Properties of Orderings and Lattices

Georg Struth

March 17, 2025

Abstract

These components add further fundamental order and lattice-theoretic concepts and properties to Isabelle's libraries. They follow by and large the introductory sections of the *Compendium of Continuous Lattices*, covering directed and filtered sets, down-closed and up-closed sets, ideals and filters, Galois connections, closure and co-closure operators. Some emphasis is on duality and morphisms between structures—as in the Compendium. To this end, three ad-hoc approaches to duality are compared.

Contents

1 Introductory Remarks

Basic order- and lattice-theoretic concepts are well covered in Isabelle's libraries, and widely used. More advanced components are spread out over various sites (e.g. [11, 9, 8, 1, 4, 2]).

This formalisation takes the initial steps towards a modern structural approach to orderings and lattices, as for instance in denotational semantics of programs, algebraic logic or pointfree topology. Building on the components for orderings and lattices in Isabelle's main libraries, it follows the classical textbook *A Compendium of Continuous Lattices* [3] and, to a lesser extent, Johnstone's monograph on *Stone Spaces* [5]. By integrating material from other sources and extending it, a formalisation of undergraduate-level textbook material on orderings and lattices might eventually emerge.

In the textbooks mentioned, concepts such as dualities, isomorphisms between structures and relationships between categories are emphasised. These are essential to modern mathematics beyond orderings and lattices; their formalisation with interactive theorem provers is therefore of wider interest. Nevertheless such notions seem rather underexplored with Isabelle, and I am not aware of a standard way of modelling and using them. The present setting is perhaps the simplest one in which their formalisation can be studied.

These components use Isabelle's axiomatic approach without carrier sets. This is certainly a limitation, but it can be taken quite far. Yet well known facts such as Tarski's theorem—the set of fixpoints of an isotone endofunction on a complete lattice forms a complete lattice—seem hard to formalise with it (at least without using recent experimental extensions [7]).

Firstly, leaner versions of complete lattices are introduced: Sup-lattices (and their dual Inf-lattices), in which only Sups (or Infs) are axiomatised, whereas the remaining operators, which are axiomatised in the standard Isabelle class for complete lattices, are defined explicitly. This not only reduces of proof obligations in instantiation or interpretation proofs, it also helps in constructions where only suprema are represented faithfully (e.g. using morphisms that preserve sups, but not infs, or vice versa). At the moment, Sup-lattices remain rather loosely integrated into Isabelle's lattice hierarchy; a tighter one seems rather delicate.

Order and lattice duality is modelled, rather ad hoc, within a type class that can be added to those for orderings and lattices. Duality thus becomes a functor that reverses the order and maps Sups to Infs and vice versa, as expected. It also maps order-preserving functions to order-preserving functions, Sup-preserving to Inf-preserving ones and vice versa. This simple approach has not yet been optimised for automatic generation of dual statements (which seems hard to achieve anyway). It works quite well on simple examples.

The class-based approach to duality is contrasted by an implicit, locale-based one (which is quite standard in Isabelle), and Wenzel's data-type-based one [11]. Wenzel's approach generates many properties of the duality functor automatically from Isabelle's data type package. However, duality is not involutive, and this limits the dualisation of theorems quite severely. The local-based approach dualises theorems within the context of a type class or locale highly automatically. But, unlike the present approach, it is limited to such contexts. Yet another approach to duality has been taken in HOL-Algebra [2], but it is essentially based on set theory and therefore beyond the reach of simple axiomatic type classes.

The components presented also cover fundamental concepts such as directed and filtered sets, down-closed and up-closed sets, ideals and filters, notions of sup-closure and inf-closure, sup-preservation and inf-preservation, properties of adjunctions (or Galois connections) between orderings and (complete) lattices, fusion theorems for least and greatest fixpoints, and basic properties of closure and co-closure (kernel) operations, following the Compendium (most of these concepts come as dual pairs!). As in this monograph, emphasis lies on categorical aspects, but no formal category theory is used. In addition, some simple representation theorems have been formalised, including Stone's theorem for atomic boolean algebras (objects only). The non-atomic case seems possible, but is left for future work. Dealing with opposite maps properly, which is essential for dualities, remains an issue.

Finally, in Isabelle's main libraries, complete distributive lattices and complete boolean algebras are currently based on a very strong distributivity law, which makes these structures *completely distributive* and is basically an Axiom of Choice. While powerset algebras satisfy this law, other appli-

cations, for instance in topology require different axiomatisations. Complete boolean algebras, in particular, are usually defined as complete lattices which are also boolean algebras. Hence only a finite distributivity law holds. Weaker distributivity laws are also essential for axiomatising complete Heyting algebras (aka frames or locales), which are relevant for point-free topology [5].

