A Mechanisation of Internal Galois Connections In Order Theory Formalised Without Meets

Musa Al-hassy

alhassy@gmail.com

A Thesis Submitted to the School of Graduate Studies
in Partial Fulfillment of the Requirement
for the Degree
Master of Science.

McMaster University

MASTER OF SCIENCE (2015)	McMaster University
(Computer Science)	Hamilton, Ontario
TITLE	A Mechanisation of Internal Galois Connections In Order Theory Formalised Without Meets
AUTHOR:	Musa Al-hassy, B.Sc. (Hons) (McMaster University)
SUPERVISOR:	Dr. Wolfram Kahl
NUMBER OF PAGES:	83

Abstract

Using the the dependently-typed programming language Agda, we formalise orders, with attention to the theory of Galois Connections, and showcase it by formalising a few results of the category of algebraic contexts with relational homomorphisms presented by Jipsen (2012); Moshier (2013).

We aim to exhibit an internal theory of Galois Connections and Closure operators where the ambient space need not have a notion of meets (intersections), which are the usual medium in presenting antisymmetry of partial orders. Instead we consider 'symmetric quotients' as being the relational counterpart of propositional calculus' primitive connective: equivalence. We argue that it as a more natural primitive than meet — especially its connection with certain proof heuristics regarding posets. Moreover, not only do we constrain ourselves to an unconventional set of primitive operators, but in fact we discard the familiar setting of relations and sets in favour of the more general setting of ordered categories with converse (OCCs) — in fact, a large portion does not require identities and so semigroupoids may be used instead.

Acknowledgements

I would like to thank and acknowledge my supervisor's contributions. More importantly, I am grateful to him for all of his support and encouragement. He is not only a wonderful supervisor, but also a good friend.

I am a "master's student", that is, a student of a master of the field. Indeed that is Wolfram Kahl: a master of the field.

Contents

1. Introduction **1**

2. External Galois Connections **3**
- 2.1. Defining a Poset . . . 3
- 2.2. Galois Connections . . . 4
- 2.3. Ubiquity of Galois Connections . . . 5
- 2.4. Fundamental Properties . . . 8
- 2.5. Properties of the Lower Adjoint . . . 9
- 2.6. Properties of the Upper Adjoint, by Duality . . . 13
- 2.7. Conclusion . . . 14

3. Choosing Our Operations **15**
- 3.1. Eliminating Quantifiers . . . 15
- 3.2. Primitive Operations . . . 16
- 3.3. Expressing Antisymmetry . . . 17
- 3.4. Conclusion . . . 17

4. Internal Preorder Theory **19**
- 4.1. Categoric.OSGC.Preorder . . . 19
  - 4.1.1. The Dual Preorder . . . 19
  - 4.1.2. Indirect inclusion . . . 20
  - 4.1.3. Bounds . . . 20
- 4.2. Categoric.OCC.Preorder . . . 27
  - 4.2.1. Retract Preorder and Preorder Invaraince . . . 27
  - 4.2.2. Residual Induced Preorders . . . 28
- 4.3. Categoric.OSGC.Preorder.Closure . . . 30
  - 4.3.1. Increasing . . . 30
  - 4.3.2. Quasi-idempodentcy . . . 32
  - 4.3.3. Monotonicity . . . 32
  - 4.3.4. Piecewise Closure Characterization . . . 34
  - 4.3.5. Dually: Interior Operator . . . 34
  - 4.3.6. Conclusion . . . 36
- 4.4. Categoric.OSGC.Preorder.Galois . . . 36
  - 4.4.1. Co-connection . . . 37
  - 4.4.2. Cancellation Laws . . . 37

4.4.3. Monotonicity 39
4.4.4. Quasi-semi-inverse Laws 40
4.4.5. Quasi-absorption Laws 41
4.4.6. Image Isotonicity 42
4.4.7. Induced Interior 43
4.4.8. Induced Closure 44
4.5. Categoric.OCC.Preorder.Galois 45
4.6. Categoric.OSGC.Preorder.Extrema 46
4.6.1. A Risky Duality 47
4.7. Conclusion 51

5. Internal Partial Order Theory **52**
5.1. Categoric.OCC.Order 52
5.1.1. Categoric.OCC.Order 52
5.1.2. Indirect Equality 53
5.1.3. Univalence 56
5.1.4. Extrema 57
5.1.5. Order Constructions 58
5.1.6. Preorders Induced By Residuals and Endowed with Syqs 60
5.1.7. Singletons 60
5.1.8. Orders Induced by Residuation and Endowed with Comprehension . . 61
5.1.9. Power Transpose Λ 63
5.2. Categoric.OCC.Order.Closure 64
5.2.1. Idempotence and Range Closure 64
5.2.2. GLB Closure 64
5.2.3. Dualitiy and LUB Closure 65
5.3. Categoric.OCC.Order.Galois 66
5.3.1. Semi-inverses 66
5.3.2. Map Absorption 67
5.3.3. Idempotency and Coclosure 67
5.3.4. Idempotency and Closure 67
5.4. Conclusion 68

6. Polarities: An Application **69**

7. Conclusion **72**

A. Galois Connection Properties **75**
A.1. Constructions 75
A.2. Most Commonly Used Laws 75

B. Ordered Categories with Converse **76**
B.1. Objects 76

B.2. Composition . . . 77
B.3. Identities . . . 77
B.4. Converse Operator . . . 78
B.5. Definitions of Properties of Morphisms . . . 78
B.6. Swaps . . . 79

C. Residuals **80**
C.1. Definition . . . 80
C.2. Interaction with Mappings . . . 81
C.3. Interaction with Converse . . . 81
C.4. Interaction with Identities . . . 81

D. Symmetric Quotients **82**
D.1. Definition . . . 82
D.2. Interaction with Mappings and More . . . 83
D.3. Interaction with Residuals . . . 83
D.4. Interaction with Identities . . . 83

1. Introduction

Order theory typically is based on the topos of sets and functions between them. The setting can be weakened to an allegory — a certain class of ordered categories with converse and meets — , but we go a step further and present them without meets but rather with residuals instead. The notion of residuation naturally arises when one wants to even discuss bounds, namely Sect. 4.1.3, and so rather than employing meets for the order antisymmetry and residuals for bounds, we take a more homogeneous approach and use residual operators as a common base — Appendix C provides a condensed reference for residuals.

We mechanise the basis of these developments, and for the sake of reusability we abstract the sets and relations that constitute orders to objects and morphisms of suitable categories and semigroupoids. Besides the mechanised formalisation itself, our main contribution is the insight that Galois connections, a form of categorial adjunctions, 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. To the best of our knowledge, such an approach is usually mentioned but hardly carried out and pointwise reasoning is preferred instead. Our presentation will extend the applicability of the results of the theory of Galois connections beyond the classical setting of sets and functions to the more exotic and lucrative contexts employed by computer scientists working with formal methods or using Galois connections as a form of program derivation, as Mu and Oliveira (2012); Garcia-Pardo et al. (2013).

Afterwards we feature the results in the setting of concept lattices, deriving properties of polarities from our general order theorems. This brief venture was motivated by Moshier (2013) and has already been dealt with in some detail in the submitted paper by Al-hassy and Kahl (2015).

It is to be noted that Wolfram Kahl yielded major contributions in Sect. 4.1, Sect. 4.2, Sect. 4.6, and Sect. 5.1. Whereas my contributions consisted of the remaining material; namely, Sect. 4.3 to Sect. 4.5, Sect. 5.1.2, and Sect. 5.2 and Sect. 5.3 — less the glb-closed-* results, which are due to Kahl.

Overview

We assume the reader is familiar with the fundamentals of category theory and has some exposure to the theory of dependent types — if this is not the case, then the reader may interpret all symbols as the usual relational ones and take this study as nothing more than a point-free approach to Galois connections. We use the Agda language since it is well supported and permits Unicode, hence the resulting presentation is rather close to a traditional mathematical one. Our usage of the Agda language is basic enough that an average computing scientist should be able to infer the purpose of the constructs without much trouble. As such, we make no mention of the workings of the language except the following: nearly all sequences of characters without interspersed whitespace may constitute an identifier and this is a common convention. Moreover, all of our chapters, with the exception of the first and the third chapters, are literate Agda files; i.e., actual working code.

In Chapter 2, we begin with a brief presentation of external, or pointwise, theory of Galois connections — which is summarised informally in Appendix A —, and next, in Chapter 3, we turn to the motivation for our choice of operators for the abstract setting. Next, in Chapter 4, we move on to formalising preorders, notions of upper and lower bounds, closure and co-closure operators, and finally Galois connections.

Formalisation of order relations then initiates Chapter 5. 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 ordered semigroupoids with converse (OSGC), respectively ordered categories with converse (OCC), with residuals and symmetric quotients (sqys) — Appendix B briskly enumerates relevant details of OSGC's and OCC's, while Appendix D provides a quick reference for sqys.

We conclude with Chapter 6 by witnessing a quaint bird's eye-view of the notion of polarities needed for formal concept analysis. In particular, we aim to reduce the amount of similar proof terms of Kahl (2014a), which formalises them without the common base of Galois connections.

2. External Galois Connections

When one considers an algebra of a specification or a signature, one usually automatically assumes the interpretation of the sorts and symbols as sets, functions, and relations. This is an *external* approach. However, once we pick a category, one may wonder whether such a logic can be interpreted within the category, via objects and morphisms for the sorts and function symbols. This is the *internal* approach. Sometimes the ambient category needs to have additional structure, such as the existence of all finite products or that the hom-sets are locally ordered; just to name a few.

Why consider an internal approach? *Reusability!* The ensuing results are then applicable to many an exotic beast that resembles an externally defined structure but is more complex. Of course, one may be able to rely on the external results, but then great care is needed to maintain the invariant properties of the beast. For example, consider viewing graph transformations at the level of pushouts, versus the levels of nodes and edges. Clearly the former interface is more elegant and manageable, while the latter grows exceedingly fast and the details become too cumbersome to handle except for small cases.

The remainder of this chapter is organised as follows. We begin by recalling the definition of an external pose, motivate the desire to study these connections, and, after that, providing a quick theory of (external) Galois connections.

2.1. Defining a Poset

A poset is nothing more than a type Carrier with a pair of relations _≈_, _≤_ such that the following list of axioms holds:

```
≈-refl  : {x : Carrier}   → x ≈ x
≈-sym  : {x y : Carrier} → x ≈ y → y ≈ x
≈-trans : {x y z : Carrier} → x ≈ y → y ≈ z → x ≈ z

≤-reflexive : {x y : Carrier} → x ≈ y → x ≤ y
≤-trans     : {x y z : Carrier} → x ≤ y → y ≤ z → x ≤ z
≤-antisym  : {x y : Carrier} → x ≤ y → y ≤ x → x ≈ y
```

That is to say, _≤_ is an ordering relation *with respect to* an equivalence relation _≈_.

Orderings yields a class of useful indirect proof techniques, such as

```
  -- “indirect-inclusion, from the right, to inclusion”
indir-≤→≤ : {x y : Carrier} → (∀ {z} → y ≤ z → x ≤ z) → x ≤ y
```

```
indir-≤→≤ {x} {y} pf = pf ≤-refl where ≤-refl = ≤-reflexive ≈-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)
```

Of course there are many other variations.

2.2. Galois Connections

Let us turn to the notion of (monotone) Galois connections, as popularized 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 formalism; which we then employ in the internal setting, e.g., Sect. 4.1.3.

Notational remark: we use the Agda libraries for equational, or "calculational", reasoning. Namely, we can present our proofs as a sequence of steps with justifications relating them and terminate them with the symbol □. When the proofs are small, or in proof steps, we use combinators: if it is assumed that, e.g., we have three relations ~, &, #, then the infix name _⟨~&⟩_ would denote the proof ∀ {x y z} → x ~ y → y & z → x # y, if it exists.

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

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

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

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 : Monotone A B L → Monotone 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 $(\mathsf{f} \mathrel{\dot{\leq}} \mathsf{g}) \;\Leftrightarrow\; (\forall \{\mathsf{x}\} \to \mathsf{f}\,\mathsf{x} \leq \mathsf{g}\,\mathsf{x})$ 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: $\langle \mathsf{L}, \leq \rangle \dashv \langle \mathsf{U}, \sqsubseteq \rangle \;\Leftrightarrow\; \langle \mathsf{U}, \sqsupseteq \rangle \dashv \langle \mathsf{L}, \geq \rangle$.

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

2.3. 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:

$$\mathsf{k} \leq \mathsf{m} \div \mathsf{n} \;\Leftrightarrow\; \mathsf{k} \times \mathsf{n} \leq \mathsf{m}$$

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., $(\mathsf{n} \div \mathsf{m}) \div \mathsf{d} = \mathsf{n} \div (\mathsf{d} \times \mathsf{m})$ 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:

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

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

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

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

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: $\mathsf{K}\,x\,y = x$. So we work with the complex by shifting to the manageable:

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

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

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

  $$\forall x, y, z \bullet \mathsf{max}\,(x, y) \leq z \;\Leftrightarrow\; x \leq z \wedge y \leq z$$

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

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

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

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

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

Examples of Galois Connections

Let us observe a quick enumeration of common connections.

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

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

  Then, we find

$$\forall\, Q, R, S \bullet Q \sqsubseteq R \setminus S \Leftrightarrow R \mathbin{;} Q \sqsubseteq S \Leftrightarrow R \sqsubseteq S \,/\, Q$$

  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:

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

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

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

  Note: $(f, \le) \dashv\breve{}\; (g, \sqsubseteq) \;\Leftrightarrow\; (f, \le\breve{}\,) \dashv (g, \sqsubseteq)$.
  See Appendix C 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: $R\uparrow \dashv\breve{}\; R\downarrow$ where

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

    and likewise $R\downarrow \;=\; (R\,\breve{}\,)\uparrow$. Alternatively, if we construe R as a relating objects to their properties, then $R \uparrow (A)$ = "the properties common to all objects A " and $R \downarrow (B)$ = "the objects satisfying all properties B ".
  - The original Galois connection can naively be seen as the polars between functions and elements induced by relation $f\,R\,x \;\Leftrightarrow\; f\,x = 0$.

