ≈⟨ // ⟩
    A.inc / ((A.inc \ ∈) ⨾ (∈ \ A.inc))
  ⊑⟨ /-antitone \∈⨾∈\-reflexive ⟩
    A.inc / Id
  ≈⟨ /-Id ⟩
    A.inc
  □) (⊑-begin
    A.inc
  ⊑⟨ ⊑-S/∘\S ⟩
    A.inc / (A.inc \ A.inc)
  ≈˘⟨ /-cong₂ (\-cong₁ ∈⨾-∈\) ⟩
    A.inc / ((∈ ⨾ (∈ \ A.inc)) \ A.inc)
  ≈˘⟨ /-cong₂ \\ ⟩
    A.inc / ((∈ \ A.inc) \ (∈ \ A.inc))
  ⊑⟨ /-antitone /-cancel-outer ⟩
    A.inc / ((((∈ \ A.inc) \ (∈ \ A.inc)) / (∈ \ A.inc)) ⨾ (∈ \ A.inc))
```



```
  ≈˘⟨ // ⟩
    (A.inc / (∈ \ A.inc)) / (((∈ \ A.inc) \ (∈ \ A.inc)) / (∈ \ A.inc))
  □) ⟩
  A.inc
□
```

open ACSL-FromToAContext **public using** (**C**; **C**⁻¹; **C**-⨾-**C**⁻¹; **C**⁻¹-⨾-**C**)

9.4 Categoric.OCC.CSL.FromToAContextNatIsoNaturality

```
open import RATH.Level
open import RATH.Data.Product using (Σ:•; proj₁; proj₂; _,_)
open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.MapCat using (MapCat)
open import Categoric.OSGC.PowerOp
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OrderedCategory.Residuals
open import Categoric.OSGC.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.OSGC.SyQ.WithResiduals
open import Categoric.OCC.SyQ
open import Categoric.Category
open import Categoric.Functor
```

```
module Categoric.OCC.CSL.FromToAContextNatIsoNaturality {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
  (powerOp; ℙ; ∈
  ; ∈⨾-∈╳; ∈\-\-∈\; ╳∈-⨾∈˘; ∈⨾-∈\; /∈˘-⨾∈˘
  ; ∈˘\-/∈˘\; /∈˘-/-/∈˘
  ; Ω; Ω-trans; Ω-refl; Ω╳Ω; Ω˘-trans; Ω\Ω≈Ω
  ; Λ; Λ₀; Λ-isMapping; Λ-˘-⨾-∈\
  )
open PowerOp                                     osgc powerOp
  using (Λ⨾∈˘)
open import Categoric.OSGC.PowerOrder            osgc leftResOp rightResOp powerOp
  using (Lub₀; Glb₀; Lub-cong; Glb-cong)
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 (↓⨾∊˘; ↑⨾∊˘; ↑⨾∊\)
open import Data.AContext.InOCC    occ leftResOp rightResOp powerOp
open import Data.AContext.Category occ leftResOp rightResOp powerOp
open import Categoric.OCC.CSL.FromAContext occ leftResOp rightResOp syqOp directPower splitSymIdempot
open import Categoric.OCC.CSL.ToAContext     occ leftResOp rightResOp syqOp directPower
open import Categoric.OCC.CSL.FromToAContextNatIsoPieces
                                             occ leftResOp rightResOp syqOp directPower splitSymIdempot

open Category₁ (MapCat occ)
private
  module To = Functor CSL→Ctx
  module From = Functor Ctx→CSL
  module 𝒞₂ = Category ACH-Category
open Category₂ ACH-Category