Many questions remain, in particular on tighter integrations of duality and reasoning up to isomorphism with Isabelle and beyond. In its present form, duality is often not picked up in the proofs of more complex statements. Some statements from the Compendium and Johnstone's book had to be ignored due to the absence of carrier sets in Isabelle's standard components for orderings and lattices. Whether Kuncar and Popescu's new types-to-sets translation [7] provides a satisfactory solution remains to be seen.

2 Sup-Lattices and Other Simplifications

theory *Sup-Lattice*
  imports *Main*
begin

unbundle *lattice-syntax*

Some definitions for orderings and lattices in Isabelle could be simpler. The strict order in in ord could be defined instead of being axiomatised. The function mono could have been defined on ord and not on order—even on a general (di)graph it serves as a morphism. In complete lattices, the supremum—and dually the infimum—suffices to define the other operations (in the Isabelle/HOL-definition infimum, binary supremum and infimum, bottom and top element are axiomatised). This not only increases the number of proof obligations in subclass or sublocale statements, instantiations or interpretations, it also complicates situations where suprema are presented faithfully, e.g. mapped onto suprema in some subalgebra, whereas infima in the subalgebra are different from those in the super-structure.

It would be even nicer to use a class less-eq which dispenses with the strict order symbol in ord. Then one would not have to redefine this symbol in all instantiations or interpretations. At least, it does not carry any proof obligations.

context *ord*
begin

ub-set yields the set of all upper bounds of a set; lb-set the set of all lower bounds.

definition *ub-set* :: *'a set* ⇒ *'a set* **where**

ub-set $X = \{y.\ \forall x \in X.\ x \leq y\}$

definition *lb-set* :: *'a set* ⇒ *'a set* **where**
lb-set $X = \{y.\ \forall x \in X.\ y \leq x\}$

end

definition *ord-pres* :: (*'a*::*ord* ⇒ *'b*::*ord*) ⇒ *bool* **where**
ord-pres $f = (\forall x\ y.\ x \leq y \longrightarrow f\ x \leq f\ y)$

lemma *ord-pres-mono*:
fixes f :: *'a*::*order* ⇒ *'b*::*order*
shows *mono* f = *ord-pres* f
⟨*proof*⟩

class *preorder-lean* = *ord* +
assumes *preorder-refl*: $x \leq x$
and *preorder-trans*: $x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z$

begin

definition *le* :: *'a* ⇒ *'a* ⇒ *bool* **where**
le $x\ y = (x \leq y \wedge \neg\ (x \geq y))$

end

sublocale *preorder-lean* ⊆ *prel*: *preorder* (≤) *le*
⟨*proof*⟩

class *order-lean* = *preorder-lean* +
assumes *order-antisym*: $x \leq y \Longrightarrow x \geq y \Longrightarrow x = y$

sublocale *order-lean* ⊆ *posl*: *order* (≤) *le*
⟨*proof*⟩

class *Sup-lattice* = *order-lean* + *Sup* +
assumes *Sups-upper*: $x \in X \Longrightarrow x \leq \bigsqcup X$
and *Sups-least*: $(\bigwedge x.\ x \in X \Longrightarrow x \leq z) \Longrightarrow \bigsqcup X \leq z$

begin

definition *Infs* :: *'a set* ⇒ *'a* **where**
Infs $X = \bigsqcup\{y.\ \forall x \in X.\ y \leq x\}$

definition *sups* :: *'a* ⇒ *'a* ⇒ *'a* **where**
sups $x\ y = \bigsqcup\{x,y\}$

definition *infs* :: *'a* ⇒ *'a* ⇒ *'a* **where**
infs $x\ y$ = *Infs*$\{x,y\}$

definition *bots* :: *'a* **where**
bots = $\bigsqcup \{\}$

definition *tops* :: *'a* **where**
tops = *Infs*{}

lemma *Infs-prop*: *Infs* = *Sup* ∘ *lb-set*
⟨*proof*⟩

end

class *Inf-lattice* = *order-lean* + *Inf* +
assumes *Infi-lower*: $x \in X \Longrightarrow \bigsqcap X \leq x$
and *Infi-greatest*: $(\bigwedge x.\ x \in X \Longrightarrow z \leq x) \Longrightarrow z \leq \bigsqcap X$

begin

definition *Supi* :: *'a set* ⇒ *'a* **where**
Supi X = $\bigsqcap \{y.\ \forall x \in X.\ x \leq y\}$