– Syntax is adjoint to semantics: $(\vDash\uparrow) \dashv\breve{}\ (\vDash\downarrow)$, where for a given signature, a sentence s and an algebra A over that signature, the 'true of (interpretation)' relation $\vDash$ is defined: $A \vDash s$ if and only if s is true when interpreted in model / algebra A.

  Exercise: what is the relation between $(\vDash\downarrow) ⨾ (\vDash\uparrow)$ and the notion of 'logical consequence'?

– Connections between powersets: $\mathsf{f} \dashv\breve{}\ \mathsf{g}$, between powersets, if and only if $\langle \mathsf{f}, \mathsf{g}\rangle = \langle \mathsf{R}\uparrow, \mathsf{R}\downarrow\rangle$ where $x\,R\,y \Leftrightarrow x \in f\{y\}$. Exercise: $\mathsf{f} \dashv \mathsf{g}$, between powersets, if and only if what?

– Kan Extensions — for the categorically inclined, Hinze (2012).

- **Hoare Triples:** For relation R, take

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

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

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

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

$$\{\mathsf{P}\}\ \mathsf{S}\ \{\mathsf{Q}\} \Leftrightarrow \mathsf{P} \subseteq \mathsf{wp.S.Q}$$

  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, $\mathsf{L} \dashv \mathsf{U}$.

2.4. 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)
```

2.5. 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. 2.3.

```
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 PosetJoinI 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 PosetJoinI A ⊔f in
  record {bound = L-monotone ∘ bound; universal = λ {y} Lg⊑y → (gc˘ ∘ universal) (gc ∘ Lg⊑y)}
```

Interdefiniability

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

```
_≤U⁻¹ : (x : A₀) → Poset (k ⊍ i′) j′ k′
x ≤U⁻¹ = subPoset B (λ y → x ≤ U y)
L-as-⊓ : {x : A₀} → IsMeetI (x ≤U⁻¹) (λ e → e) (L x, ≤-can)
L-as-⊓ {x} = record {bound = gc˘ ∘ proj₂; universal = λ {y, x≤Uy} y⊑id → y⊑id (L x, ≤-can)}
```

That is, ∀ 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 : Monotone 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
```

2.6. 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)
  )
```

2.7. Conclusion

The proofs 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. Appendix A is included for a quick reference of the aforementioned Galois connection properties.

We will use these proofs 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. However, certain functionals will form Galois connections and so familiarity with the fundamentals of this chapter is of integral import in the remainder of the thesis. In particular, these notions will be used when discussing bounds in Chapter 4.

So much for the theory of pointwise Galois connections.

3. Choosing Our Operations

In this cursory chapter, we attempt to motivate our relational operators from the familiar domain of the predicate calculus. Subsequently we use these to express what it means to be a poset in a pointless fashion.

3.1. Eliminating Quantifiers

Recall that the supremum of a set S is the element s precisely when it is an upper bound of all of S and the least such upper bound:

$$(\forall e \mid e \in S \bullet e \le s) \land (\forall u \mid (\forall e \mid e \in S \bullet e \le u) \bullet s \le u)$$

Needless to say, this can be expressed at the level of sets with meets, as is done in Schmidt and Ströhlein (1993). However, in Calculational Mathematics, the notion of equivalence is considered the fundamental logical connective — even more primitive than implication! An implication is defined in terms of equivalence; (Gries and Schneider, 1993). Hence, a characterisation whose main operator is an equivalence is commonly used:

$$\forall u \bullet u \le s \equiv (\forall e \mid e \in S \bullet e \le u)$$

It's not too difficult to show that this is equivalent to the above definition. Moreover, due to indirect equality, see below, this equation uniquely characterises the supremum. Hence we are in need of a relational counterpart for equivalence.

As another motivation, recall, from Sect. 2.1, proof techniques of indirect inclusion — the celebrated Yoneda Lemma in the setting of posets —

$$\forall \mathsf{x}, \mathsf{y} \bullet \mathsf{x} \preccurlyeq \mathsf{y} \Leftrightarrow (\forall \mathsf{z} \bullet \mathsf{y} \preccurlyeq \mathsf{z} \Rightarrow \mathsf{x} \preccurlyeq \mathsf{z})$$

and if in addition we have a partial order, then we also have indirect equality

$$\forall \mathsf{x}, \mathsf{y} \bullet \mathsf{x} \approx \mathsf{y} \Leftrightarrow (\forall \mathsf{z} \bullet \mathsf{z} \preccurlyeq \mathsf{x} \Leftrightarrow \mathsf{z} \preccurlyeq \mathsf{y})$$

and, of course, variations thereof. Hence, we are in need of expressing universal quantifications relationally as well.

The reader may pause to note that $\le$ is a preorder precisely when it satisfies the law of indirect inclusion. What can be said of the other law?

3.2. Primitive Operations

In order to present a point-free approach to the theory, we must be able to express existential quantifiers, universal quantifiers, and universally quantified equivalences in a point-free fashion. In relational algebra, these take the form of *relation composition, residuals, symmetric quotients, converse,* and *identity*, respectively:

$$x \,(R\,\breve{}\,)\, y \Leftrightarrow y\,R\,x$$

$$x\,\mathsf{Id}\,y \Leftrightarrow x \approx y$$

$$R \approx S \Leftrightarrow (\forall\, x, y \bullet x\,R\,y \Leftrightarrow x\,S\,y\rangle$$

$$x\,(R \fatsemi S)\,y \Leftrightarrow \langle \exists\, z \bullet x\,R\,z \wedge z\,S\,y\rangle$$

$$x\,(R \setminus S)\,y \Leftrightarrow \langle \forall\, z \bullet z\,R\,x \Rightarrow z\,S\,y\rangle$$

$$x\,(R \,/\, S)\,y \Leftrightarrow \langle \forall\, z \bullet x\,R\,z \Rightarrow y\,S\,z\rangle$$

$$x\,(R \between S)\,y \Leftrightarrow \langle \forall\, z \bullet z\,R\,x \Leftrightarrow z\,S\,y\rangle$$

Note that the equality symbol $\_\approx\_$ is overloaded, as in traditional mathematics.

A detailed survey on residuals and symmetric quotients (syqs) can be found in Furusawa and Kahl (1998); while a quick reference can be found in Appendices C and D. Though a key point about division is that it can be uniquely characterised without reference to points,

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

$$\forall\, Q, R, S \bullet Q \sqsubseteq R \between S \Leftrightarrow R \fatsemi Q \sqsubseteq S \wedge Q \fatsemi S\,\breve{}\, \sqsubseteq R\,\breve{}$$

Furthermore, observe that the residuals are instances of external Galois connections. Can the same thing be said of syqs? What of the relation of syqs and relational equality?

We leave these questions to the adamant reader and turn our attention elsewhere. Namely, we show how pointfree is not pointless, but rather is elegant.

- indirect equality

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

  $\Leftrightarrow \mathsf{Id} \;=\; \preccurlyeq \between \preccurlyeq$

  Nested quantifications disappear.

- $\preccurlyeq$ preorder

  $\Leftrightarrow (\forall\, x \bullet x \preccurlyeq x)$ and $(\forall\, x, y, z \bullet x \preccurlyeq y \wedge y \preccurlyeq z \Rightarrow x \preccurlyeq z)$

  $\Leftrightarrow \mathsf{Id} \sqsubseteq \preccurlyeq$ and $\preccurlyeq \fatsemi \preccurlyeq \sqsubseteq \preccurlyeq$

  $\Leftrightarrow \preccurlyeq \;=\; \preccurlyeq \setminus \preccurlyeq$ ($\Leftrightarrow$ indirect inclusion)

  Two universal quantifiers drop down to a single clear equation.

- $\approx$ equivalence

  $\Leftrightarrow \approx$ symmetric, reflexive, and transitive

  $\Leftrightarrow \approx \;=\; \approx \between \approx$

  Three conditions become one.

Hence, it seems that composition, residuals, and quotients suffice for preoders. Then what of orders?

3.3. Expressing Antisymmetry

The antisymmetric law

$$\forall\, x, y \bullet x \preccurlyeq y \wedge y \preccurlyeq x \Rightarrow x \approx y$$

is usually formalised with meets as

$$\preccurlyeq \sqcap \preccurlyeq \;\sqsubseteq\; \mathsf{Id}$$

However, with the above observations we may calculate,

$$
\begin{array}{l}
\quad \preccurlyeq \text{ antisymmetric} \\
\Leftrightarrow \langle \text{ definition } \rangle \\
\quad \forall\, x, y \bullet x \preccurlyeq y \wedge y \preccurlyeq z \Rightarrow x \approx y \\
\Leftrightarrow \langle \text{ indirect inclusion } \rangle \\
\quad \forall\, x, y \bullet (\forall\, z \bullet z \preccurlyeq x \Rightarrow z \preccurlyeq y) \wedge (\forall\, z \bullet z \preccurlyeq y \Rightarrow z \preccurlyeq x) \Rightarrow x \approx y \\
\Leftrightarrow \langle \text{ quantifiers } \rangle \\
\quad \forall\, x, y \bullet (\forall\, z \bullet (z \preccurlyeq x \Rightarrow z \preccurlyeq y) \wedge (z \preccurlyeq y \Rightarrow z \preccurlyeq x)) \Rightarrow x \approx y \\
\Leftrightarrow \langle \text{ bi-implication } \rangle \\
\quad \forall\, x, y \bullet (\forall\, z \bullet z \preccurlyeq x \Leftrightarrow z \preccurlyeq y) \Rightarrow x \approx y \\
\Leftrightarrow \langle \text{ symmetric quotient and containment } \rangle \\
\quad \preccurlyeq \between \preccurlyeq \;\sqsubseteq\; \mathsf{Id}
\end{array}
$$

Hence, even antisymmetry can be formalised within our setting, without the need for meets! In fact, notice that antisymmetry is in fact indirect equality since $\forall\, R \bullet \mathsf{Id} \sqsubseteq R \between R$ — i.e. equivalence is reflexive, even at the relational level.

3.4. Conclusion

Therefore, it appears that locally ordered categories with residuals and symmetric quotients suffices to display a working theory of internal orders. However the notion of converse will crop-up in the need to define other concepts, such as Galois connections.

For those unfamiliar with our setting, Appendix B is provided. There we review what constitutes an ordered category with converse in a record presentation of **field**s and ensuing results. In particular, there is recorded the internalisation of external properties, such as totality and univalence, that the reader is accustomed to. The layout is insouciant as there is a modest level of detail — the material in whole is from Wolfram Kahl's RATH library (Kahl, 2011, 2014b), which is freely accessible and so we shan't waste space reproducing his exposition in detail. Rather the diligent reader who desires true comprehension would do well to prove the results directly.

4. Internal Preorder Theory

Let us begin with formalising preorders in ordered semigroupoids with converse. Due to the lack of identities, identity-like properties are handled with the notions of being a super- and a sub-identity. Namely, p is a sub-identity precisely when ∀ {R S} → R ⨾ p ⨾ S ⊑ R ⨾ S, and dually for super-identities — more can be found in Appendix B.5.

After the main definition, assuming residuals we consider bounds and in particular that the they form an external antitone Galois connection. Then we consider what added power we obtain by including identities as well. Next we turn to closure operators and Galois connections within the setting of preorders. Finally, with the added assumption of symmetric quotients, we investigate the behaviour of extrema in the internal setting.

Many proofs in the following (individually acknowledged) were first proved by Wolfram Kahl in the OCC setting. The explorations of how much of this development can already be done in an OSGC setting is entirely my contribution.

4.1. Categoric.OSGC.Preorder

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

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

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

4.1.1. The Dual Preorder

The converse of a preorder is again a preorder,

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

```
      (E ⨾ E) ˘
    ⊑⟨ ˘-monotone trans ⟩
      E ˘
    □

idempot : IsIdempotent E
idempot = ⊑-antisym trans rightSupId

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

More explicitly,

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

4.1.2. Indirect inclusion

The heuristic of indirect inclusion, as presented in Sect. 3.1, can also be moved over, though generalizing from 'points', functions to a terminal object, to arbitrary functions f, g as

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

In fact, a total and univalent pair suffice:

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

4.1.3. 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); in-fact we must acknowledge that these particular proofs, among others to follows, are due to Wolfram Kahl.

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

Useful combinators:

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

The 'cones' and 'closures' of bounds:

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

```
          ≈⟨⟩
            (Q ⨾ E ˘) ˘ \ E
          ≈⟨ \-cong₁ ˘-involutionRightConv ⟩
            (E ⨾ Q ˘) \ E
          ≈⟨ ubd-downcone₀ ⟩
            ubd Q
          □

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

The relation between mappings and bounds:

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

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

⨾order-⊑-ubd-→ : {I : Obj} {Q R : Mor I A} → R ⨾ E ⊑ ubd Q → R ⊑ ubd Q
⨾order-⊑-ubd-→ {I} {Q} {R} R⨾E⊑ubdQ
        = \-universal (⨾-monotone₂ rightSupId ⟨⊑⊑⟩ \-universal′ R⨾E⊑ubdQ)
⨾order-⊑-ubd-← : {I : Obj} {Q R : Mor I A} → R ⊑ ubd Q → R ⨾ E ⊑ ubd Q
⨾order-⊑-ubd-← {I} {Q} {R} R⊑ubdQ = ⨾-monotone₁ R⊑ubdQ ⟨⊑≈⟩ ubd-upclosed

order-\ : E \ E ≈ E
order-\ = ⊑-antisym (⊑-begin
```

```
    E \ E
  ⊑⟨ proj₁ supId ⟩
    E ⨾ (E \ E)
  ⊑⟨ \-cancel-outer ⟩
    E
  □) (\-universal trans)
order-/ : E / E ≈ E
order-/ = ⊑-antisym (proj₂ supId ⟨⊑⊑⟩ /-cancel-outer) (/-universal trans)
```

Compare these results with indirect-E above, and with the formulations presented in Sect. 3.1.