C-naturality : {A B : AContext} → {F : AContextHom A B}
          → (To.mor (From.mor F) ⨾₂ C {B}) ≈₂ (C {A} ⨾₂ F)
C-naturality {A} {B} {F} = ˘-cong (≈-begin
    Λ₀ Id ⨾ (B.R ↓₀ ⨾ B.≼ ↑₀ ⨾ (A.≼ ⨾ F₁.map₀ ˘) ↓₀) ⨾ ∊ ˘
  ≈⟨ ⨾-cong₂ (⨾-assoc₃₊₁ ⟨≈≈⟩ ⨾-cong₂₂ ↓⨾∊˘) ⟩
    Λ₀ Id ⨾ B.R ↓₀ ⨾ B.≼ ↑₀ ⨾ (∊ \ (A.≼ ⨾ F₁.map₀ ˘) ˘)
  ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong ΛId-⨾-↓ ↑⨾∊\ ⟩
    Λ₀ (B.R ˘) ⨾ ((∊ \ B.≼) ˘ \ (A.≼ ⨾ F₁.map₀ ˘) ˘)
  ≈⟨ \-inner-⨾ Λ-isMapping ⟨≈≈⟩ \-cong₁ (⨾-˘ ⟨≈˘≈⟩ ˘-cong Λ-˘-⨾-∊\) ⟩
    (B.R \ B.≼) ˘ \ (A.≼ ⨾ F₁.map₀ ˘) ˘
  ≈⟨ /-˘ ⟨≈˘≈⟩ ˘-cong (≈-begin
        (A.≼ ⨾ F₁.map₀ ˘) / (B.R \ B.≼)
      ≈˘⟨ /-flip (Mapping.prf F₁.map) ⟩
        A.≼ / ((B.R \ B.≼) ⨾ F₁.map₀)
      ≈⟨ /-cong₂ (⨾-cong₁ (B.Q˘⨾-\-Q˘⨾ ⟨≈≈⟩ \-cong (⨾-assocL ⟨≈≈⟩ ⨾-cong₁ ↑↓⨾↑ ⟨≈≈⟩ ↑⨾∊˘) (B.inc↑↓-⨾-Ω-⨾-Q ⟨≈≈⟩ B.Ω⨾Q))) ⟩
        A.≼ / (((∊ \ B.inc) \ (∊ \ (∊ ⨾ B.Q))) ⨾ F₁.map₀)
      ≈⟨ /-cong₂ (⨾-cong₁ (∊\-\-∊\ ⟨≈≈˘⟩ \-outer-⨾-≈-˘M B.Q˘-isMapping) ⟨≈≈⟩ ⨾-assoc) ⟩
        A.≼ / ((B.inc \ ∊) ⨾ B.Q ⨾ F₁.map₀)
      ≈⟨⟩
        A.≼ / ((B.inc \ ∊) ⨾ B.Q ⨾ B.Q˘ ⨾ B.inc ↑₀ ⨾ F.mor ↓₀ ⨾ A.Q)
      ≈⟨ /-cong₂ (⨾-cong₂ (⨾-cong₁&₂₁ (B.factors ⟨≈≈˘⟩ ran↓-≈-ran↑↓)) ⟨≈≈⟩ ⨾-assocL) ⟩
        A.≼ / (((B.inc \ ∊) ⨾ B.inc ↓₀ ˘) ⨾ B.inc ↓₀ ⨾ B.inc ↑₀ ⨾ F.mor ↓₀ ⨾ A.Q)
      ≈⟨ /-cong₂ (⨾-cong (\-outer-⨾-≈ (Mapping.prf (B.inc ↓)) ⟨≈≈⟩ \-cong₂ (⨾˘-˘ ⟨≈˘≈⟩ ˘-cong ↓⨾∊˘ ⟨≈≈⟩ \˘-˘))
                           (⨾-assocL ⟨≈≈⟩ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ F.trgCompat)) ⟩
        A.≼ / ((B.inc \ (B.inc / ∊ ˘)) ⨾ F.mor ↓₀ ⨾ A.Q)
      ≈⟨ /-cong (⨾-assocL ⟨≈≈⟩ ⨾-cong₁ A.Q˘⨾Ω) ⨾-assocL ⟩
        ((A.S \ ∊) ⨾ A.Q) / (((B.inc \ (B.inc / ∊ ˘)) ⨾ F.mor ↓₀) ⨾ A.Q)
      ≈⟨ /-flip-˘ A.Q˘-isMapping ⟨≈˘≈⟩ /-cong₂ (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ A.factors ⟨≈≈⟩ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ F.srcCompat′) ⟩
        (A.S \ ∊) / ((B.inc \ (B.inc / ∊ ˘)) ⨾ F.mor ↓₀)
      ≈⟨ /-flip (Mapping.prf (F.mor ↓)) ⟨≈≈⟩ /-cong₁ (\-outer-⨾-≈ (Mapping.prf (F.mor ↓))) ⟩
        (A.S \ (∊ ⨾ F.mor ↓₀ ˘)) / (B.inc \ (B.inc / ∊ ˘))
      ≈⟨ /-cong₁ (\-cong₂ (⨾˘-˘ ⟨≈˘≈⟩ ˘-cong ↓⨾∊˘ ⟨≈≈⟩ \˘-˘)) ⟩
        (A.S \ (F.mor / ∊ ˘)) / (B.inc \ (B.inc / ∊ ˘))
      ≈⟨ /-cong \/-≈ \/-≈ ⟩
        ((A.S \ F.mor) / ∊ ˘) / ((B.inc \ B.inc) / ∊ ˘)
      ≈⟨ /∊˘-/-/∊˘ ⟩
        (A.S \ F.mor) / (B.inc \ B.inc)
      ≈˘⟨ \/-≈ ⟩
        A.S \ (F.mor / (B.inc \ B.inc))
      ≈⟨ \-cong₂ F.trgCompat-\′ ⟩
```



