Abstract Context Lattices Formalised for Concrete Applications

AContext-1.0

WOLFRAM KAHL

kahl@cas.mcmaster.ca

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

7 January 2014

Abstract

Using the the dependently-typed programming language Agda, we formalise a category of algebraic contexts with relational homomorphisms presented by Jipsen (2012); Moshier (2013). We do this in the abstract setting of locally ordered categories with converse (OCCs) with residuals and direct powers, 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

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.

We now set out to mechanise the basis of these developments, and for the sake of reusability we abstract the sets and relations that constitute contexts to objects and morphisms of suitable categories and semigroupoids. Besides the mechanised formalisation itself, our main contribution is the insight that Moshier's relational context category can be formalised in categories of "abstract relations" where neither meet (intersection) nor join (union) are available, and that a large part of this development does not even require the presence of identity relations.

Overview

Since formal concept analysis concentrates on subsets of the constituent sets of the contexts we are interested in, we formalise, in Chapter 2, an 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), directly in the setting of locally ordered semigroupoids with converse (OSGCs). Adding also residuals to that setting proves sufficient for the formalisation of the "compatibility conditions" of Moshier's relational context homomorphisms, in Sect. 3.1. Defining composition of these homomorphisms requires making identity relations available, that is, moving from semigroupoids to categories (Sect. 3.2); locally ordered categories with converse (OCCs) and residuals and a power operator are sufficient to formalise the context category, in Sect. 3.3.

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

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

Chapter 2

Power Operators

2.1 Categoric.OSGC.PowerOp

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

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

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

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

Before defining power allegories as a special kind of division allegories, Freyd and Scedrov (1990, Sect. 2.4) give an alternative definition that (if recast in a typed setting) enriches allegories with a type operator ℙ : Obj → Obj, a transformation ∋ : {B : Obj} → Mor (ℙ B) B, and a 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; 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
```

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

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

We follow them in naming Λ “power transpose”; we present the equivalence as two implications in the Fields below:

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

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 IdℙP
    ≈₁⟨ Λ-⨾∈˘ {f = f ⨾₁ IdℙP} ⟨≈˘≈⟩ Λ-cong ⨾-assoc ⟩
      Λ (Mapping.mor f ⨾ Λ₀ (∈ ˘) ⨾ ∈ ˘)
    ≈₁⟨ Λ-cong (⨾-cong₂ Λ⨾∈˘) ⟩
      Λ (Mapping.mor f ⨾ ∈ ˘)
    ≈₁⟨ Λ-⨾∈˘ {f = f} ⟩
      f
    □₁

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.IdℙP
  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 ˘
    □)

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 staisfying conditions appropriate for a “subset relation”:

```
record IsPowerOrder (Ω : {A : Obj} → Mor (ℙ A) (ℙ A)) : Set (i ⊍ j ⊍ k₁ ⊍ k₂) where
  field
    ∈⨾Ω : {A : Obj} → ∈ ⨾ Ω {A} ⊑ ∈
    Ω-universal : {A : Obj} {R : Mor (ℙ A) (ℙ A)} → ∈ ⨾ R ⊑ ∈ → R ⊑ Ω
  Ω˘-universal : {A : Obj} {R : Mor (ℙ A) (ℙ A)} → R ⨾ ∈ ˘ ⊑ ∈ ˘ → R ⊑ Ω ˘
  Ω˘-universal {A} {R} R⨾∈˘⊑∈˘ = ⊑-˘-swap (Ω-universal (⊑-begin
      ∈ ⨾ R ˘
    ≈˘⟨ ˘-involutionRightConv ⟩
      (R ⨾ ∈ ˘) ˘
    ⊑⟨ ˘-⊑-swap R⨾∈˘⊑∈˘ ⟩
      ∈
    □))
```

2.2 Categoric.OSGC.PowerOrder