definition *supi* :: *'a* ⇒ *'a* ⇒ *'a* **where**
supi x y = *Supi*{*x*,*y*}

definition *infi* :: *'a* ⇒ *'a* ⇒ *'a* **where**
infi x y = $\bigsqcap \{x,y\}$

definition *boti* :: *'a* **where**
boti = *Supi*{}

definition *topi* :: *'a* **where**
topi = $\bigsqcap \{\}$

lemma *Supi-prop*: *Supi* = *Inf* ∘ *ub-set*
⟨*proof*⟩

end

sublocale *Inf-lattice* ⊆ *ldual*: *Sup-lattice Inf* (≥)
rewrites *ldual.Infs* = *Supi*
and *ldual.infs* = *supi*
and *ldual.sups* = *infi*
and *ldual.tops* = *boti*
and *ldual.bots* = *topi*
⟨*proof*⟩

sublocale *Sup-lattice* ⊆ *supclat*: *complete-lattice Infs Sup-class.Sup infs* (≤) *le sups bots tops*
⟨*proof*⟩

sublocale *Inf-lattice* $\subseteq$ *infclat*: *complete-lattice Inf-class.Inf Supi infi* ($\leq$) *le supi boti topi*
⟨*proof*⟩

end

3 Ad-Hoc Duality for Orderings and Lattices

theory *Order-Duality*
imports *Sup-Lattice*

begin

This component presents an "explicit" formalisation of order and lattice duality. It augments the data type based one used by Wenzel in his lattice components [11], and complements the "implicit" formalisation given by locales. It uses a functor dual, supplied within a type class, which is simply a bijection (isomorphism) between types, with the constraint that the dual of a dual object is the original object. In Wenzel's formalisation, by contrast, dual is a bijection, but not idempotent or involutive. In the past, Preoteasa has used a similar approach with Isabelle [8].

Duality is such a fundamental concept in order and lattice theory that it probably deserves to be included in the type classes for these objects, as in this section.

class *dual* =
fixes *dual* :: $'a \Rightarrow 'a$ ($\langle\partial\rangle$)
assumes *inj-dual*: *inj* ∂
and *invol-dual* [*simp*]: $\partial \circ \partial = id$

This type class allows one to define a type dual. It is actually a dependent type for which dual can be instantiated.

typedef (**overloaded**) $'a$ *dual* = *range* (*dual*::$'a$::*dual* $\Rightarrow$ $'a$)
⟨*proof*⟩

setup-lifting *type-definition-dual*

At the moment I have no use for this type.

context *dual*
begin

lemma *invol-dual-var* [*simp*]: $\partial\ (\partial\ x) = x$
⟨*proof*⟩

lemma *surj-dual*: *surj* ∂
⟨*proof*⟩

lemma *bij-dual*: *bij* ∂
⟨*proof*⟩

lemma *inj-dual-iff*: $(\partial\ x = \partial\ y) = (x = y)$
⟨*proof*⟩

lemma *dual-iff*: $(\partial\ x = y) = (x = \partial\ y)$
⟨*proof*⟩

lemma *the-inv-dual*: *the-inv* $\partial = \partial$
⟨*proof*⟩

end

In boolean algebras, duality is of course De Morgan duality and can be expressed within the language.

sublocale *boolean-algebra* $\subseteq$ *ba-dual*: *dual uminus*
⟨*proof*⟩

definition *map-dual*:: $('a \Rightarrow 'b) \Rightarrow 'a$::*dual* $\Rightarrow 'b$::*dual* $(\langle\partial_F\rangle)$ **where**
$\partial_F\ f = \partial \circ f \circ \partial$

lemma *map-dual-func1*: $\partial_F\ (f \circ g) = \partial_F\ f \circ \partial_F\ g$
⟨*proof*⟩

lemma *map-dual-func2* [*simp*]: $\partial_F\ id = id$
⟨*proof*⟩

lemma *map-dual-nat-iso*: $\partial_F\ f \circ \partial = \partial \circ id\ f$
⟨*proof*⟩

lemma *map-dual-invol* [*simp*]: $\partial_F \circ \partial_F = id$
⟨*proof*⟩

Thus map-dual is naturally isomorphic to the identify functor: The function dual is a natural transformation between map-dual and the identity functor, and, because it has a two-sided inverse — itself, it is a natural isomorphism.