```
        A.S \ F.mor
      ≈⟨ \-cong₂ (⊑-antisym ⊑-S/∘\S (F.srcCompat-/\ ⟨⊑≈⟩ S/∘\S-S)) ⟩
        A.S \ (A.inc / (F.mor \ A.inc))
      ≈⟨ \/-≈ ⟩
        (A.S \ A.inc) / (F.mor \ A.inc)
      ≈˘⟨ /-cong₁ A.R-simpl ⟩
        A.R / (F.mor \ A.inc)
      □) ⟨≈≈⟩ /-˘ ⟩
    (F.mor \ A.inc) ˘ \ A.R ˘
  ≈˘⟨ \-inner-⨾ Λ-isMapping ⟨≈≈⟩ \-cong₁ (⨾-˘ ⟨≈˘≈⟩ ˘-cong Λ-˘-⨾-∊\) ⟩
    Λ₀ (F.mor ˘) ⨾ ((∊ \ A.inc) ˘ \ A.R ˘)
  ≈˘⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong ΛId-⨾-↓ ↑⨾∊\ ⟩
    Λ₀ Id ⨾ F.mor ↓₀ ⨾ A.inc ↑₀ ⨾ (∊ \ A.R ˘)
  ≈˘⟨ ⨾-cong₂ (⨾-assoc₃₊₁ ⟨≈≈⟩ ⨾-cong₂₂ ↓⨾∊˘) ⟩
    Λ₀ Id ⨾ (F.mor ↓₀ ⨾ A.inc ↑₀ ⨾ A.R ↓₀) ⨾ ∊ ˘
  □)
  where
    module B where
      open ACSL-FromToAContext {B} public
      open ToACSL B public
    module A where
      open ACSL-FromToAContext {A} public
      open ToACSL A public
    module F where
      open AContextHom F public
      open AContextHomToACSLHom F public
    module F₁ = ACSLHom (From.mor F)
```

9.5 Categoric.OCC.CSL.ContextDualityPieces

While Agda does not manage to finish checking Categoric.OCC.CSL.ContextDuality, we include this module, which only imports all the pieces procing the categoric duality between ACSLs and AContexts, without stating the duality as a categoric equivalence between ACSL-OpCategory and the category ACH-Category.