```
module Categoric.OSGC.PowerOrder {i j k₁ k₂} {Obj : Set i} (osgc : OSGC j k₁ k₂ Obj)
  (leftResOp   : LeftResOp (OSGC.orderedSemigroupoid osgc))
```

```
  (rightResOp : RightResOp (OSGC.orderedSemigroupoid osgc))
  (powerOp : PowerOp osgc) where
open OSGC osgc
open ResidualOps leftResOp rightResOp
open OSGC-Residuals osgc leftResOp rightResOp
open PowerOp osgc powerOp
```

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

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

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

```
Ω-trans : {A : Obj} → Ω ⨾ Ω ⊑ Ω {A}
Ω-trans = \-cancel-middle
Idℙ⊑Ω : {A : Obj} → Idℙ₀ ⊑ Ω {A}
Idℙ⊑Ω = \-universal (⨾-cong₁ ˘˘ ⟨≈˘⊑⟩ ˘⨾Λ)
Idℙ⊑Ω′ : {A : Obj} → Idℙ₀ ⊑ Ω {A}
Idℙ⊑Ω′ = \-universal (⊑-begin
    ∈ ⨾ Idℙ₀
  ≈˘⟨ ⨾-cong₁ ˘˘ ⟩
    (∈ ˘) ˘ ⨾ 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 / ∈ ˘
  □


Ma : {X A : Obj} → Mor X (ℙ A) → Mor X (ℙ A)
Ma R = R ˘ \ Ω
Ma′ : {X A : Obj} (R : Mor X (ℙ A)) → Ma R ≈ (∈ ⨾ R ˘) \ ∈
Ma′ R = ≈-begin
    R ˘ \ (∈ \ ∈)
  ≈⟨ \\ ⟩
    (∈ ⨾ R ˘) \ ∈
  □
Ma˘′ : {X A : Obj} (R : Mor X (ℙ A)) → Ma R ˘ ≈ ∈ ˘ / (R ⨾ ∈ ˘)
Ma˘′ R = ≈-begin
    Ma R ˘
  ≈⟨ ˘-cong (Ma′ R) ⟩
    ((∈ ⨾ R ˘) \ ∈) ˘
```

```
  ≈⟨ \-˘ ⟩
    ∈ ˘ / (∈ ⨾ R ˘) ˘
  ≈⟨ /-cong₂ ˘-involutionRightConv ⟩
    ∈ ˘ / (R ⨾ ∈ ˘)
  □

Mi : {X A : Obj} → Mor X (ℙ A) → Mor X (ℙ A)
Mi R = R ˘ \ Ω ˘

Mi′ : {X A : Obj} (R : Mor X (ℙ A)) → Mi R ≈ R ˘ \ (∈ ˘ / ∈ ˘)
Mi′ R = ≈-begin
    R ˘ \ (∈ \ ∈) ˘
  ≈⟨ \-cong₂ \-˘ ⟩
    R ˘ \ (∈ ˘ / ∈ ˘)
  □

MiMa′ : {X A : Obj} (R : Mor X (ℙ A)) → Mi (Ma R) ≈ (∈ ˘ / (R ⨾ ∈ ˘)) \ (∈ ˘ / ∈ ˘)
MiMa′ R = ≈-begin
    Mi (Ma R)
  ≈⟨ Mi′ (Ma R) ⟩
    (Ma R) ˘ \ (∈ ˘ / ∈ ˘)
  ≈⟨ \-cong₁ (Ma˘′ R) ⟩
    (∈ ˘ / (R ⨾ ∈ ˘)) \ (∈ ˘ / ∈ ˘)
  □
```

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

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

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

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

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

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

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

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

2.3 Categoric.OSGC.PowerRes

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

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

open OSGC osgc
open PowerOp osgc powerOp
```