An immediate nifty consequence,

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

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. The following seven direct proofs are all due to Wolfram Kahl.

```
  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, Wolfram Kahl thought to prove them and compare the resulting sizes of the 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
    □
```

4.2. Categoric.OCC.Preorder

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

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


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

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

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

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

```
˘-refl : IsReflexive (E ˘)
˘-refl = ⊑-begin
    Id
  ≈˘⟨ Id˘ ⟩
    Id ˘
  ⊑⟨ ˘-monotone refl ⟩
    E ˘
  □
˘-isPreorder : IsPreorder (E ˘)
˘-isPreorder = record {refl = ˘-refl; trans = ˘-trans}
```

4.2.1. Retract Preorder and Preorder Invaraince

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

```
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
```

4.2.2. 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— a Wolfram Kahl derivation — :

```
    \-isPreorder : {A B : Obj} (R : Mor A B) → IsPreorder (R \ R)
    \-isPreorder R = record {refl = \-isReflexive; trans = \-cancel-middle}
```

```
\-isPreorder₀ : {A B : Obj} (R : Mor A B) → IsPreorder₀ osgc (R \ R)
\-isPreorder₀ R = record
    {supId = reflexiveIsSuperidentity \-isReflexive
    ; trans = \-cancel-middle
    }
module \-Preorder {A B : Obj} (R : Mor A B) where
    E : Mor B B
    E = R \ R
    isPreorder : IsPreorder E
    isPreorder = \-isPreorder R
    open IsPreorder′ isPreorder public
    ubdE≈\ : {I : Obj} {Q : Mor I B} → ubd Q ≈ (R ⨾ Q ˘) \ R
    ubdE≈\ {I} {Q} = ≈-begin
        Q ˘ \ (R \ R)
        ≈⟨ \\ ⟩
          (R ⨾ Q ˘) \ R
        □
    ubd˘E≈/ : {I : Obj} {Q : Mor I B} → ubd Q ˘ ≈ R ˘ / (Q ⨾ R ˘)
    ubd˘E≈/ {I} {Q} = ≈-begin
          ubd Q ˘
        ≈⟨ ˘-cong ubdE≈\ ⟩
          ((R ⨾ Q ˘) \ R) ˘
        ≈⟨ \-˘ ⟩
          R ˘ / (R ⨾ Q ˘) ˘
        ≈⟨ /-cong₂ ˘-involutionRightConv ⟩
          R ˘ / (Q ⨾ R ˘)
        □
    lbd-ubd-≈-twist : {I : Obj} {Q : Mor I B} → lbd (ubd Q) ≈ (R ˘ / (Q ⨾ R ˘)) \ (R ˘ / R ˘)
    lbd-ubd-≈-twist {I} {Q} = ≈-begin
          lbd (ubd Q)
        ≈⟨ \-cong₂ \-˘ ⟩
          (ubd Q) ˘ \ (R ˘ / R ˘)
        ≈⟨ \-cong₁ ubd˘E≈/ ⟩
          (R ˘ / (Q ⨾ R ˘)) \ (R ˘ / R ˘)
        □
```

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

4.3. Categoric.OSGC.Preorder.Closure

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

$$\forall\, x, y \bullet x \preccurlyeq C\,(x) \;\Leftrightarrow\; C\,(x) \preccurlyeq C\,(y)$$

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

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

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

We use the name char as an abbreviation of characterisation.

4.3.1. Increasing

Before showing that the closure operator is increasing, let us observe that both sides of the characterization are super identities— see Appendix B.5 for the concept of super-identities.

```
        CEC˘-supId : isSuperidentity (C ⨾ E ⨾ C ˘)
        CEC˘-supId = (λ {B} {R} → ⊑-begin
                  R
              ⊑⟨ proj₁ C.total ⟩
                  (C ⨾ C ˘) ⨾ R
              ⊑⟨ ⨾-monotone₁₂ A.leftSupId ⟩
```

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

Consequently, we have the combinators,

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

4.3.2. Quasi-idempodentcy

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

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

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

4.3.3. Monotonicity

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

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

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

```
comonotone˘ : C ˘ ⨾ E ⊑ E ⨾ C ˘
comonotone˘ = ⊑-begin
        C ˘ ⨾ E
    ⊑⟨ ⨾-monotone₂ (proj₂ EC˘-supId) ⟩
        C ˘ ⨾ E ⨾ E ⨾ C ˘
    ≈⟨ ⨾-cong₂ (⨾-assoc ⟨≈˘≈⟩ ⨾-cong₁ A.idempot) ⟩
        C ˘ ⨾ E ⨾ C ˘
```

```
    ≈⟨ ⨾-cong₂ char ⟩
        C ˘ ⨾ C ⨾ E ⨾ C ˘
    ≈⟨ ⨾-cong₂ (⨾-assoc ⟨≈˘≈˘⟩ ⨾-cong₁ idempE) ⟩
        C ˘ ⨾ (C ⨾ C ⨾ E) ⨾ C ˘
    ≈⟨ ⨾-cong₂ ⨾-assoc₃₊₁ ⟨≈≈⟩ ⨾-assocL ⟩
        (C ˘ ⨾ C) ⨾ C ⨾ E ⨾ C ˘
    ⊑⟨ proj₁ C.unival ⟩
        C ⨾ E ⨾ C ˘
    ≈˘⟨ char ⟩
        E ⨾ C ˘
    □

comonotone : E ˘ ⨾ C ⊑ C ⨾ E ˘
comonotone = ⊑-begin
        E ˘ ⨾ C
  ≈˘⟨ ˘-involutionLeftConv ⟩
        (C ˘ ⨾ E) ˘
  ⊑⟨ ˘-monotone comonotone˘ ⟩
        (E ⨾ C ˘) ˘
  ≈⟨ ˘-involutionRightConv ⟩
        C ⨾ E ˘
    □

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


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

Furthermore, we have a peculiar result:

```
CE-⊑-EC˘ : C ⨾ E ⊑ E ⨾ C ˘
CE-⊑-EC˘ = ⊑-begin
```

```
    C ⨾ E
⊑⟨ ⨾-monotone₂ expansion ⟩
    C ⨾ E ⨾ C ˘
≈˘⟨ char ⟩
    E ⨾ C ˘
□
```

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

4.3.4. Piecewise Closure Characterization

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

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

4.3.5. Dually: Interior Operator

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

```
record PreCoclosureOp {A : Obj} {E : Mor A A}
      (A-isPreorder : IsPreorder osgc E) (CC : Mapping A A) : Set k₁ where
```

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

Dually, interior operators have an equivalent piecewise formulation.

```
module _ {A : Obj} {E : Mor A A} (A-isPreorder : IsPreorder osgc E) (CC : Mapping A A)
    where
    private
        module A = IsPreorder osgc A-isPreorder
        module C = Mapping CC
    open C using () renaming (mor to C)
```

```
piecewise-to-interior : (decreasing : C ⊑ E ˘) (idemp : C ⨾ C ≈ C) (monotone : E ⨾ C ⊑ C ⨾ E)
                          → PreCoclosureOp {A} {E} A-isPreorder CC
piecewise-to-interior decreasing idemp monotone = record {char = ⊑-antisym (⊑-begin
        C ⨾ E
    ⊑⟨ ⨾-monotone₂ (proj₂ C.total) ⟩
        C ⨾ E ⨾ C ⨾ C ˘
    ⊑⟨ ⨾-monotone₂ (⨾-assoc ⟨≈˘⊑⟩ (⨾-monotone₁ monotone ⟨⊑≈⟩ ⨾-assoc)) ⟩
        C ⨾ C ⨾ E ⨾ C ˘
    ≈⟨ ⨾-assoc₄ ⟨≈˘≈⟩ (⨾-cong₁₁ idemp ⟨≈≈⟩ ⨾-assoc) ⟩
        C ⨾ E ⨾ C ˘
    □)(⊑-begin
        C ⨾ E ⨾ C ˘
    ⊑⟨ ⨾-monotone₂₂ (˘-monotone decreasing ⟨⊑≈⟩ ˘˘) ⟩
        C ⨾ E ⨾ E
    ⊑⟨ ⨾-monotone₂ A.trans ⟩
        C ⨾ E
    □)}
```

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

4.3.6. Conclusion

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

4.4. Categoric.OSGC.Preorder.Galois

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

```
record PreGaloisConnection {A₁ A₂ : Obj} {E₁ : Mor A₁ A₁} {E₂ : Mor A₂ A₂}
  (A₁-isPreorder : IsPreorder osgc E₁) (A₂-isPreorder : IsPreorder osgc E₂)
  (LL : Mapping A₁ A₂) (UU : Mapping A₂ A₁) : Set k₁ where
    private
      module A₁ = IsPreorder osgc A₁-isPreorder
      module A₂ = IsPreorder osgc A₂-isPreorder
      module L = Mapping LL
      module U = Mapping UU
```

```
open L using () renaming (mor to L)
open U using () renaming (mor to U)
field gc : L ⨾ E₂ ≈ E₁ ⨾ U ˘
```

4.4.1. Co-connection

For convenience, we name the converse orders.

```
∃₁ : Mor A₁ A₁
∃₁ = E₁ ˘
∃₂ : Mor A₂ A₂
∃₂ = E₂ ˘
```

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 ⨾ ∃₁ ≈ ∃₂ ⨾ L ˘
gc˘ = ≈-sym (˘-involution ⟨≈˘≈⟩ (˘-cong gc ⟨≈≈⟩ ˘-involutionRightConv))
```

4.4.2. 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˘-⊑-∃ : U ˘ ⨾ L ˘ ⊑ ∃₁
U˘⨾L˘-⊑-∃ = ˘-involution ⟨≈˘⊑⟩ (˘-⊑-swap (LU-⊑-E ⟨⊑≈˘⟩ ˘˘))
EU˘L˘-supId : isSuperidentity (E₁ ⨾ U ˘ ⨾ L ˘)
EU˘L˘-supId = (λ {B} {R} → ⊑-begin
        R
    ⊑⟨ proj₁ L.total ⟩
        (L ⨾ L ˘) ⨾ R
    ⊑⟨ ⨾-monotone₁₁ A₂.rightSupId ⟩
        ((L ⨾ E₂) ⨾ L ˘) ⨾ R
    ≈⟨ ⨾-cong₁ (⨾-cong₁ gc ⟨≈≈⟩ ⨾-assoc) ⟩
        (E₁ ⨾ U ˘ ⨾ L ˘) ⨾ R
    □),
    (λ {B} {S} → swap-⨾-⊑-total L.total
        (⨾-monotone ⊑-refl L-⊑-E⨾U˘) ⟨⊑≈⟩ (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ⨾-assoc))
```

```
LU∃-supId : isSuperidentity (L ⨾ U ⨾ ∃₁)
LU∃-supId = (λ {B} {R} → ⊑-begin
          R
      ≈˘⟨ ˘˘ ⟩
          R ˘ ˘
      ⊑⟨ ˘-monotone (proj₂ EU˘L˘-supId) ⟩
        (R ˘ ⨾ E₁ ⨾ U ˘ ⨾ L ˘) ˘
      ≈⟨ ˘-cong (⨾-cong₂ (⨾-assoc ⟨≈˘≈˘⟩ ⨾-cong₁ ˘-involutionRightConv)) ⟩
        (R ˘ ⨾ (U ⨾ E₁ ˘) ˘ ⨾ L ˘) ˘
      ≈˘⟨ ˘-cong (⨾-cong₂ ˘-involution) ⟩
        (R ˘ ⨾ (L ⨾ U ⨾ E₁ ˘) ˘) ˘
      ≈⟨ ˘-coinvolution ⟩
          (L ⨾ U ⨾ E₁ ˘) ⨾ R
      □), (λ {B} {S} → ⊑-begin
          S
      ≈˘⟨ ˘˘ ⟩
          S ˘ ˘
      ⊑⟨ ˘-monotone (proj₁ EU˘L˘-supId) ⟩
        ((E₁ ⨾ U ˘ ⨾ L ˘) ⨾ S ˘) ˘
      ≈⟨       ˘-cong (⨾-cong₁ (⨾-assoc ⟨≈˘≈˘⟩ ⨾-cong₁ ˘-involutionRightConv)) ⟩
        (((U ⨾ E₁ ˘) ˘ ⨾ L ˘) ⨾ S ˘) ˘
      ≈˘⟨ ˘-cong (⨾-cong₁ ˘-involution) ⟩
        ((L ⨾ U ⨾ E₁ ˘) ˘ ⨾ S ˘) ˘
      ≈⟨ ˘-coinvolution ⟩
          S ⨾ (L ⨾ U ⨾ E₁ ˘)
      □)
```

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

```
U-⊑-∃⨾L˘ : U ⊑ ∃₂ ⨾ L ˘
U-⊑-∃⨾L˘ = A₁.˘-rightSupId ⟨⊑≈⟩ gc˘

UL-⊑-∃ : U ⨾ L ⊑ ∃₂
UL-⊑-∃ = swap-⊑-⨾-unival˘ L.unival U-⊑-∃⨾L˘

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

∃L˘U˘-supId : isSuperidentity (∃₂ ⨾ L ˘ ⨾ U ˘)
∃L˘U˘-supId = (λ {B} {R} → ⊑-begin
          R
        ⊑⟨ proj₁ U.total ⟩
          (U ⨾ U ˘) ⨾ R
      ⊑⟨ ⨾-monotone₁₂ A₁.˘-leftSupId ⟩
          (U ⨾ ∃₁ ⨾ U ˘) ⨾ R
      ≈⟨ ⨾-cong₁ (⨾-assoc ⟨≈˘≈⟩ ⨾-cong₁ gc˘ ⟨≈≈⟩ ⨾-assoc) ⟩
          (∃₂ ⨾ L ˘ ⨾ U ˘) ⨾ R
      □),
   (λ {B} {S} → swap-⨾-⊑-total U.total (⨾-monotone ⊑-refl U-⊑-∃⨾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)
    □)
```

4.4.3. 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 ˘ ⨾ ∃₁ ⊑ ∃₂ ⨾ L ˘
L-monotone˘ = ˘˘ ⟨≈˘⊑⟩
    (˘-monotone (˘-coinvolution ⟨≈⊑⟩ (L-monotone ⟨⊑≈˘⟩ ˘-coinvolution)) ⟨⊑≈⟩ ˘˘)
L-comonotone : ∃₁ ⨾ L ⊑ L ⨾ ∃₂
L-comonotone = swap-⨾-⊑-total˘ L.total
    (⨾-assoc ⟨≈˘⊑⟩ swap-⊑-⨾-unival˘ L.unival L-monotone˘)
L-comonotone˘ : L ˘ ⨾ E₁ ⊑ E₂ ⨾ L ˘
L-comonotone˘ = ˘-involutionLeftConv
    ⟨≈˘⊑⟩ ˘-monotone L-comonotone ⟨⊑≈⟩ ˘-involutionRightConv
U-comonotone : ∃₂ ⨾ U ⊑ U ⨾ ∃₁
U-comonotone = ⊑-begin
            ∃₂ ⨾ U
```

```
          ⊑⟨ ⨾-monotone₂ (proj₁ ∃L˘U˘-supId) ⟩
            ∃₂ ⨾ (∃₂ ⨾ L ˘ ⨾ U ˘) ⨾ U
          ≈⟨ ⨾-₂₂assoc₁₂₁ ⟨≈˘≈⟩ (⨾-cong₂ ⨾-assoc ⟨≈≈⟩ ⨾-cong₁ A₂.˘-idempot) ⟩
            ∃₂ ⨾ L ˘ ⨾ U ˘ ⨾ U
          ⊑⟨ ⨾-monotone₂ (proj₂ U.unival) ⟩
            ∃₂ ⨾ L ˘
          ≈˘⟨ gc˘ ⟩
            U ⨾ ∃₁
          □

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

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

U-monotone˘ : U ˘ ⨾ ∃₂ ⊑ ∃₁ ⨾ U ˘
U-monotone˘ = ˘˘ ⟨≈˘⊑⟩
    (˘-monotone (˘-coinvolution ⟨≈⊑⟩ (U-monotone ⟨⊑≈˘⟩ ˘-coinvolution)) ⟨⊑≈⟩ ˘˘)
```