The generic function set-dual provides another natural transformation (see below). Before introducing it, we introduce useful notation for a widely used function.

abbreviation $\eta \equiv (\lambda x.\ \{x\})$

lemma *eta-inj*: *inj* η
⟨*proof*⟩

definition *set-dual* $= \eta \circ \partial$

lemma *set-dual-prop*: *set-dual* $(\partial\ x) = \{x\}$
⟨*proof*⟩

The next four lemmas show that (functional) image and preimage are functors (on functions). This does not really belong here, but it is useful for what follows. The interaction between duality and (pre)images is needed in applications.

lemma *image-func1*: $(`)\ (f \circ g) = (`)\ f \circ (`)\ g$
⟨*proof*⟩

lemma *image-func2*: $(`)\ id = id$
⟨*proof*⟩

lemma *vimage-func1*: $(-`)\ (f \circ g) = (-`)\ g \circ (-`)\ f$
⟨*proof*⟩

lemma *vimage-func2*: $(-`)\ id = id$
⟨*proof*⟩

lemma *iso-image*: *mono* $((`)\ f)$
⟨*proof*⟩

lemma *iso-preimage*: *mono* $((-`)\ f)$
⟨*proof*⟩

context *dual*
begin

lemma *image-dual* [*simp*]: $(`)\ \partial \circ (`)\ \partial = id$
⟨*proof*⟩

lemma *vimage-dual* [*simp*]: $(-`)\ \partial \circ (-`)\ \partial = id$
⟨*proof*⟩

end

The following natural transformation between the powerset functor (image) and the identity functor is well known.

lemma *power-set-func-nat-trans*: $\eta \circ id\ f = (`)\ f \circ \eta$
⟨*proof*⟩

As an instance, set-dual is a natural transformation with built-in type coercion.

lemma *dual-singleton*: $(`)\ \partial \circ \eta = \eta \circ \partial$
⟨*proof*⟩

lemma *finite-dual* [*simp*]: *finite* $\circ\ (`)\ \partial =$ *finite*
⟨*proof*⟩

lemma *finite-dual-var* [*simp*]: *finite* (∂ ‘ X) = *finite* X
⟨*proof*⟩

lemma *subset-dual*: (X = ∂ ‘ Y) = (∂ ‘ X = Y)
⟨*proof*⟩

lemma *subset-dual1*: (X ⊆ Y) = (∂ ‘ X ⊆ ∂ ‘ Y)
⟨*proof*⟩

lemma *dual-empty* [*simp*]: ∂ ‘ {} = {}
⟨*proof*⟩

lemma *dual-UNIV* [*simp*]: ∂ ‘ *UNIV* = *UNIV*
⟨*proof*⟩

lemma *fun-dual1*: (f = g ∘ ∂) = (f ∘ ∂ = g)
⟨*proof*⟩

lemma *fun-dual2*: (f = ∂ ∘ g) = (∂ ∘ f = g)
⟨*proof*⟩

lemma *fun-dual3*: (f = g ∘ (‘) ∂) = (f ∘ (‘) ∂ = g)
⟨*proof*⟩

lemma *fun-dual4*: (f = (‘) ∂ ∘ g) = ((‘) ∂ ∘ f = g)
⟨*proof*⟩

lemma *fun-dual5*: (f = ∂ ∘ g ∘ ∂) = (∂ ∘ f ∘ ∂ = g)
⟨*proof*⟩

lemma *fun-dual6*: (f = (‘) ∂ ∘ g ∘ (‘) ∂) = ((‘) ∂ ∘ f ∘ (‘) ∂ = g)
⟨*proof*⟩

lemma *fun-dual7*: (f = ∂ ∘ g ∘ (‘) ∂) = (∂ ∘ f ∘ (‘) ∂ = g)
⟨*proof*⟩

lemma *fun-dual8*: (f = (‘) ∂ ∘ g ∘ ∂) = ((‘) ∂ ∘ f ∘ ∂ = g)
⟨*proof*⟩

lemma *map-dual-dual*: (∂_F f = g) = (∂_F g = f)
⟨*proof*⟩

The next facts show incrementally that the dual of a complete lattice is a complete lattice.