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

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

Chapter 3

Abstract Contexts

3.1 Data.AContext.InOSGC

```
module Data.AContext.InOSGC {i j k₁ k₂} {Obj : Set i} (osgc : OSGC j k₁ k₂ Obj)
                            (leftResOp  : LeftResOp (OSGC.orderedSemigroupoid osgc))
                            (rightResOp : RightResOp (OSGC.orderedSemigroupoid osgc))
                            (powerOp : PowerOp osgc) where
open OSGC osgc
open ResidualOps leftResOp rightResOp
open OSGC-Residuals osgc leftResOp rightResOp
open PowerOp osgc powerOp
open import Categoric.OSGC.PowerOrder osgc leftResOp rightResOp powerOp
private
  module MapSG = Semigroupoid (MapSG osgc)
open Semigroupoid₁ (MapSG osgc)
```

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

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

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


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


↑˘⨾∈˘ : {A B : Obj} {R : Mor A B} → R ↑₀ ˘ ⨾ ∈ ˘ ⊑ ∈ \ (R ˘)
↑˘⨾∈˘ {A} {B} {R} = \-universal (⊑-begin
```

```
     ∈ ⨾ (Λ₀ (∈ \ R)) ˘ ⨾ ∈ ˘
  ≈⟨ ⨾-assocL ⟨≈≈˘⟩ ⨾-cong₁ ˘-involutionRightConv ⟩
     (Λ₀ (∈ \ R) ⨾ ∈ ˘) ˘ ⨾ ∈ ˘
  ≈⟨ ⨾-cong₁ (˘-cong Λ⨾∈˘) ⟩
     (∈ \ R) ˘ ⨾ ∈ ˘
  ≈˘⟨ ˘-involution ⟩
     (∈ ⨾ (∈ \ R)) ˘
  ⊑⟨ ˘-monotone \-cancel-outer ⟩
     R ˘
  □)
```

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

The fact that the operators _↑ and _↓ form a Galois connection, set-theoretically

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

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

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

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

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

```
          □))
        (/-universal (⊑-begin
              (∈ / (Q ⨾ Λ₀ (∈ \ R ˘))) ⨾ (Q ⨾ ∈ ˘)
          ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-⨾-inner ⟩
              (∈ / Λ₀ (∈ \ R ˘)) ⨾ ∈ ˘
          ⊑⟨ ⨾-monotone₂ (proj₁ Λ-total ⟨⊑≈⟩ ⨾-assoc) ⟩
              (∈ / Λ₀ (∈ \ R ˘)) ⨾ Λ₀ (∈ \ R ˘) ⨾ Λ₀ (∈ \ R ˘) ˘ ⨾ ∈ ˘
          ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ /-cancel-outer ⟩
              ∈ ⨾ Λ₀ (∈ \ R ˘) ˘ ⨾ ∈ ˘
          ≈⟨ ⨾-assocL ⟨≈≈˘⟩ ⨾-cong₁ ˘-involutionRightConv ⟩
              (Λ₀ (∈ \ R ˘) ⨾ ∈ ˘) ˘ ⨾ ∈ ˘
          ≈⟨ ⨾-cong₁ (˘-cong Λ⨾∈˘) ⟩
              (∈ \ R ˘) ˘ ⨾ ∈ ˘
          ⊑⟨ ⨾-cong₁ \˘-˘ ⟨≈⊑⟩ /-cancel-outer ⟩
              R
          □))) ⟩
        (∈ / (Q ⨾ Λ₀ (∈ \ R ˘))) ˘
      ≈⟨ /-˘ ⟩
        (Q ⨾ Λ₀ (∈ \ (R ˘))) ˘ \ ∈ ˘
        □) ⟩
        Λ₀ ((Q ⨾ Λ₀ (∈ \ (R ˘))) ˘ \ ∈ ˘)
  □


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

```
      (∈ / (Q ⨾ Λ₀ (∈ \ R))) ˘
    ≈⟨ /-˘ ⟩
      (Q ⨾ Λ₀ (∈ \ R)) ˘ \ ∈ ˘
      □) ⟩
      Λ₀ ((Q ⨾ Λ₀ (∈ \ R)) ˘ \ ∈ ˘)
  □
```

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.