4.4.4. 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
  □)
```

```
ULU∃≈U∃ : (U ⨾ L ⨾ U) ⨾ ∃₁ ≈ U ⨾ ∃₁
ULU∃≈U∃ = ⊑-antisym (⊑-begin
        (U ⨾ L ⨾ U) ⨾ ∃₁
    ⊑⟨ ⨾-monotone₁ (⨾-assoc ⟨≈˘⊑⟩ ⨾-monotone₁ UL-⊑-∃) ⟩
        (∃₂ ⨾ U) ⨾ ∃₁
    ⊑⟨ ⨾-monotone₁ U-comonotone ⟨⊑≈⟩ ⨾-assoc ⟩
        U ⨾ E₁ ˘ ⨾ ∃₁
    ⊑⟨     ⨾-monotone₂ A₁.˘-trans ⟩
        U ⨾ ∃₁
    □) (⊑-begin
        U ⨾ ∃₁
    ⊑⟨ ⨾-monotone₂ (proj₁ L.total ⟨⊑≈⟩ ⨾-assoc) ⟩
        U ⨾ L ⨾ L ˘ ⨾ ∃₁
    ≈˘⟨ ⨾-cong₂₂₂ A₁.˘-idempot ⟩
        U ⨾ L ⨾ L ˘ ⨾ ∃₁ ⨾ ∃₁
    ⊑⟨ ⨾-monotone₂₂ (⨾-assoc ⟨≈˘⊑⟩ ⨾-monotone₁ L-monotone˘ ⟨⊑≈⟩ ⨾-assoc) ⟩
        U ⨾ L ⨾ ∃₂ ⨾ L ˘ ⨾ ∃₁
    ≈⟨ ⨾-cong₂ ⨾-assocL₃₊₁ ⟨≈≈˘⟩ ⨾-cong₂₁₂ gc˘ ⟩
        U ⨾ (L ⨾ U ⨾ ∃₁) ⨾ ∃₁
    ≈⟨ ⨾-cong₂ (⨾-assoc₃₊₁ ⟨≈≈⟩ ⨾-cong₂₂ A₁.˘-idempot) ⟨≈≈⟩ ⨾-assocL₃₊₁ ⟩
        (U ⨾ L ⨾ U) ⨾ ∃₁
    □)
```

4.4.5. 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-abs∃ : {C : Obj} {Q R : Mor C A₂} → R ⨾ U ⨾ L ≈ Q ⨾ U ⨾ L → R ⨾ U ⨾ ∃₁ ≈ Q ⨾ U ⨾ ∃₁
U-abs∃ {C} {Q} {R} RUL≈QUL = ≈-begin
          R ⨾ U ⨾ ∃₁
        ≈⟨ ⨾-cong₂ ULU∃≈U∃ ⟨≈˘≈˘⟩ ⨾-assoc ⟩
          (R ⨾ U ⨾ L ⨾ U) ⨾ ∃₁
        ≈⟨ ⨾-cong₁ (⨾-assoc₃₊₁ ⟨≈˘≈⟩ ⨾-cong₁ RUL≈QUL)
          ⟨≈≈⟩ (⨾-cong₁ ⨾-assoc₃₊₁ ⟨≈≈⟩ (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ⨾-assoc₃₊₁)) ⟩
          Q ⨾ U ⨾ L ⨾ U ⨾ ∃₁
        ≈⟨ ⨾-cong₂ (⨾-assoc₃₊₁ ⟨≈˘≈⟩ ULU∃≈U∃) ⟩
```

```
    Q ⨾ U ⨾ ∃₁
  □
```

4.4.6. Image Isotonicity

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

```
L-isotone-on-U : U ⨾ L ⨾ E₂ ⨾ L ˘ ⨾ U ˘ ≈ U ⨾ E₁ ⨾ U ˘
L-isotone-on-U = ≈-sym (≈-begin
        U ⨾ E₁ ⨾ U ˘
    ≈⟨ ⨾-cong₂₁ A₁.idempot ⟨≈˘≈⟩ ⨾-cong₂ ⨾-assoc ⟩
        U ⨾ E₁ ⨾ E₁ ⨾ U ˘
    ≈˘⟨ ⨾-cong₂₂ ˘-involutionRightConv ⟩
        U ⨾ E₁ ⨾ (U ⨾ ∃₁) ˘
    ≈˘⟨ ⨾-cong₂₂ (˘-cong ULU∃≈U∃) ⟩
        U ⨾ E₁ ⨾ ((U ⨾ L ⨾ U) ⨾ ∃₁) ˘
    ≈⟨ ⨾-cong₂₂ ˘-involutionRightConv ⟩
        U ⨾ E₁ ⨾ E₁ ⨾ (U ⨾ L ⨾ U) ˘
    ≈⟨ ⨾-cong₂ (⨾-assoc ⟨≈˘≈⟩ (⨾-cong₁ A₁.idempot ⟨≈≈⟩ ⨾-cong₂ ˘-involution)) ⟩
        U ⨾ E₁ ⨾ ((L ⨾ U) ˘ ⨾ U ˘)
    ≈⟨ ⨾-cong₂₂ (⨾-cong₁ ˘-involution ⟨≈≈⟩ ⨾-assoc) ⟩
        U ⨾ E₁ ⨾ (U ˘ ⨾ L ˘ ⨾ U ˘)
    ≈˘⟨ ⨾-cong₂ ⨾-assoc ⟩
        U ⨾ (E₁ ⨾ U ˘) ⨾ L ˘ ⨾ U ˘
    ≈˘⟨ ⨾-cong₂₁ gc ⟩
        U ⨾ (L ⨾ E₂) ⨾ L ˘ ⨾ U ˘
    ≈⟨ ⨾-cong₂ ⨾-assoc ⟩
        U ⨾ L ⨾ E₂ ⨾ L ˘ ⨾ U ˘
    □)
L-coisotone-on-U : U ⨾ L ⨾ ∃₂ ⨾ L ˘ ⨾ U ˘ ≈ U ⨾ ∃₁ ⨾ U ˘
L-coisotone-on-U = ≈-begin
        U ⨾ L ⨾ ∃₂ ⨾ L ˘ ⨾ U ˘
    ≈˘⟨ ⨾-cong₁ ⨾-assoc ⟨≈≈⟩ (⨾-assoc ⟨≈≈⟩ ⨾-cong₂ ⨾-assoc) ⟩
        ((U ⨾ L) ⨾ ∃₂) ⨾ L ˘ ⨾ U ˘
    ≈˘⟨ ⨾-cong₁ (˘-involution ⟨≈≈⟩ ⨾-cong₁ ˘-coinvolution) ⟩
        (E₂ ⨾ L ˘ ⨾ U ˘) ˘ ⨾ L ˘ ⨾ U ˘
    ≈˘⟨ ˘-involution ⟨≈≈⟩ ⨾-cong₁ ˘-involution ⟨≈≈⟩ ⨾-assoc ⟩
        (U ⨾ L ⨾ E₂ ⨾ L ˘ ⨾ U ˘) ˘
    ≈⟨ ˘-cong L-isotone-on-U ⟩
        (U ⨾ E₁ ⨾ U ˘) ˘
    ≈⟨ ˘-involution ⟨≈≈⟩ (⨾-cong₁ ˘-involutionRightConv ⟨≈≈⟩ ⨾-assoc) ⟩
        U ⨾ ∃₁ ⨾ U ˘
    □
U-coisotone-on-L : L ⨾ U ⨾ ∃₁ ⨾ U ˘ ⨾ L ˘ ≈ L ⨾ ∃₂ ⨾ L ˘
U-coisotone-on-L = ≈-sym (≈-begin
        L ⨾ ∃₂ ⨾ L ˘
```

≈⟨ ⨾-cong$_{21}$ A$_2$.˘-idempot ⟨≈˘≈⟩ ⨾-cong$_2$ ⨾-assoc ⟩
L ⨾ $\exists_2$ ⨾ $\exists_2$ ⨾ L ˘
≈˘⟨ ⨾-cong$_{22}$ ˘-involution ⟩
L ⨾ $\exists_2$ ⨾ (L ⨾ E$_2$) ˘
≈˘⟨ ⨾-cong$_{22}$ (˘-cong LULE≈LE) ⟩
L ⨾ $\exists_2$ ⨾ ((L ⨾ U ⨾ L) ⨾ E$_2$) ˘
≈⟨ ⨾-cong$_{22}$ ˘-involution ⟩
L ⨾ $\exists_2$ ⨾ $\exists_2$ ⨾ (L ⨾ U ⨾ L) ˘
≈⟨ ⨾-cong$_2$ (⨾-assoc ⟨≈˘≈⟩ (⨾-cong$_1$ A$_2$.˘-idempot ⟨≈≈⟩ ⨾-cong$_2$ ˘-involution)) ⟩
L ⨾ $\exists_2$ ⨾ ((U ⨾ L) ˘ ⨾ L ˘)
≈⟨ ⨾-cong$_{22}$ (⨾-cong$_1$ ˘-involution ⟨≈≈⟩ ⨾-assoc) ⟩
L ⨾ $\exists_2$ ⨾ (L ˘ ⨾ U ˘ ⨾ L ˘)
≈˘⟨ ⨾-cong$_2$ ⨾-assoc ⟩
L ⨾ ($\exists_2$ ⨾ L ˘) ⨾ U ˘ ⨾ L ˘
≈˘⟨ ⨾-cong$_{21}$ gc˘ ⟩
L ⨾ (U ⨾ $\exists_1$) ⨾ U ˘ ⨾ L ˘
≈⟨ ⨾-cong$_2$ ⨾-assoc ⟩
L ⨾ U ⨾ $\exists_1$ ⨾ U ˘ ⨾ L ˘
□)

U-isotone-on-L : L ⨾ U ⨾ E$_1$ ⨾ U ˘ ⨾ L ˘ ≈ L ⨾ E$_2$ ⨾ L ˘
U-isotone-on-L = ≈-begin
L ⨾ U ⨾ E$_1$ ⨾ U ˘ ⨾ L ˘
≈˘⟨ ⨾-cong$_1$ ⨾-assoc ⟨≈≈⟩ (⨾-assoc ⟨≈≈⟩ ⨾-cong$_2$ ⨾-assoc) ⟩
((L ⨾ U) ⨾ E$_1$) ⨾ U ˘ ⨾ L ˘
≈˘⟨ ⨾-cong$_1$ (˘-involutionLeftConv ⟨≈≈⟩ ⨾-cong$_1$ ˘-coinvolution) ⟩
($\exists_1$ ⨾ U ˘ ⨾ L ˘) ˘ ⨾ U ˘ ⨾ L ˘
≈˘⟨ ˘-involution ⟨≈≈⟩ ⨾-cong$_1$ ˘-involution ⟨≈≈⟩ ⨾-assoc ⟩
(L ⨾ U ⨾ $\exists_1$ ⨾ U ˘ ⨾ L ˘) ˘
≈⟨ ˘-cong U-coisotone-on-L ⟩
(L ⨾ $\exists_2$ ⨾ L ˘) ˘
≈⟨ ˘-involution ⟨≈≈⟩ ⨾-cong$_1$ ˘-coinvolution ⟨≈≈⟩ ⨾-assoc ⟩
L ⨾ E$_2$ ⨾ L ˘
□

4.4.7. Induced Interior

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

interior : (U ⨾ L) ⨾ E$_2$ ⨾ (U ⨾ L) ˘ ≈ (U ⨾ L) ⨾ E$_2$
interior = ≈-begin
(U ⨾ L) ⨾ E$_2$ ⨾ (U ⨾ L) ˘
≈⟨ ⨾-cong$_{22}$ ˘-involution ⟨≈≈⟩ ⨾-assoc ⟩
U ⨾ L ⨾ E$_2$ ⨾ L ˘ ⨾ U ˘
≈⟨ L-isotone-on-U ⟩
U ⨾ E$_1$ ⨾ U ˘
≈⟨ ⨾-cong$_2$ 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) ˘
    ; C∃C˘-supId  to UL⨾∃⨾UL˘-supId          -- isSuperidentity ((U ⨾ L) ⨾ ∃₂ ⨾ (U ⨾ L) ˘)
    ; ∃C˘-supId   to ∃⨾UL˘-supId             -- isSuperidentity (∃₂ ⨾ (U ⨾ L) ˘), cf ₂-can′
    ; decreasing  to UL-decreasing           -- U ⨾ L ⊑ ∃₂. cf ₂-can
    ; contraction to UL-contraction          -- E₂ ⊑ (U ⨾ L) ⨾ E₂
    ; ∃C-⊑-∃      to ∃UL-⊑-∃                 -- ∃₂ ⨾ U ⨾ L ⊑ ∃₂
    ; C∃-⊑-∃      to UL⨾∃-⊑-∃                -- (U ⨾ L) ⨾ ∃₂ ⊑ ∃₂
    ; ∃C˘C-⊑-C∃   to ∃⨾UL⨾UL˘-⊑-UL⨾∃         -- ∃₂ ⨾ (U ⨾ L) ˘ ⨾ (U ⨾ L) ⊑ (U ⨾ L) ⨾ ∃₂
    ; idemp∃      to UL-idempE               -- (U ⨾ L) ⨾ (U ⨾ L) ⨾ ∃₂ ≈ (U ⨾ L) ⨾ ∃₂
    ; C∃-⊑-∃C˘    to UL⨾∃-⊑-∃-UL˘            -- (U ⨾ L) ⨾ ∃₂ ⊑ ∃₂ ⨾ (U ⨾ L) ˘
    ; monotone˘   to UL-monotone˘            -- (U ⨾ L) ˘ ⨾ ∃₂ ⊑ ∃₂ ⨾ (U ⨾ L) ˘
    ; monotone    to UL-monotone             -- E₂ ⨾ (U ⨾ L) ⊑ (U ⨾ L) ⨾ E₂
    ; comonotone to UL-comonotone            -- ∃₂ ⨾ (U ⨾ L) ⊑ (U ⨾ L) ⨾ ∃₂
    ; comonotone˘ to UL-comonotone˘          -- (U ⨾ L) ˘ ⨾ E₂ ⊑ E₂ ⨾ (U ⨾ L) ˘
    )
```

4.4.8. Induced Closure

While the reverse composition yields a closure operator.