```
open import RATH.Level
open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.SyQ

module Categoric.OCC.CSL.ContextDualityPieces {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 import Categoric.OCC.CSL.FromAContext occ leftResOp rightResOp syqOp directPower splitSymIdempot
  public using (Ctx→CSL)
open import Categoric.OCC.CSL.ToAContext     occ leftResOp rightResOp syqOp directPower
  public using (CSL→Ctx)
```

```
open import Categoric.OCC.CSL.ToFromAContextNatIsoPieces
                    occ leftResOp rightResOp syqOp directPower splitSymIdempot public
open import Categoric.OCC.CSL.ToFromAContextNatIsoNaturality
                    occ leftResOp rightResOp syqOp directPower splitSymIdempot public
open import Categoric.OCC.CSL.FromToAContextNatIsoPieces
                    occ leftResOp rightResOp syqOp directPower splitSymIdempot public
open import Categoric.OCC.CSL.FromToAContextNatIsoNaturality
                    occ leftResOp rightResOp syqOp directPower splitSymIdempot public
```

Chapter 10

Topping Off the Duality

Module Categoric.OCC.CSL.ContextDualityFromPieces (Sect. 10.1) takes the re-exports of the module Categoric.OCC.CSL.ContextDualityPieces (Sect. 9.5) and assembles them directly into a single Equivalence-of-categories record as defined in Functor.Equivalence. However, even with 52GB of heap (on a machine with 64GB of RAM), the current development version of Agda still runs out of heap space after a few days when checking this…

The two modules Categoric.OCC.CSL.ToFromAContext (Sect. 10.2) and Categoric.OCC.CSL.ToFromAContext (Sect. 10.2) only name the composed functors ToFrom and FromTo (and provide Functor modules for them); these are not necessary for Categoric.OCC.CSL.ContextDualityFromPieces (Sect. 10.1), but occur in the types of the natural isomorphisms assembled there. In the three-line bodies of these two modules, the type signature and the module definition do not contribute significantly to the typechecking cost: Even with only the definition of the composed functor, without type signature and without the module definition, type-checking either of these two modules alone currently requires one hour respectively two-and-a-half hours of time, and over half a gigabyte of stack space.

In Categoric.OCC.CSL.ToFromAContextNatIso (Sect. 10.4) and Categoric.OCC.CSL.FromToAContextNatIso (Sect. 10.5), we use the pieces from the previous chapter to assemble only the natural isomorphism between FromTo, respectively ToFrom, and the respective identity functor. FromToAContextNatIso has been checked by the Agda development version with 12GB heap in over two hours. For ToFromAContextNatIso, 28GB of heap are not sufficient (40 hours).

In Categoric.OCC.CSL.ContextDuality (Sect. 10.6), we use these natural isomorphisms to assemble the ContextDuality as a single Equivalence-of-categories record as defined in Functor.Equivalence. We hope that might type-check with less ressources than Categoric.OCC.CSL.ContextDualityFromPieces (Sect. 10.1)…

The modules of this chapter are not referenced from the top-level module AContext2 listed in Chapter 1, to keep that module checkable on standard machines.

10.1 Categoric.OCC.CSL.ContextDualityFromPieces

```
open import RATH.Level
open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.Functor.Equivalence

module Categoric.OCC.CSL.ContextDualityFromPieces {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 DirectPower directPower using (powerOp)
open import Data.AContext.Category occ leftResOp rightResOp powerOp using (ACH-Category)
open import Categoric.OCC.CSL       occ leftResOp rightResOp syqOp using (ACSL-OpCategory)
open import Categoric.OCC.CSL.ContextDualityPieces
                                    occ leftResOp rightResOp syqOp directPower splitSymIdempot