In the power-allegory of sets, if we set Q = ⊥ and R = ⊤, then we have:

```
  (∈ / Q) \ R
≈⟨ ≈-refl ⟩    -- Def. Q and R
  (∈ / ⊥) \ ⊤
≈⟨ ⊤ ⨾ ⊥ ⊑ ∈ ⟩
  ⊤ \ ⊤
≈⟨ ⊤ ⨾ ⊤ ⊑ ⊤ ⟩
  ⊤
≉⟨  {-!!! -}  ⟩
  ⊥
≈˘⟨ ⊥-leftZero ⟩
  ⊥ ⨾ (∈ \ ⊤)
≈⟨ ≈-refl ⟩    -- Def. Q and R
  Q ⨾ (∈ \ R)
```

Therefore, the closest we can have to Glb-cocontinuous (R ↑) is the following:

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

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

```
      (Λ₀ (Q ˘ \ ∈ ˘) ⨾ Λ₀ (∈ \ R)) ⨾ ∈ ˘
  ≈⟨ Λ⇒∈ {f = Λ (Q ˘ \ ∈ ˘) ⨾₁ Λ (∈ \ R)} (≈-begin
      Λ₀ (Q ˘ \ ∈ ˘) ⨾ Λ₀ (∈ \ R)
    ≈⟨ assumption ⟩
      Λ₀ ((Q ⨾ Λ₀ (∈ \ R)) ⨾ ∈ ˘)
    ≈⟨ Λ-cong (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘) ⟩
      Λ₀ (Q ⨾ (∈ \ R))
    □) ⟩
    (Q ⨾ (∈ \ R))
  □
```

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 ↓↑)

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

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

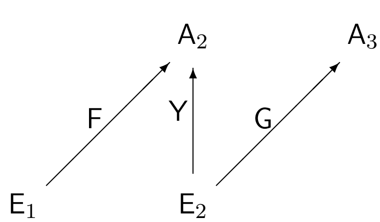
```
∈⨾↑↓ : {A B : Obj} {R : Mor A B} → ∈ ⨾ R ↑↓₀ ⊑ ∈
∈⨾↑↓ {A} {B} {R} = ⊑-begin
    ∈ ⨾ R ↑↓₀
  ⊑⟨ ⨾-monotone₁ (⊑-˘-swap ∈˘⊑↑↓⨾∈˘ ⟨⊑≈⟩ ˘-involutionRightConv) ⟩
    (∈ ⨾ (R ↑↓₀) ˘) ⨾ R ↑↓₀
  ⊑⟨ ⨾-assoc ⟨≈⊑⟩ proj₂ (Mapping.unival (R ↑↓)) ⟩
    ∈
  □

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

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

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

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

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

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

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

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


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


↑↓-idempotent : {A B : Obj} {R : Mor A B} → R ↑↓ ⨾₁ R ↑↓ ≈₁ R ↑↓
↑↓-idempotent = ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ↓⨾↑↓
↓↑-idempotent : {A B : Obj} {R : Mor A B} → R ↓↑ ⨾₁ R ↓↑ ≈₁ R ↓↑
↓↑-idempotent = ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ ↓↑⨾↓


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