```
closure : (L ⨾ U) ⨾ E₁ ⨾ (L ⨾ U) ˘        ≈ E₁ ⨾ (L ⨾ U) ˘
closure = ≈-begin
      (L ⨾ U) ⨾ E₁ ⨾ (L ⨾ U) ˘
    ≈⟨ ⨾-cong₂₂ ˘-involution ⟨≈≈⟩ ⨾-assoc ⟩
      L ⨾ U ⨾ E₁ ⨾ U ˘ ⨾ L ˘
    ≈⟨ U-isotone-on-L ⟩
      L ⨾ E₂ ⨾ L ˘
    ≈⟨ ⨾-assoc ⟨≈˘≈⟩ ⨾-cong₁ gc ⟩
      (E₁ ⨾ U ˘) ⨾ L ˘
    ≈⟨ ⨾-assoc ⟨≈≈˘⟩ ⨾-cong₂ ˘-involution ⟩
      E₁ ⨾ (L ⨾ U) ˘
    □

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

LU₀ : Mor A₁ A₁
LU₀ = Mapping.mor LU
```

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


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

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

4.5. Categoric.OCC.Preorder.Galois

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

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


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

```
open PreGaloisConnection osgc {A₁} {A₂} {E₁} {E₂}
  {A₁.isPreorder₀} {A₂.isPreorder₀} {LL} {UU} (record {gc = gc}) public hiding (gc)
```

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

```
EU˘L˘-refl : Id ⊑ E₁ ⨾ U ˘ ⨾ L ˘
EU˘L˘-refl = swap-⨾-⊑-total L.total (leftId ⟨≈⊑⟩ L-⊑-E⨾U˘) ⟨⊑≈⟩ ⨾-assoc
LU∃-refl : Id ⊑ L ⨾ U ⨾ E₁ ˘
LU∃-refl = Id˘ ⟨≈˘⊑⟩ (˘-monotone EU˘L˘-refl
  ⟨⊑≈⟩ (˘-cong (⨾-cong₂ ˘-involution)
  ⟨≈˘≈⟩ ˘-involutionRightConv ⟨≈≈⟩ ⨾-assoc))
∃L˘U˘-refl : Id ⊑ ∃₂ ⨾ L ˘ ⨾ U ˘
∃L˘U˘-refl = swap-⨾-⊑-total U.total (leftId ⟨≈⊑⟩ U-⊑-∃⨾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 ∃L˘U˘-refl
  ⟨⊑≈⟩ (˘-cong (⨾-cong₂ ˘-involution)
  ⟨≈˘≈⟩ (˘-coinvolution ⟨≈≈⟩ ⨾-assoc)))
```

This is nearly three times larger than the indirect proof.

So much for considering identities.

4.6. Categoric.OSGC.Preorder.Extrema

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

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

```
open ResidualOps                  leftResOp rightResOp
open OSGC-Residuals          osgc leftResOp rightResOp
open PreorderWithResiduals osgc leftResOp rightResOp using (module IsPreorder′)
```

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

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

4.6.1. A Risky Duality

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

Proof term line count:

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

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

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

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

```
private
module gre-glb {A : Obj} {E : Mor A A} (E-isPreorder : IsPreorder osgc E) where
```

```
open IsPreorder osgc E-isPreorder using (˘-idempot)
open IsPreorder′ E-isPreorder public

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

Recall,

glb S = s

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

That is, S glb s ⇔ s (≤ ╳ (≥ / ∋) S, i.e., glb = (≥ / ∋) ╳ ≤ — see Appendix D regarding converses and syqs. Formally,

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

The expected congruence laws,

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

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

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

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

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

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

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

First we obtain some useful lemmas,

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

Now we can obtain our desired result,

```
lub-≈-glb-ubd : {I : Obj} {Q : Mor I A} → lub Q ≈ glb (ubd Q)
lub-≈-glb-ubd {I} {Q} = ≈-begin
    lub Q
  ≈⟨⟩
    ubd Q ˘ ╳ E ˘
  ≈⟨ ⊑-antisym ˘╳˘-⊑-/-╳
    (╳-universal
      ˘-⨾-/-╳-⊑
```

```
    ((⊑-begin
        ((E / ubd Q) ╳ E) ⨾ (E ˘) ˘
      ⊑⟨ ⨾-monotone ╳-⊑-\ (⊑-reflexive ˘˘) ⟩
        ((E / ubd Q) \ E) ⨾ E
      ≈⟨ ⨾-cong₁ \S∘S/∘\S ⟨≈≈⟩ ubd-upclosed ⟩
        ubd Q
      □) ⟨⊑≈˘⟩ ˘˘))
  ⟩
  (E / ubd Q) ╳ E
≈˘⟨ ╳-cong₁ lbd-ubd-˘ ⟩
  lbd (ubd Q) ˘ ╳ E
≈⟨⟩
  glb (ubd Q)
□
```

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

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

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

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

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

4.7. Conclusion

It appears that a great deal can be performed within the setting of an OSGC; as identities provide little gain — except in the case of residuals inducing preorders. Moreover, it seems that, currently, we cannot make efficient use of the dual nature of many a definition due to Agda's compilation system. However, we hope that our indications of such difference, and our evidence so that they may be reproduced, would lead to improvement in this area.

On the other hand, a great deal of the theory of Galois connections and of closure operators has been internalised and, more importantly, presented in an algorithmic fashion, whence practical for programming and proving purposes.

5. Internal Partial Order Theory

This chapter begins by building upon our development by adjoining antisymmetry. We subsequently continue the previous approach of closures and Galois connections. The new antisymmetry axiom gives us the usual results that one encounters when utilising posets — an indication that we are heading in a right direction.

5.1. Categoric.OCC.Order

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

Nearly all proofs, except Sect. 5.1.2 are credited to Wolfram Kahl; my contribution was to separate them from the material that can already be set in an OSGC.

5.1.1. Categoric.OCC.Order

```
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

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

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

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

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

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

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

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

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

5.1.2. Indirect Equality

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

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

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

Formally,

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

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

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

then the resulting formalization would be:

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

```
    E ⨉ (E ⨾ G ˘)
  ≈⟨ ⨉-M-in-right G-map ⟨≈˘≈⟩ ⨾-cong₁ antisym≈ ⟨≈≈⟩ leftId ⟩
    G ˘
  □
```

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

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

However, the most straightforward approach would be

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

— compare with the point-level, rather than morphism level, presentation of Sect. 3.1 — and to this end we formulate some lemmas and formalize the desideratum as indirect-≈.

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

Compare this with the indirect inclusion of preorders, Sect. 4.1.2.

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

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

indirect-⊑ : {B : Obj} {F G : Mor B A}
  → isMapping F → isMapping G → Id ⊑ (E ⨾ F ˘) ╳ (E ⨾ G ˘) → G ⊑ F
indirect-⊑ {B} {F} {G} map-F map-G indir = leftId ⟨≈˘⊑⟩
  swap-⊑-⨾-unival˘ (proj₁ map-G) (indirect-⊑-⨾ map-F map-G indir)

indirect-˘-≈ : {B : Obj} {F G : Mor B A}
  → isMapping F → isMapping G → E ⨾ F ˘ ≈ E ⨾ G ˘ → F ≈ G
indirect-˘-≈ {B} {F} {G} map-F map-G indir = ⊑-antisym
  (indirect-⊑ map-G map-F (noy-isReflexive ⟨⊑≈⟩ ╳-cong₁ indir))
  (indirect-⊑ map-F map-G (noy-isReflexive ⟨⊑≈˘⟩ ╳-cong₁ indir))

˘-indirect-≈ : {B : Obj} {F G : Mor B A}
  → isMapping F → isMapping G → F ⨾ E ˘ ≈ G ⨾ E ˘ → F ≈ G
˘-indirect-≈ map-F map-G indir = indirect-˘-≈ map-F map-G
  (˘-involutionRightConv ⟨≈˘≈⟩ (˘-cong indir ⟨≈≈⟩ ˘-involutionRightConv))
```

5.1.3. Univalence

Since the predecessors of an element under a relation are unique, we have certain partial functions:

```
╳-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˘-univalentl : {B : Obj} {R : Mor A B} → isUnivalentl (R ╳ E ˘)
╳-order˘-univalentl {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-l ╳-order˘-univalentl
```

5.1.4. Extrema

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

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

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

Next, mappings are fixed-points of extrema.

```
lub-mapping : {I : Obj} {R : Mor I A} → isMapping R → lub R ≈ R
lub-mapping {I} {R} R-map = ≈-begin
      lub R
   ≈⟨⟩
      ubd R ˘ ╳ E ˘
   ≈⟨ ╳-cong₁ (˘-cong (ubd-mapping R-map) ⟨≈≈⟩ ˘-involution) ⟩
      (E ˘ ⨾ R ˘)      ╳ E ˘
   ≈˘⟨ ╳-in-left R-map ⟩
      R ⨾ (E ˘ ╳ E ˘)
```

```
    ≈⟨ ⨾-cong₂ ˘-antisym≈ ⟨≈≈⟩ rightId ⟩
      R
    □

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

Additionally, extrema are always univalent.

```
  lub-isUnivalentI : {I : Obj} {R : Mor I A} → isUnivalentI (lub R)
  lub-isUnivalentI {I} {R} = ╳-order˘-univalentI

  lub-isUnivalent : {I : Obj} {R : Mor I A} → isUnivalent (lub R)
  lub-isUnivalent = isUnivalent-from-I lub-isUnivalentI

  glb-isUnivalentI : {I : Obj} {R : Mor I A} → isUnivalentI (glb R)
  glb-isUnivalentI {I} {R} = ╳-order-univalentI

  glb-isUnivalent : {I : Obj} {R : Mor I A} → isUnivalent (glb R)
  glb-isUnivalent = isUnivalent-from-I glb-isUnivalentI
```

5.1.5. Order Constructions

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

An antisymmetric preorder is a preorder,

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

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

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

```
  }
  where open IsOrder E₁-isOrder
```

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

```
module SubOrder {A : Obj} {E : Mor A A} (E-isOrder : IsOrder E)
          {Z : Obj} (F : Mapping Z A) (F-inj : isInjective (Mapping.mor F)) where
  open IsOrder E-isOrder
  private
    F₀ = Mapping.mor F
    F-isM = Mapping.prf F
    F-unival = mappingUnivalent F
  open IsPreorder₂ occ (retractPreorder occ isPreorder F)
  subOrder : Mor Z Z
  subOrder = F₀ ⨾ E ⨾ F₀ ˘
  subOrder-isOrder    : IsOrder subOrder
  subOrder-isOrder = record
    {refl = refl₂; trans = trans₂
    ; antisym = ⊑-begin
            (F₀ ⨾ E ⨾ F₀ ˘) ⊓ (F₀ ⨾ E ⨾ F₀ ˘)
        ≈⟨ ⊓-cong ⨾-assocL ⨾-assocL ⟩
            ((F₀ ⨾ E) ⨾ F₀ ˘) ⊓ ((F₀ ⨾ E) ⨾ F₀ ˘)
        ≈⟨ ⊓-in-left F-isM ⟨≈˘≈˘⟩ ⨾-cong₂ (⊓-in-right (˘-isBijective F-isM)) ⟩
            F₀ ⨾ ((F₀ ⨾ E) ⊓ (F₀ ⨾ E)) ⨾ F₀ ˘
        ⊑⟨ retract⊓ rightSupId rightSupId
          (⊑-begin
              (E ⨾ F₀ ˘) ⨾ (F₀ ⨾ E) ⨾ F₀ ˘
            ⊑⟨ ⨾-assoc ⟨≈⊑⟩ ⨾-monotone₂ (⨾-₁₂₁assoc₂₂ ⟨≈⊑⟩ proj₁ F-unival) ⟩
              E ⨾ E ⨾ F₀ ˘
            ⊑⟨ ⨾-assocL ⟨≈⊑⟩ ⨾-monotone₁ trans ⟩
              E ⨾ F₀ ˘
            □)
          (⊑-begin
              F₀ ⨾ (F₀ ⨾ E) ˘ ⨾ (E ⨾ F₀ ˘) ˘
            ≈⟨ ⨾-cong₂ (⨾-cong ˘-involution ˘-involutionRightConv ⟨≈≈⟩ ⨾-assoc) ⟩
              F₀ ⨾ E ˘ ⨾ F₀ ˘ ⨾ F₀ ⨾ E ˘
            ⊑⟨ ⨾-monotone₂₂ (⨾-assocL ⟨≈⊑⟩ proj₁ F-unival) ⟩
              F₀ ⨾ E ˘ ⨾ E ˘
            ⊑⟨ ⨾-monotone₂ ˘-trans ⟩
              F₀ ⨾ E ˘
            ≈˘⟨ ˘-involutionRightConv ⟩
              (E ⨾ F₀ ˘) ˘
            □)
          ⟩
            ((E ⨾ F₀ ˘) ⊓ (E ⨾ F₀ ˘))
        ≈⟨     ⊓-in-left F-isM ⟨≈˘≈˘⟩ ⨾-cong₂ (⊓-in-right (˘-isBijective F-isM)) ⟩
            F₀ ⨾ (E ⊓ E) ⨾ F₀ ˘
```