ContextDuality : CatEquivalence ACSL-OpCategory ACH-Category
ContextDuality = record
  {To                    = CSL→Ctx
  ; From                 = Ctx→CSL
  ; ToFrom = record
    {indmor              = λ {A} → L {A}
    ; naturality         = λ {A} {B} {F} → L-naturality {A} {B} {F}
    ; indmor⁻¹           = λ {A} → L⁻¹ {A}
    ; indmor-⨾-indmor⁻¹  = λ {A} → L-⨾-L⁻¹ {A}
    ; indmor⁻¹-⨾-indmor  = λ {A} → L⁻¹-⨾-L {A}
    }
  ; FromTo = record
    {indmor              = λ {A} → C {A}
    ; naturality         = λ {A} {B} {F} → C-naturality {A} {B} {F}
    ; indmor⁻¹           = λ {A} → C⁻¹ {A}
    ; indmor-⨾-indmor⁻¹  = λ {A} → C-⨾-C⁻¹ {A}
    ; indmor⁻¹-⨾-indmor  = λ {A} → C⁻¹-⨾-C {A}
    }
  }
```

10.2 Categoric.OCC.CSL.ToFromAContext

```
open import RATH.Level
open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.Functor using (Functor; module Functor; _⨾⨾_)

module Categoric.OCC.CSL.ToFromAContext {i j k₁ k₂ : Level} {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 import Categoric.OCC.CSL               occ leftResOp rightResOp syqOp
  using (ACSL-OpCategory)
open import Categoric.OCC.CSL.FromAContext occ leftResOp rightResOp syqOp directPower splitSymIdempot
  using (Ctx→CSL)
open import Categoric.OCC.CSL.ToAContext   occ leftResOp rightResOp syqOp directPower
  using (CSL→Ctx)

ToFrom : Functor ACSL-OpCategory ACSL-OpCategory
ToFrom = CSL→Ctx ⨾⨾ Ctx→CSL
module ToFrom = Functor ToFrom
```

10.3 Categoric.OCC.CSL.FromToAContext

```
open import RATH.Level
open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.OSGC.PowerOp
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.Category
open import Categoric.Functor using (Functor; module Functor; _⨾⨾_)

module Categoric.OCC.CSL.FromToAContext {i j k₁ k₂ : Level} {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
module ℙ = DirectPower directPower
open ℙ using (powerOp)
open import Data.AContext.Category          occ leftResOp rightResOp powerOp
  using (ACH-Category)
open import Categoric.OCC.CSL.FromAContext occ leftResOp rightResOp syqOp directPower splitSymIdempot
  using (Ctx→CSL)
open import Categoric.OCC.CSL.ToAContext   occ leftResOp rightResOp syqOp directPower
  using (CSL→Ctx)

FromTo : Functor ACH-Category ACH-Category
FromTo = Ctx→CSL ⨾⨾ CSL→Ctx
module FromTo = Functor (Ctx→CSL ⨾⨾ CSL→Ctx)
```

10.4 Categoric.OCC.CSL.ToFromAContextNatIso

```
open import RATH.Level
open import RATH.Data.Product using (Σ:•; proj₁; proj₂; _,_)
open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.SyQ
```

```
open import Categoric.Functor using (Identity; NatIso)

module Categoric.OCC.CSL.ToFromAContextNatIso {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 import Categoric.OCC.CSL occ leftResOp rightResOp syqOp
  using (ACSL-OpCategory)
open import Categoric.OCC.CSL.ToFromAContext occ leftResOp rightResOp syqOp directPower splitSymIdempot
  using (ToFrom)
open import Categoric.OCC.CSL.ToFromAContextNatIsoPieces
                    occ leftResOp rightResOp syqOp directPower splitSymIdempot
open import Categoric.OCC.CSL.ToFromAContextNatIsoNaturality
                    occ leftResOp rightResOp syqOp directPower splitSymIdempot