```
    ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ↑↓⨾∈˘ ⟩
      Λ₀ (∈ \ (Q ˘)) ⨾ (R / (∈ \ R)) ˘
    ≈⟨ ⊑-antisym
      (\-universal (⊑-begin
            ∈ ⨾ Λ₀ (∈ \ (Q ˘)) ⨾ (R / (∈ \ R)) ˘
         ⊑⟨ ⨾-assocL ⟨≈⊑⟩ /-universal′ (⊑-begin
            ∈ ⨾ Λ₀ (∈ \ (Q ˘))
            ⊑⟨ /-universal (⊑-begin
                  (∈ ⨾ Λ₀ (∈ \ (Q ˘))) ⨾ ∈ ˘
               ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘ ⟩
                  ∈ ⨾ (∈ \ (Q ˘))
               ⊑⟨ \-cancel-outer ⟩
                  Q ˘
                 □) ⟩
               Q ˘ / ∈ ˘
            ≈˘⟨ \-˘ ⟩
               (∈ \ Q) ˘
            ≈˘⟨ ˘-cong (\T∘S/∘\S p) ⟩
               ((R / (∈ \ R)) \ Q) ˘
            ≈⟨ \-˘ ⟩
               Q ˘ / (R / (∈ \ R)) ˘
            □) ⟩
            Q ˘
         □))
      (⊑-begin
            ∈ \ (Q ˘)
         ≈˘⟨ Λ⨾∈˘ ⟩
            Λ₀ (∈ \ (Q ˘)) ⨾ ∈ ˘
         ⊑⟨ ⨾-monotone₂ (˘-monotone ⊑-S/∘\S) ⟩
            Λ₀ (∈ \ (Q ˘)) ⨾ (R / (∈ \ R)) ˘
         □) ⟩
      ∈ \ (Q ˘)
    □) ⟩
    Λ₀ (∈ \ (Q ˘))
  ≈⟨ ≈-refl ⟩
    Q ↓₀
  □
```

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

We name the necessary conditions as "source compatibility" respectively "target compatibility", ans show equivalent formulations:

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

TrgCompat : {A₁ A₂ B : Obj} (R : Mor A₁ B) (Y : Mor A₂ B) → Set k₁
TrgCompat R Y = Y ↓↑ ⨾₁ R ↓ ≈₁ R ↓


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

SrcCompat-⇐ : {A B₁ B₂ : Obj} (X : Mor A B₁) (R : Mor A B₂)
             → X ↑↓₀ ⨾ ∈ ˘ ⊑ R ↑↓₀ ⨾ ∈ ˘ → SrcCompat X R
SrcCompat-⇐ X R X■R = ↓⨾↑↓′ (˘-isotone (↑↓⨾∈˘ ⟨≈˘⊑⟩ X■R ⟨⊑≈⟩ ↑↓⨾∈˘))


TrgCompat-⇒ : {A₁ A₂ B : Obj} (R : Mor A₁ B) (Y : Mor A₂ B)
             → TrgCompat R Y → Y ↓↑₀ ⨾ ∈ ˘ ⊑ R ↓↑₀ ⨾ ∈ ˘
TrgCompat-⇒ R Y trgCompat = ⊑-begin
    Y ↓↑₀ ⨾ ∈ ˘
  ⊑⟨ ⨾-monotone₂ ∈˘⊑↓↑⨾∈˘ ⟩
    Y ↓↑₀ ⨾ R ↓↑₀ ⨾ ∈ ˘
  ≈⟨ ⨾-assocL ⟨≈≈⟩ ⨾-cong₁ (⨾-assocL ⟨≈≈⟩ ⨾-cong₁ trgCompat) ⟩
    R ↓↑₀ ⨾ ∈ ˘
  □

TrgCompat-⇐ : {A₁ A₂ B : Obj} (R : Mor A₁ B) (Y : Mor A₂ B)
             → Y ↓↑₀ ⨾ ∈ ˘ ⊑ R ↓↑₀ ⨾ ∈ ˘ → TrgCompat R Y
TrgCompat-⇐ R Y Y■R = ∈⇒Λ {f = Y ↓↑ ⨾₁ R ↓} (≈-begin
    (Y ↓↑₀ ⨾ R ↓₀) ⨾ ∈ ˘
  ≈⟨ ⨾-assoc ⟨≈≈⟩ ⨾-cong₂ Λ⨾∈˘ ⟩
    Y ↓↑₀ ⨾ (∈ \ R ˘)
  ≈⟨ \-inner-⨾ (Mapping.prf (Y ↓↑)) ⟩
    (∈ ⨾ Y ↓↑₀ ˘) \ R ˘
```

```
  ≈⟨ \-cong₁ (˘-involutionRightConv ⟨≈˘≈⟩ ˘-cong ↓↑⨾∈˘) ⟩
    ((Y / ∈ ˘) \ Y) ˘ \ R ˘
  ≈˘⟨ /-˘ ⟩
    (R / ((Y / ∈ ˘) \ Y)) ˘
  ≈⟨ ˘-cong (T/∘\S∘S/ (↓↑⨾∈˘ ⟨≈˘⊑⟩ Y▬R ⟨⊑≈⟩ ↓↑⨾∈˘)) ⟩
    (R / ∈ ˘) ˘
  ≈⟨ /˘-˘ ⟩
    ∈ \ R ˘
    □)
```