```
        ⊑⟨ ⨾-cong₂ (⨾-cong₁ antisym≈ ⟨≈≈⟩ leftId) ⟨≈⊑⟩ isInjective-to-I F-inj ⟩
          Id
        □
  }
```

5.1.6. Preorders Induced By Residuals and Endowed with Syqs

Just as before, residuals induce a preorder.

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

5.1.7. Singletons

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

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

Formally, and generally,

```
  wrap : Mor A B
  wrap = Id ╳ R
  wrap-injective : isInjectiveI wrap
  wrap-injective =     ⊑-begin
            (Id ╳ R) ⨾ (Id ╳ R) ˘
      ≈⟨ ⨾-cong₂ ╳-˘ ⟩
            (Id ╳ R) ⨾ (R ╳ Id)
```

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

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

5.1.8. Orders Induced by Residuation and Endowed with Comprehension

```
module \-OrderWithComprehension {A B : Obj} {R : Mor A B}
  (isOrder : IsOrder (R \ R)) (comprehensiveI : {C : Obj} {Q : Mor A C} → isTotalI (Q ≬ R))
  where
  open \-Preorder″ R
  open IsOrder isOrder
  comprehensive : {C : Obj} {Q : Mor A C} → isTotal (Q ≬ R)
```

```
comprehensive = isTotal-from-I comprehensiveI

Ω : Mor B B
Ω = R \ R

Ω˘ : Ω ˘ ≈ R ˘ / R ˘
Ω˘ = \-˘

R⨾-R╳ : {C : Obj} {Q : Mor A C} → R ⨾ (R ╳ Q) ≈ Q
R⨾-R╳ {C} {Q} = ╳-surjective-cancel-left (isSurjectiveFromTotal (≈-isTotal ╳-˘ comprehensive))
```

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

```
lubΩ≈╳ : {I : Obj} {Q : Mor I B} → lub Q ≈ (R ⨾ Q ˘) ╳ R
lubΩ≈╳ {I} {Q} = ≈-sym (total⊑unival-≈ comprehensive lub-isUnivalent (⊑-begin
          (R ⨾ Q ˘) ╳ R
     ⊑⟨ ╳⊑\˘╳\˘ ⟩
          ((R ⨾ Q ˘) \ R) ˘ ╳ (R \ R) ˘
     ≈⟨     ╳-cong (\-˘ ⟨≈≈⟩ /-cong₂ ˘-involutionRightConv) \-˘ ⟩
          (R ˘ / (Q ⨾ R ˘)) ╳ (R ˘ / R ˘)
     ≈˘⟨        ╳-cong₁ // ⟩
          ((R ˘ / R ˘) / Q) ╳ (R ˘ / R ˘)
     ≈˘⟨        ╳-cong (/-cong₁ Ω˘) Ω˘ ⟩
          (Ω ˘ / Q) ╳ Ω ˘
     ≈˘⟨ ╳-cong₁ ˘\-˘ ⟩
          (Q ˘ \ Ω) ˘ ╳ Ω ˘
     ≈⟨⟩
          ubd Q ˘ ╳ Ω ˘
     ≈⟨⟩
          lub Q
     □))

lubΩ-totalI : {I : Obj} {Q : Mor I B} → isTotalI (lub Q)
lubΩ-totalI = ≈-isTotalI lubΩ≈╳   comprehensiveI

lubΩ-total : {I : Obj} {Q : Mor I B} → isTotal (lub Q)
lubΩ-total = ≈-isTotal lubΩ≈╳ comprehensive

lubΩ : {I : Obj} (Q : Mor I B) → Mapping I B
lubΩ Q = record {mor = lub Q; prf = lub-isUnivalent, lubΩ-total}
```

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

```
glbΩ-totalI : {I : Obj} {Q : Mor I B} → isTotalI (glb Q)
glbΩ-totalI = ≈-isTotalI glb-≈-lub-lbd lubΩ-totalI

glbΩ-total : {I : Obj} {Q : Mor I B} → isTotal (glb Q)
glbΩ-total = ≈-isTotal glb-≈-lub-lbd lubΩ-total

glbΩ : {I : Obj} (Q : Mor I B) → Mapping I B
glbΩ Q = record {mor = glb Q; prf = glb-isUnivalent, glbΩ-total}

lub-⨾wrap : {I : Obj} {Q : Mor I A} → isTotal wrap → lub (Q ⨾ wrap) ≈ Q ˘ ╳ R
lub-⨾wrap {I} {Q} total = ≈-begin
```

```
      lub (Q ⨾ wrap)
    ≈⟨ lubΩ≈╳ ⟨≈≈⟩ ╳-cong₁ (⨾-cong₂ ˘-involution) ⟩
      (R ⨾ wrap ˘ ⨾ Q ˘) ╳ R
    ≈⟨ ╳-cong₁ (⨾-assocL ⟨≈≈⟩ ⨾-cong₁ (R⨾wrap˘ total) ⟨≈≈⟩ leftId) ⟩
      Q ˘ ╳ R
    □

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

5.1.9. 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:

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

Indeed,

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

∈⇒Λ : {I : Obj} {Q : Mor I A} {f : Mapping I B}
  → Mapping.mor f ⨾ R ˘ ≈ Q → Mapping.mor f ≈ Λ₀ Q
∈⇒Λ {I} {Q} {f} f⨾R˘≈Q  =  ≈-sym (≈-begin
    Q ˘ ╳ R
```

```
≈⟨ ⨉-cong₁ (˘-cong f⨾R˘≈Q ⟨≈˘≈⟩ ˘-involutionRightConv) ⟩
   (R ⨾ f₀ ˘) ⨉ R
≈˘⟨ lubΩ≈⨉ ⟩
   lub f₀
≈⟨ lub-mapping (Mapping.prf f) ⟩
   f₀
□) where f₀ = Mapping.mor f
```

5.2. Categoric.OCC.Order.Closure

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

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

5.2.1. 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
```

5.2.2. GLB Closure

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

```
glb-closed-⊑ : {I : Obj} {R : Mor I A} → R ⨾ C ≈ R → glb R ⨾ C ⊑ glb R
glb-closed-⊑ {I} {R} R⨾C≈R = ˘⨉-universal
```

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

Gratitude to Wolfram Kahl for these three proofs.

5.2.3. 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
```

5.3. Categoric.OCC.Order.Galois

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

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

5.3.1. 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 ULU∃≈U∃
```

5.3.2. 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-abs∃ eq ⟨≈≈˘⟩ ⨾-assoc)
```

5.3.3. Idempotency and Coclosure

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

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

5.3.4. 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
)
```

5.4. Conclusion

It is amazing what we have achieved by the sheer addition of antisymmetry! Not only do we now have the traditional poset results, along with the a few closure and Galois connection results, but they are in a general setting, ready to be re-used both as a programming construct and as a generic reasoning construct.

6. Polarities: An Application

We now access certain results of Kahl (2014a) and present them as instances of the theorems of Galois connections. Kahl did not have access to internal Galois connections and so proved his results directly. We shall use his proof of the polarities' connection and derive many of his results. Moreover, we shall briefly compare the normal form term complexity of his direct proofs with our derived ones.

We will not name all of our derivations but rather only those mentioned in Kahl (2014a) that express a property in terms of the order and the adjoints alone. In particular, we consider

```
↑↓⊑Ω : {A B : Obj} {R : Mor A B} → R ↑↓₀ ⊑ Ω
↓↑⊑Ω : {A B : Obj} {R : Mor A B} → R ↓↑₀ ⊑ Ω
↓⨾↑↓ : {A B : Obj} {R : Mor A B} → R ↓ ⨾₁ R ↑↓ ≈₁ R ↓
↑↓-idempotent : {A B : Obj} {R : Mor A B} → R ↑↓ ⨾₁ R ↑↓ ≈₁ R ↑↓
↓↑-idempotent : {A B : Obj} {R : Mor A B} → R ↓↑ ⨾₁ R ↓↑ ≈₁ R ↓↑
↓↑-monotone : {A B : Obj} {S : Mor A B} → Ω ⨾ S ↓↑₀ ⊑ S ↓↑₀ ⨾ Ω
```

where

$$R \uparrow (A) \; = \; \text{“the } R\text{-successors of all of } A\text{”} \; = \; \{s \;\mid\; (\forall e \;\mid\; e \in A \;\bullet\; e\,R\,s)\} \; = \; \Lambda(\in \backslash R)(A)$$

and likewise $R \downarrow (B)$ = “the R-predecessors of all of B”. These two operations constitute an antitone Galois connection, as already proved in Kahl (2014a).

```
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 hiding (∃₁; ∃₂) 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-⊑-∃ 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-abs∃ 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 ˘
    ; ULU∃≈U∃ to ↓↑↓-semi-Ω˘   -- : (R ↓₀ ⨾ R ↑↓₀) ⨾ Ω ˘ ≈ R ↓₀ ⨾ Ω ˘
    ; L-monotone      to ↑-antitone        -- : Ω ⨾ R ↑₀ ⊑ R ↑₀ ⨾ Ω ˘
    ; U-monotone      to ↓-antitone        -- : Ω ˘ ⨾ R ↓₀ ⊑ R ↓₀ ⨾ Ω
    ; L-isotone-on-U to ↑-isotone-on-↓   -- : R ↓₀ ⨾ R ↑₀ ⨾ Ω ˘ ⨾ R ↑₀ ˘ ⨾ R ↓₀ ˘ ≈ R ↓₀ ⨾ Ω ⨾ R ↓₀ ˘
    ; U-isotone-on-L to ↓-isotone-on-↑   -- : R ↑₀ ⨾ R ↓₀ ⨾ Ω ⨾ R ↓₀ ˘ ⨾ R ↑₀ ˘ ≈ R ↑₀ ⨾ Ω ˘ ⨾ R ↑₀ ˘
    ; UL-comonotone to ↓↑-monotone₀   -- : Ω ˘ ˘ ⨾ R ↓↑₀ ⊑ R ↓↑₀ ⨾ Ω ˘ ˘
    ; LU-monotone    to ↑↓-monotone    -- : Ω ⨾ R ↑↓₀ ⊑ R ↑↓₀ ⨾ Ω
    )
```

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

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

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

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

name	direct proof			via Galois connection		
	lines	xz	min	lines	xz	min
↓↑-monotone	13902	14904	8	653203	861160	280
↑↓⊑Ω :	3171	5000	2.5	7691	10744	3
↓↑↓-semi	27231	26716	19	49274	46396	40
↓↑-idemp	27860	28552	18	336908	324960	171
P.↓↑-ranClosed-←	615	1444	0.383	615	1444	0.383
↓↑-ranClosed-→	62026	61316	25	-	-	> 60

Besides the costs, notice that many theorems fall out of the connection. We have stated even some that do not occur in Kahl (2014a). However, with our modules in hand, such results can now immediately be instantiated and thus save time.

7. Conclusion

We have now displayed an mechanised internalisation of Galois connections, which may serve for both programming and reasoning tasks. Since the exposition is in a general framework, it is amiable to resuability; and the calculational presentation is readable and not much more difficult than a LaTeX presentation.

The contributions of this thesis may be summarised as follows:

- a mechanised formulation of the already existing theory of external Galois connections
- a mechanised formulation of a newly introduced theory of internal closure operators
- a mechanised formulation of a newly introduced theory of internal Galois connections

Moreover, our novel theories permit a translation of the common pointwise setting into the internal, more general, setting — namely, the proof heuristics of indirect inclusion, for preorders, and indirect equality for orders.

As the observant reader may have noticed, the junctivity propositions shown in the external setting have no counterpart in the internal setting. This is an aim for future work. Also for future work are the following

- To isolate the conditions a model must satisfy such that only the quasi laws hold and their true counterparts do not; if such models exist.
- Categorical structure of GC's — they are closed under composition and the identity is connected to itself.
- Formulations of internal GC's as residuated mappings.
- The interaction of internal GC's and fixed points.
- Interaction with Chu Spaces — the GC characterisation is an instance of the coherency condition on morphisms in the category of Chu spaces, and so of interest would be to see how our properties can be generalised to that setting.
- Characterisation internal lattices via existence of certain adjoints.
  A similar undertaking has begun in Al-hassy and Kahl (2015).

There is so much to be done; we have only begun!

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 and Wolfram Kahl. Relation-algebraic order theory in ordered categories without meets. April 2015. 16 pages, submitted for publication.

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.

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

Hitoshi Furusawa and Wolfram Kahl. A study on symmetric quotients. Technical Report 1998-06, Fakultät für Informatik, Universität der Bundeswehr München, December 1998.

F. Garcia-Pardo, I. P. Cabrera, P. Cordero, and Manuel Ojeda-Aciego. On Galois connections and soft computing. In *Advances in Computational Intelligence*, volume 7903 of *LNCS*, pages 224–235. Springer-Verlag Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-38682-4.

David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math*. Monographs in Computer Science. Springer, 1993. ISBN 3-540-94115-0. URL http://www.springer.de/cgi-bin/search_book.pl?isbn=3-540-94115-0.

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.

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. Dependently-typed formalisation of relation-algebraic abstractions. In Harrie de Swart, editor, *Relational and Algebraic Methods in Computer Science, RAMiCS 2011*, volume 6663 of *LNCS*, pages 230–247. Springer, 2011. doi: 10.1007/978-3-642-21070-9_18.

Wolfram Kahl. A mechanised abstract formalisation of concept lattices. In Peter Höfner, Peter Jipsen, Wolfram Kahl, and Martin Eric Müller, editors, *Relational and Algebraic Methods in Computer Science, RAMiCS 2014*, volume 8428 of *LNCS*, pages 242–260. Springer, 2014a. ISBN 978-3-319-06250-1. doi: 10.1007/978-3-319-06251-8.

Wolfram Kahl. Relation-Algebraic Theories in Agda — RATH-Agda-2.0.1. Mechanically checked Agda theories available for download, with 456 pages literate document output. `http://RelMiCS.McMaster.ca/RATH-Agda/`., February 2014b. With contributions by Yuhang Zhao.

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

Shin-Cheng Mu and José Nuno Oliveira. Programming from Galois connections. *J. Logic and Algebraic Programming*, 81(6):680–704, August 2012. doi: 10.1016/j.jlap.2012.05.003.