ToFromNatIso : NatIso ToFrom (Identity ACSL-OpCategory)
ToFromNatIso = record
  {indmor              = λ {A} → L {A}
  ; naturality         = λ {A} {B} {F} → L-naturality {A} {B} {F}
  ; indmor⁻¹           = λ {A} → L⁻¹ {A}
  ; indmor-⨾-indmor⁻¹  = λ {A} → L-⨾-L⁻¹ {A}
  ; indmor⁻¹-⨾-indmor  = λ {A} → L⁻¹-⨾-L {A}
  }
```

10.5 Categoric.OCC.CSL.FromToAContextNatIso

```
open import RATH.Level
open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.OSGC.PowerOp
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.Category
open import Categoric.Functor

module Categoric.OCC.CSL.FromToAContextNatIso {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 DirectPower directPower using (powerOp)
```



```
open import Data.AContext.Category occ leftResOp rightResOp powerOp
  using (ACH-Category)
open import Categoric.OCC.CSL.FromToAContext occ leftResOp rightResOp syqOp directPower splitSymIdempot
  using (FromTo)
open import Categoric.OCC.CSL.FromToAContextNatIsoPieces
  occ leftResOp rightResOp syqOp directPower splitSymIdempot
open import Categoric.OCC.CSL.FromToAContextNatIsoNaturality
  occ leftResOp rightResOp syqOp directPower splitSymIdempot

FromToNatIso : NatIso FromTo (Identity ACH-Category)
FromToNatIso = record
  {indmor              = λ {A} → C {A}
  ; naturality         = λ {A} {B} {F} → C-naturality {A} {B} {F}
  ; indmor⁻¹           = λ {A} → C⁻¹ {A}
  ; indmor-⨾-indmor⁻¹  = λ {A} → C-⨾-C⁻¹ {A}
  ; indmor⁻¹-⨾-indmor  = λ {A} → C⁻¹-⨾-C {A}
  }
```

10.6 Categoric.OCC.CSL.ContextDuality

```
open import RATH.Level
open import Categoric.OCC
import Categoric.OCC.DirectPower as OCC-DirectPower
open import Categoric.OrderedSemigroupoid.Residuals
open import Categoric.OSGC.SyQ
open import Categoric.Functor.Equivalence

module Categoric.OCC.CSL.ContextDuality {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 DirectPower directPower using (powerOp)
open import Data.AContext.Category             occ leftResOp rightResOp powerOp using (ACH-Category)
open import Categoric.OCC.CSL                  occ leftResOp rightResOp syqOp using (ACSL-OpCategory)
open import Categoric.OCC.CSL.FromAContext occ leftResOp rightResOp syqOp directPower splitSymIdempot
  using (Ctx→CSL)
open import Categoric.OCC.CSL.ToAContext       occ leftResOp rightResOp syqOp directPower
  using (CSL→Ctx)
open import Categoric.OCC.CSL.ToFromAContextNatIso
                                               occ leftResOp rightResOp syqOp directPower splitSymIdempot
open import Categoric.OCC.CSL.FromToAContextNatIso
                                               occ leftResOp rightResOp syqOp directPower splitSymIdempot

ContextDuality : CatEquivalence ACSL-OpCategory ACH-Category
ContextDuality = record
  {To       = CSL→Ctx
  ; From    = Ctx→CSL
  ; ToFrom  = ToFromNatIso
  ; FromTo  = FromToNatIso
```

Chapter 11

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 than 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, and proved that this category is dual to 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.

With a development of this size and complexity we would find it hard to believe that, for example, no allegory laws that do not necessarily hold in the OCC setting have been used accidentally, were it not for the confidence provided by the mechanised proof checker.

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. URL `http://hdl.handle.net/11375/17276`.

Rudolf Berghammer, Gunther Schmidt, and Hans Zierer. Symmetric quotients. Technical Report TUM-INFO 8620, TU München, Fak. 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, Univ. 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, with 456 pages literate document output. http://relmics.mcmaster.ca/RATH-Agda/, February 2014b.

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.