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

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

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

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

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

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

For each context, its incidence defines its identity homomorphism:

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

3.2 Data.AContext.InOCC

Abstract Contexts

```
module Data.AContext.InOCC {i j k₁ k₂} {Obj : Set i} (occ : OCC j k₁ k₂ Obj)
                           (leftResOp   : LeftResOp (OCC.orderedSemigroupoid occ))
                           (rightResOp : RightResOp (OCC.orderedSemigroupoid occ))
                           (powerOp : PowerOp (OCC.osgc occ)) where
open OCC occ
open ResidualOps leftResOp rightResOp
```

```
open OSGC-Residuals osgc leftResOp rightResOp
open OrdCat-Residual-Props orderedCategory leftResOp rightResOp
open PowerOp osgc powerOp
open import Categoric.OSGC.PowerOrder osgc leftResOp rightResOp powerOp
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 ∈ f$\{b\}$:

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

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

We always have [R ↓] ≈ R:

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

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

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

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

3.3 Data.AContext.Category

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

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

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

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

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

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

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

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

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

```
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 4

Conclusion

Beyond the theoretically interesting fact that context categories can be formalised in OCCs with residuals and powers, this development also demonstrates that such an essentially theoretical development can be fully mechanised and still be presented in readable calculational style, where writing is not significantly more effort that a conventional calculational presentation in LaTeX.

In comparison with similar developments in Isabelle/HOL ((Kahl, 2003)), the use of Agda enables a completely natural mathematical treatment of categories, nested calculational proofs, and direct use of theories as modules of executable programs.

Bibliography

Rudolf Berghammer, Gunther Schmidt, and Hans Zierer. Symmetric quotients. Technical Report TUM-INFO 8620, Technische Universität München, Fakultät für Informatik, 1986. 18 p.

Rudolf Berghammer, Gunther Schmidt, and Hans Zierer. Symmetric quotients and domain constructions. *Inform. Process. Lett.*, 33(3):163–168, 1989.

Richard S. Bird and Oege de Moor. *Algebra of Programming*, volume 100 of *International Series in Computer Science*. Prentice Hall, 1997.

Marcel Erné. Categories of contexts. Preprint, `http://www.iazd.uni-hannover.de/~erne/preprints/CatConts.pdf`, 2005.

Peter J. Freyd and Andre Scedrov. *Categories, Allegories*, volume 39 of *North-Holland Mathematical Library*. North-Holland, Amsterdam, 1990. ISBN 0-444-70368-3 and 0-444-70367-5 (pbk).

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. Calculational relation-algebraic proofs in Isabelle/Isar. In Rudolf Berghammer, Bernhard Möller, and Georg Struth, editors, *Relational and Kleene-Algebraic Methods in Computer Science*, volume 3051 of *LNCS*, pages 178–190. Springer, 2003.

M. Andrew Moshier. A relational category of polarities. (unpublished draft), November 2013.

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.