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.

A. Galois Connection Properties

A.1. Constructions

We list a few approaches to obtain connections from a given connection.

- Dualisation: $\langle \mathsf{L}, \leq \rangle \dashv \langle \mathsf{U}, \sqsubseteq \rangle \Leftrightarrow \langle \mathsf{U}, \sqsupseteq \rangle \dashv \langle \mathsf{L}, \geq \rangle$
- Uniqueness: If $\mathsf{L} \dashv \mathsf{U} \wedge \mathsf{L}' \dashv \mathsf{U}'$ then $\mathsf{L} \approx \mathsf{L}' \Leftrightarrow \mathsf{U} \approx \mathsf{U}'$.
- Composition: If $\mathsf{L} \dashv \mathsf{U}$ and $\mathsf{L}' \dashv \mathsf{U}'$, then $\mathsf{L} \fatsemi \mathsf{L}' \dashv \mathsf{U}' \fatsemi \mathsf{U}$.
- Pre- and Post-Composition: If $\mathsf{L} \dashv \mathsf{U}$ then, on mappings, $(\fatsemi L) \dashv (\fatsemi U)$ and $(U \fatsemi) \dashv (L \fatsemi)$.
- GC Hom-Functor: If $\mathsf{L} \dashv \mathsf{U}$ then $\mathsf{M}\ (\mathsf{L}, \mathsf{U}) \dashv \mathsf{M}\ (\mathsf{U}, \mathsf{L})$, where $\mathsf{M}\ (\mathsf{f}, \mathsf{g}) = \lambda\ (\mathsf{h} : \mathsf{Monotone}) \to \mathsf{g} \fatsemi \mathsf{h} \fatsemi \mathsf{f}$.

A.2. Most Commonly Used Laws

We recall the most common laws used when utilising Galois connections as a tool.

To avoid repetition, take $\{\mathsf{f}, \mathsf{g}\} = \{\mathsf{L}, \mathsf{U}\}$, $\langle \Box_L, \Box_U, \sim_L, \sim_U \rangle = \langle \sqcap, \sqcup, \leq, \sqsupseteq \rangle$, and let h, k be arbitrary well-typed maps.

- Functional Connection: $\mathsf{h} \fatsemi \mathsf{L} \sqsubseteq \mathsf{k} \Leftrightarrow \mathsf{h} \leq \mathsf{k} \fatsemi \mathsf{U}$
- Interdefinability: $\mathsf{k} \fatsemi \mathsf{f} \approx (\Box_f\ \mathsf{h} \mid \mathsf{k}\ \sim_f\ \mathsf{h} \fatsemi \mathsf{g} \bullet \mathsf{h})$
- Cancellation: $\mathsf{Id}\ \sim_f\ \mathsf{f} \fatsemi \mathsf{g}$
- Congruence: $\mathsf{h} \approx \mathsf{k} \Rightarrow \mathsf{h} \fatsemi \mathsf{f} \approx \mathsf{k} \fatsemi \mathsf{f}$
- Monotonicity: $\mathsf{h}\ \sim_f\ \mathsf{k} \Rightarrow \mathsf{k} \fatsemi \mathsf{f}\ \sim_g\ \mathsf{h} \fatsemi \mathsf{f}$
- Image Isotonicity: $\mathsf{h} \fatsemi \mathsf{g} \fatsemi \mathsf{f}\ \sim_g\ \mathsf{k} \fatsemi \mathsf{g} \fatsemi \mathsf{f} \Leftrightarrow \mathsf{k} \fatsemi \mathsf{g}\ \sim_f\ \mathsf{h} \fatsemi \mathsf{g}$
- Semi-inverse: $\mathsf{f} \fatsemi \mathsf{g} \fatsemi \mathsf{f} \approx \mathsf{f}$
- Absorption: $\mathsf{h} \fatsemi \mathsf{g} \fatsemi \mathsf{f} \approx \mathsf{k} \fatsemi \mathsf{g} \fatsemi \mathsf{f} \Leftrightarrow \mathsf{h} \fatsemi \mathsf{g} \approx \mathsf{k} \fatsemi \mathsf{g}$
- Elimination / Introduction: $\mathsf{k} \fatsemi \mathsf{g} \fatsemi \mathsf{f}\ \sim_g\ \mathsf{h} \fatsemi \mathsf{f} \Leftrightarrow \mathsf{h}\ \sim_f\ \mathsf{k} \fatsemi \mathsf{g}$
- Interchange: $\mathsf{h} \fatsemi \mathsf{g} \fatsemi \mathsf{f}\ \sim_g\ \mathsf{k} \fatsemi \mathsf{f} \Leftrightarrow \mathsf{k} \fatsemi \mathsf{f} \fatsemi \mathsf{g}\ \sim_f\ \mathsf{h} \fatsemi \mathsf{g}$
- Image Junctivity: $\Box\ (\mathsf{h} \fatsemi \mathsf{g} \fatsemi \mathsf{f}) \approx (\Box\ (\mathsf{h} \fatsemi \mathsf{g})) \fatsemi \mathsf{f}$ where $\Box \in \{\bigsqcup, \bigsqcap\}$
  In fact, the infimum preserving functions are precisely the upper adjoints, while the supremum preserving functions are precisely the lower adjoints.
- General Junctivity: $\Box_f\ (\mathsf{h} \fatsemi \mathsf{g}) \approx (\Box_f\ \mathsf{h}) \fatsemi \mathsf{g}$
- (Co)Closure: $\mathsf{f} \fatsemi \mathsf{g}$ is idempotent, monotonic, and increasing
- Closure = Image: $\mathsf{h} \fatsemi \mathsf{g} \fatsemi \mathsf{f} \approx \mathsf{h} \Leftrightarrow (\exists\ \mathsf{k} \bullet \mathsf{k} \fatsemi \mathsf{f} \approx \mathsf{h}) \Leftrightarrow \mathsf{h} \fatsemi \mathsf{g} \fatsemi \mathsf{f}\ \sim_g\ \mathsf{h}$
- Perfection: $\mathsf{f} \fatsemi \mathsf{g} \approx \mathsf{Id} \Leftrightarrow \mathsf{f}$ monic $\Leftrightarrow \mathsf{f}$ isotonic $\Leftrightarrow \mathsf{g}$ isotonic $\Leftrightarrow \mathsf{g}$ epic.

B. Ordered Categories with Converse

Ordered Categories with Converse & Division

Informally, We have a sort 'objects', such that

- objects act like types for a sort 'arrows'
- arrows are ordered $\_\sqsubseteq\_$
- 'appropriate' arrows can be 'multiplied' $\_ ⨾ \_$
- the multiplication is associative with a unit, and respects the order
- arrows can be 'flipped' with a 'converse' operator $\_\breve{}$
- converse respects the order, but flips the multiplication
- three operations $/, \backslash, ╳$ on arrows that satisfy the characterisations:

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

$$\forall\, Q, R, S \bullet Q \sqsubseteq R ╳ S \quad \Leftrightarrow \quad R ⨾ Q \sqsubseteq S \wedge Q ⨾ S\,\breve{} \sqsubseteq R\,\breve{}$$

In this chapter we speedily build up the OCC structure, while the subsequent chapters discuss division. This material is completely formalised within the RATH-Agda (Kahl, 2011, 2014b) library.

B.1. Objects

We have a collection of 'objects' Obj,

record OCC {i : Level} (j k_1 k_2 : Level) (Obj : Set i) : Set (i ⊍ ℓsuc (j ⊍ k_1 ⊍ k_2)) **where**

For each pair of objects A and B, we have a poset Hom A B, whose underlying set is termed Mor A B, and the ordering is denoted $\_\sqsubseteq\_$,

```
field
  Hom : Obj → Obj → Poset j k₁ k₂
Mor : Obj → Obj → Set j
```

B.2. Composition

We have a method to ‘multiply’ appropriate elements of the posets,

```
field
  _⨾_      :   {A B C : Obj} → Mor A B → Mor B C → Mor A C
  ⨾-cong   :   {A B C : Obj} {f₁ f₂ : Mor A B} {g₁ g₂ : Mor B C}
           →   f₁ ≈ f₂ → g₁ ≈ g₂ → (f₁ ⨾ g₁) ≈ (f₂ ⨾ g₂)
  ⨾-assoc  :   {A B C D : Obj} {f : Mor A B} {g : Mor B C} {h : Mor C D}
           →   ((f ⨾ g) ⨾ h) ≈ (f ⨾ (g ⨾ h))
⨾-assocL   : ∀ {f g h} → f ⨾ (g ⨾ h) ≈ (f ⨾ g) ⨾ h
⨾-assoc₄   : ∀ {f g h j} → ((f ⨾ g) ⨾ h) ⨾ j ≈ f ⨾ (g ⨾ (h ⨾ j))
⨾-assoc₃₊₁ : ∀ {f g h j} → (f ⨾ g ⨾ h) ⨾ j ≈ f ⨾ (g ⨾ (h ⨾ j))
```

The multiplication respects the poset order,

```
field
  ⨾-monotone : {A B C : Obj} {f f' : Mor A B} {g g' : Mor B C}
             → f ⊑ f' → g ⊑ g' →     (f ⨾ g) ⊑ (f' ⨾ g')
⨾-monotone₁ : ∀ {f f′ g} → f ⊑ f′ → (f ⨾ g) ⊑ (f' ⨾ g)
⨾-monotone₂ : ∀ {f g g′} → g ⊑ g′ → (f ⨾ g) ⊑ (f ⨾ g′)
⨾-monotone₁₂ : ∀ {f g g′ h} → g ⊑ g′ → ((f ⨾ g) ⨾ h) ⊑ ((f ⨾ g′) ⨾ h)
```

Exercise: formulate and prove other variations of $\mathsf{monotone}_{xy}$.

B.3. Identities

So far we have described ordered semigroupoids with converse (OSGC’s), with identities we obtain ordered categories.
(Furthermore, we obtain ‘allegories’ if we request our Hom posets be meet semilattices and request ‘The DeDekind Rule’ hold.)

```
field
  Id      : {A : Obj} → Mor A A
  leftId  : {A B : Obj} → {f : Mor A B} → (Id ⨾ f) ≈ f
  rightId : {A B : Obj} → {f : Mor A B} → (f ⨾ Id) ≈ f
```

B.4. Converse Operator

We have an a contravariant, monotonic, involution:

```
field
  _˘          : {A B : Obj}                                  → Mor A B → Mor B A
  ˘-cong      : {A B : Obj}       {R S : Mor A B}             → R ≈ S → R ˘ ≈ S ˘
  ˘˘          : {A B : Obj}       {R : Mor A B}               → (R ˘) ˘ ≈ R
  ˘-involution : {A B C : Obj} {R : Mor A B} {S : Mor B C} → (R ⨾ S) ˘ ≈ (S ˘ ⨾ R ˘)
≈-˘-swap              : ∀ {R S} → R ≈ S ˘ → R ˘ ≈ S
˘-≈-swap              : ∀ {R S} → R ˘ ≈ S → R ≈ S ˘
˘-≈-˘                 : ∀ {R S} → R ˘ ≈ S ˘ → R ≈ S
un˘-cong              : ∀ {R S} → R ˘ ≈ S ˘ → R ≈ S
˘-coinvolution        : ∀ {R S} → (S ˘ ⨾ R ˘) ˘ ≈ (R ⨾ S)
˘-involutionLeftConv  : ∀ {R S} → (S ˘ ⨾ R) ˘ ≈ (R ˘ ⨾ S)
˘-involutionRightConv : ∀ {R S} → (S ⨾ R ˘) ˘ ≈ (R ⨾ S ˘)
field
    ˘-monotone : {A B : Obj} {R S : Mor A B} → R ⊑ S → (R ˘) ⊑ (S ˘)
˘-isotone : ∀ {R S} → (R ˘) ⊑ (S ˘) → R ⊑   S
⊑-˘-swap : ∀ {R S} → (R ˘) ⊑ S →       R ⊑ (S ˘)
˘-⊑-swap : ∀ {R S} → R   ⊑ (S ˘) → (R ˘) ⊑ S
```

B.5. Definitions of Properties of Morphisms

```
IsTransitive : {A : Obj} → Mor A A → Set k₂
IsTransitive R = R ⨾ R ⊑ R

isCotransitive : {A : Obj} → Mor A A → Set k₂
isCotransitive R = R ⊑ R ⨾ R

isLeftSubidentity : {A : Obj} → (p : Mor A A) → Set (i ⊍ j ⊍ k₂)
isLeftSubidentity {A} p = {B : Obj} {R : Mor A B} → p ⨾ R ⊑ R

isRightSubidentity : {A : Obj} → (p : Mor A A) → Set (i ⊍ j ⊍ k₂)
isRightSubidentity {A} p = {B : Obj} {S : Mor B A} → S ⨾ p ⊑ S

isSubidentity : {A : Obj} → (p : Mor A A) → Set (i ⊍ j ⊍ k₂)
isSubidentity p = isLeftSubidentity p × isRightSubidentity p

isLeftSuperidentity : {A : Obj} → (p : Mor A A) → Set (i ⊍ j ⊍ k₂)
isLeftSuperidentity {A} p = {B : Obj} {R : Mor A B} → R ⊑ p ⨾ R

isRightSuperidentity : {A : Obj} → (p : Mor A A) → Set (i ⊍ j ⊍ k₂)
isRightSuperidentity {A} p = {B : Obj} {S : Mor B A} → S ⊑ S ⨾ p

isSuperidentity : {A : Obj} → (p : Mor A A) → Set (i ⊍ j ⊍ k₂)
isSuperidentity p = isLeftSuperidentity p × isRightSuperidentity p

isUnivalent : {A B : Obj} → Mor A B → Set (i ⊍ j ⊍ k₂)
isUnivalent R = isSubidentity (R ˘ ⨾ R)
```

```
isTotal : {A B : Obj} → Mor A B → Set (i ⊍ j ⊍ k₂)
isTotal R = isSuperidentity (R ⨾ R ˘)

isMapping : {A B : Obj} → Mor A B → Set (i ⊍ j ⊍ k₂)
isMapping R = isUnivalent R × isTotal R

isInjective : {A B : Obj} → Mor A B → Set (i ⊍ j ⊍ k₂)
isInjective R = isSubidentity (R ⨾ R ˘)

isSurjective : {A B : Obj} → Mor A B → Set (i ⊍ j ⊍ k₂)
isSurjective R = isSuperidentity (R ˘ ⨾ R)

isBijective : {A B : Obj} → Mor A B → Set (i ⊍ j ⊍ k₂)
isBijective R = isInjective R × isSurjective R

record Mapping (A B : Obj) : Set (i ⊍ j ⊍ k₂) where
  constructor mkMapping
  field
    mor : Mor A B
    prf : isMapping mor

  unival : isUnivalent mor
  unival = proj₁ prf

  total : isTotal mor
  total = proj₂ prf

isDifunctional : {A B : Obj} → Mor A B → Set k₂
isDifunctional R = R ⨾ R ˘ ⨾ R ⊑ R

isCodifunctional : {A B : Obj} → Mor A B → Set k₂
isCodifunctional R = R ⊑ R ⨾ R ˘ ⨾ R
```

B.6. Swaps

```
swap-⊑-⨾-unival  : ∀ {Q R S} → isUnivalent R → Q ⊑ R ⨾ S → R ˘ ⨾ Q ⊑ S
swap-⊑-⨾-unival˘ : ∀ {Q R S} → isUnivalent S → Q ⊑ R ⨾ S ˘ → Q ⨾ S ⊑ R

swap-⨾-⊑-total˘  : ∀ {Q R S} → isTotal Q → Q ˘ ⨾ R ⊑ S → R ⊑ Q ⨾ S
swap-⨾-⊑-total   : ∀ {Q R S} → isTotal R → Q ⨾ R ⊑ S → Q ⊑ S ⨾ R ˘

swap-⊑-⨾-inj˘    : ∀ {Q R S} → isInjective R → Q ⊑ R ˘ ⨾ S → R ⨾ Q ⊑ S
swap-⊑-⨾-inj     : ∀ {Q R S} → isInjective S → Q ⊑ R ⨾ S → Q ⨾ S ˘ ⊑ R

swap-⨾-⊑-surj    : ∀ {Q R S} → isSurjective Q → Q ⨾ R ⊑ S → R ⊑ Q ˘ ⨾ S
swap-⨾-⊑-surj˘   : ∀ {Q R S} → isSurjective R → Q ⨾ R ˘ ⊑ S → Q ⊑ S ⨾ R

swap-≈-⨾-totalInj  : ∀ {Q R S} → isTotal S → isInjective S → Q ≈ R ⨾ S → Q ⨾ S ˘ ≈ R
swap-≈-⨾-totalInj˘ : ∀ {Q R S} → isTotal R → isInjective R → Q ≈ R ˘ ⨾ S → R ⨾ Q ≈ S

swap-≈-⨾-univalSurj  : ∀ {Q R S} → isUnivalent R → isSurjective R → Q ≈ R ⨾ S → R ˘ ⨾ Q ≈ S
swap-≈-⨾-univalSurj˘ : ∀ {Q R S} → isUnivalent S → isSurjective S → Q ≈ R ⨾ S ˘ → Q ⨾ S ≈ R
```

C. Residuals

A residual is a form of division operation; see Sect. 2.3 for instances of this concept. Residuals are characterised by the Galois connections,

$$\forall\, Q, R, S \bullet Q \sqsubseteq R \setminus S \Leftrightarrow R \mathbin{;} Q \sqsubseteq S \Leftrightarrow R \sqsubseteq S \,/\, Q$$

That is, $R \setminus S$, read “R under S”, is the largest solution Q of the inclusion $R \mathbin{;} Q \sqsubseteq S$; while $S \,/\, Q$, read “S over Q”, is the largest solution R of the inclusion $R \mathbin{;} Q \sqsubseteq S$.

C.1. Definition

```
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
  field
    _/_            : {A B C : Obj} → Mor A C → Mor B C → Mor A B
    /-cancel-outer : {A B C : Obj} {S : Mor A C} {R : Mor B C} → (S / R) ⨾ R ⊑ S
    /-universal    : {A B C : Obj} {S : Mor A C} {R : Mor B C} {Q : Mor A B} → Q ⨾ R ⊑ S → Q ⊑ S / R
  /-couniversal    : ∀ {S R T}    → ({X : Mor A B} → X ⨾ R ⊑ S → X ⊑ T) →       S / R ⊑
  /-universal′     : ∀ {S R Q}    →          Q ⊑ S / R → Q ⨾ R ⊑ S
  /-cancel-inner   : ∀ {T S}      → T ⊑ (T ⨾ S) / S
  /-monotone       : ∀ {S₁ S₂ R}  → S₁ ⊑ S₂ → S₁ / R ⊑ S₂ / R
  /-cong₁          : ∀ {S₁ S₂ R}  → S₁ ≈ S₂ → S₁ / R ≈ S₂ / R
  /-antitone       : ∀ {S R₁ R₂}  → R₂ ⊑ R₁ → S / R₁ ⊑ S / R₂
  /-cong₂          : ∀ {S R₁ R₂}  → R₁ ≈ R₂ → S / R₁ ≈ S / R₂
  /-cong           : ∀ {S₁ S₂ R₁ R₂} → S₁ ≈ S₂ → R₁ ≈ R₂ → S₁ / R₁ ≈ S₂ / R₂
  /-cancel-outer²  : ∀ {S R T}    →      (S / R) ⨾ (R / T) ⨾ T ⊑ S
  /-cancel-middle  : ∀ {S R T}    →      (S / R) ⨾ (R / T) ⊑ S / T
  /-cancel-⨾       : ∀ {S R T}    →      S / R ⊑ (S ⨾ T) / (R ⨾ T)
  /-outer-⨾        : ∀ {F S R}    →      F ⨾ (S / R) ⊑ (F ⨾ S) / R
  //               : ∀ {Q R S}    →      (S / R) / Q ≈ S / (Q ⨾ R)
  /-cancel-⨾-inner : ∀ {Q R S}    → (S / (Q ⨾ R)) ⨾ Q ⊑ S / R
```

Now, you the reader, dualise and define the notion of right residuals. We can now obtain relationships connecting the two residuals, e.g.,

```
⊑-S/∘\S : ∀ {S Q} → Q ⊑ S / (Q \ S)
⊑-\S∘S/ : ∀ {S Q} → R ⊑ (S / R) \ S
```

C.2. Interaction with Mappings

```
\-inner-⨾ : ∀ {S Q F} → isMapping F → F ⨾ (Q \ S) ≈ (Q ⨾ F ˘) \ S
/-inner-⨾ : ∀ {S R F} → isMapping F → (S / R) ⨾ F ˘ ≈ S / (F ⨾ R)

/-outer-⨾-≈  : ∀ {F S R} → isMapping F → F ⨾ (S / R) ≈ (F ⨾ S) / R
\-outer-⨾-≈  : ∀ {F S Q} → isMapping F → (Q \ S) ⨾ F ˘ ≈ Q \ (S ⨾ F ˘)

/-flip   : ∀ {S R F} → isMapping F → S / (R ⨾ F) ≈ (S ⨾ F ˘) / R
\-flip-M : ∀ {S F Q} → isMapping F → (F ˘ ⨾ Q) \ S ≈ Q \ (F ⨾ S)
flip-\ : ∀ {S F Q} → isMapping F → Q \ (F ⨾ S) ≈ (F ˘ ⨾ Q) \ S
```

C.3. Interaction with Converse

```
/-˘   : ∀ {S R} → (S / R) ˘      ≈ R ˘ \ S ˘
/˘-˘ : ∀ {S R} → (S / R ˘) ˘    ≈ R \ S ˘
˘/-˘ : ∀ {S R} → (S ˘ / R) ˘    ≈ R ˘ \ S
˘/˘-˘ : ∀ {S R} → (S ˘ / R ˘) ˘ ≈ R \ S
\-˘   : ∀ {Q S} → (Q \ S) ˘      ≈ S ˘ / Q ˘
˘\-˘ : ∀ {Q S} → (Q ˘ \ S) ˘    ≈ S ˘ / Q
\˘-˘ : ∀ {Q S} → (Q \ S ˘) ˘    ≈ S / Q ˘
˘\˘-˘ : ∀ {Q S} → (Q ˘ \ S ˘) ˘ ≈ S / Q
```

C.4. Interaction with Identities

```
/-isReflexive    : ∀ {R} → Id ⊑ R / R
/-isSuperidentity : ∀ {R} → isSuperidentity (R / R)
/-Id             : ∀ {R} → R / Id ≈ R
preorder-/       : ∀ {E} → IsReflexive E → IsTransitive E → E / E ≈ E
```

Dually for /; formalise it, reader.

```
/-twist-up-≈ : ∀ {S R} → S / R ≈ (S / S) \ (S / R)
\-twist-up-≈ : ∀ {S Q} → Q \ S ≈ (Q \ S) / (S \ S)
```

D. Symmetric Quotients

Symmetric quotients were originally studied by Berghammer et al. (1986, 1989) in relation algebras, and by Freyd and Scedrov (1990) in division allegories. Syqs are characterised by the equivalence,

$$\forall\, Q, R, S \bullet Q \sqsubseteq R \between S \quad \Leftrightarrow \quad R \mathbin{;} Q \sqsubseteq S \wedge Q \mathbin{;} S^{\smile} \sqsubseteq R^{\smile}$$

That is, $R \between S$ is the largest solution Q to the system of simultaneous inclusions:

$$\begin{aligned} R \mathbin{;} Q &\sqsubseteq S \\ Q \mathbin{;} S^{\smile} &\sqsubseteq R^{\smile} \end{aligned}$$

The symbol _≬_ is usually read “syq”.

D.1. Definition

```
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
  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 : ∀ {Q₁ Q₂ S} → Q₁ ≈ Q₂ → Q₁ ≬ S ≈ Q₂ ≬ S
  ≬-cong₂ : ∀ {Q S₁ S₂} → S₁ ≈ S₂ → Q ≬ S₁ ≈ Q ≬ S₂

  ≬-universal-right  :  ∀ {Q S R} → R ⊑ Q ≬ S → Q ⨾ R ⊑ S
  ≬-universal-left   :  ∀ {Q S R} → R ⊑ Q ≬ S → R ⨾ S ˘ ⊑ Q ˘
  ≬-universal-left˘  :  ∀ {Q S R} → R ⊑ Q ≬ S → S ⨾ R ˘ ⊑ Q

  ≬-˘                  : ∀ {Q S}    → (Q ≬ S) ˘ ≈ S ≬ Q
  ≬-cancel-inner       : ∀ {Q S P} →  Q ≬ S ⊑ (P ⨾ Q) ≬ (P ⨾ S)
  ≬-cancel-middle      : ∀ {Q S T} →  (Q ≬ S) ⨾ (S ≬ T) ⊑ Q ≬ T
  ≬-isDifunctional     : ∀ {Q S}    → IsDifunctional (Q ≬ S)

  ≬-total-cancel-right  : ∀ {Q S}   →  isTotal (Q ≬ S) → (Q ≬ S) ⨾ S ˘ ≈ Q ˘
  ≬-total-cancel-middle : ∀ {Q S T} →  isTotal (Q ≬ S) → (Q ≬ S) ⨾ (S ≬ T) ≈ Q ≬ T
```

```
╳-surjective-cancel-left   : ∀ {Q S} → isSurjective (Q ╳ S) → Q ⨾ (Q ╳ S) ≈ S
╳-surjective-cancel-middle : ∀ {Q S T} → isSurjective (S ╳ T) → (Q ╳ S) ⨾ (S ╳ T) ≈ Q ╳ T
╳-iso-shift-left : ∀ {Q S T} → isBijective T → isMapping T → Q ╳ (T ⨾ S) ≈ (T ˘ ⨾ Q) ╳
╳-iso-shift-right : ∀ {Q S T} → isBijective T → isMapping T → (T ⨾ Q) ╳ S ≈ Q ╳ (T ˘ ⨾ S)
```

D.2. Interaction with Mappings and More

```
╳-in-left    : ∀ {Q S F} → isMapping F → F ⨾ (Q ╳ S) ≈ (Q ⨾ F ˘) ╳ S
╳-M-in-right : ∀ {Q S F} → isMapping F → (Q ╳ S) ⨾ F ˘ ≈ Q ╳ (S ⨾ F ˘)
noy-⊒-subidentity : ∀ {Q p} → isSubidentity p → p ⊑ (Q ╳ Q)
noy-isSubidentity : ∀ {Q}   → isUnivalent Q → isSurjective Q → isSubidentity (Q ╳ Q)
inj-╳-inj         : ∀ {Q S} → isInjective Q → isInjective S → Q ˘ ⨾ S ⊑ Q ╳ S
```

D.3. Interaction with Residuals

```
╳-⊑-\   : ∀ {Q S} → Q ╳ S ⊑ Q \ S
╳-⊑-/   : ∀ {Q S} → Q ╳ S ⊑ Q ˘ / S ˘
╳-⊑-\˘  : ∀ {Q S} → Q ╳ S ⊑ (S \ Q) ˘
˘╳-⊑-/  : ∀ {Q S} → Q ˘ ╳ S ⊑ Q / S ˘
╳˘-⊑-/  : ∀ {Q S} → Q ╳ S ˘ ⊑ Q ˘ / S
˘╳˘-⊑-/ : ∀ {Q S} → Q ˘ ╳ S ˘ ⊑ Q / S
⊑╳-from-\,/ : ∀ {Q S R} → R ⊑ Q \ S → R ⊑ Q ˘ / S ˘ → R ⊑ Q ╳ S
╳-isMeet    : ∀ {Q S} → IsMeet (Q \ S) (Q ˘ / S ˘) (Q ╳ S)
╳⊑\˘╳\˘ : {A B C : Obj} {Q : Mor A B} {S : Mor A C} → Q ╳ S ⊑ (Q \ S) ˘ ╳ (S \ S) ˘
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
```

D.4. Interaction with Identities

```
noy-isReflexive    : ∀ {R} → Id ⊑ R ╳ R
noy-isCoreflexive  : ∀ {R} → isUnivalentI R → isSurjectiveI R → R ╳ R ⊑ Id
noy-unival-surj-≈Id : ∀ {R} → isUnivalentI R → isSurjectiveI R → R ╳ R ≈ Id
noy-Id             : {A : Obj} → Id ╳ Id ≈ Id {A}
```