class *ord-with-dual* = *dual* + *ord* +
assumes *ord-dual*: $x \leq y \Longrightarrow \partial\ y \leq \partial\ x$

begin

lemma *dual-dual-ord*: $(\partial\ x \leq \partial\ y) = (y \leq x)$
⟨*proof*⟩

end

lemma *ord-pres-dual*:
fixes $f :: 'a{::}ord\text{-}with\text{-}dual \Rightarrow 'b{::}ord\text{-}with\text{-}dual$
shows *ord-pres* $f \Longrightarrow$ *ord-pres* $(\partial_F\ f)$
⟨*proof*⟩

lemma *map-dual-anti*: $(f{::}'a{::}ord\text{-}with\text{-}dual \Rightarrow 'b{::}ord\text{-}with\text{-}dual) \leq g \implies \partial_F\ g \leq \partial_F\ f$
⟨*proof*⟩

class *preorder-with-dual* = *ord-with-dual* + *preorder*

begin

lemma *less-dual-def-var*: $(\partial\ y < \partial\ x) = (x < y)$
⟨*proof*⟩

end

class *order-with-dual* = *preorder-with-dual* + *order*

lemma *iso-map-dual*:
fixes $f :: 'a{::}order\text{-}with\text{-}dual \Rightarrow 'b{::}order\text{-}with\text{-}dual$
shows *mono* $f \Longrightarrow$ *mono* $(\partial_F\ f)$
⟨*proof*⟩

class *lattice-with-dual* = *lattice* + *dual* +
assumes *sup-dual-def*: $\partial\ (x \sqcup y) = \partial\ x \sqcap \partial\ y$

begin

subclass *order-with-dual*
⟨*proof*⟩

lemma *inf-dual*: $\partial\ (x \sqcap y) = \partial\ x \sqcup \partial\ y$
⟨*proof*⟩

lemma *inf-to-sup*: $x \sqcap y = \partial\ (\partial\ x \sqcup \partial\ y)$
⟨*proof*⟩

lemma *sup-to-inf*: $x \sqcup y = \partial\ (\partial\ x \sqcap \partial\ y)$
⟨*proof*⟩

end

class *bounded-lattice-with-dual* = *lattice-with-dual* + *bounded-lattice*

begin

lemma *bot-dual*: $\partial\ \bot = \top$
⟨*proof*⟩

lemma *top-dual*: $\partial\ \top = \bot$
⟨*proof*⟩

end

class *boolean-algebra-with-dual* = *lattice-with-dual* + *boolean-algebra*

sublocale *boolean-algebra* ⊆ *badual*: *boolean-algebra-with-dual* - - - - - - - - *uminus*
⟨*proof*⟩

class *Sup-lattice-with-dual* = *Sup-lattice* + *dual* +
assumes *Sups-dual-def*: $\partial \circ Sup = Infs \circ (`)\ \partial$

class *Inf-lattice-with-dual* = *Inf-lattice* + *dual* +
assumes *Sups-dual-def*: $\partial \circ Supi = Inf \circ (`)\ \partial$

class *complete-lattice-with-dual* = *complete-lattice* + *dual* +
assumes *Sups-dual-def*: $\partial \circ Sup = Inf \circ (`)\ \partial$

sublocale *Sup-lattice-with-dual* ⊆ *sclatd*: *complete-lattice-with-dual Infs Sup infs* (≤) *le sups bots tops* ∂
⟨*proof*⟩

sublocale *Inf-lattice-with-dual* ⊆ *iclatd*: *complete-lattice-with-dual Inf Supi infi* (≤) *le supi boti topi* ∂
⟨*proof*⟩

context *complete-lattice-with-dual*
begin

lemma *Inf-dual*: $\partial \circ Inf = Sup \circ (`)\ \partial$
⟨*proof*⟩

lemma *Inf-dual-var*: $\partial\ (\bigsqcap X) = \bigsqcup (\partial\ `\ X)$
⟨*proof*⟩

lemma *Inf-to-Sup*: $Inf = \partial \circ Sup \circ (`)\ \partial$
⟨*proof*⟩

lemma *Inf-to-Sup-var*: $\bigsqcap X = \partial\ (\bigsqcup (\partial\ `\ X))$
⟨*proof*⟩

lemma *Sup-to-Inf*: $Sup = \partial \circ Inf \circ (`) \partial$
⟨*proof*⟩

lemma *Sup-to-Inf-var*: $\bigsqcup X = \partial\ (\bigsqcap (\partial\ `\ X))$
⟨*proof*⟩

lemma *Sup-dual-def-var*: $\partial\ (\bigsqcup X) = \bigsqcap\ (\partial\ `\ X)$
⟨*proof*⟩

lemma *bot-dual-def*: $\partial\ \top = \bot$
⟨*proof*⟩

lemma *top-dual-def*: $\partial\ \bot = \top$
⟨*proof*⟩

lemma *inf-dual2*: $\partial\ (x \sqcap y) = \partial\ x \sqcup \partial\ y$
⟨*proof*⟩

lemma *sup-dual*: $\partial\ (x \sqcup y) = \partial\ x \sqcap \partial\ y$
⟨*proof*⟩

subclass *lattice-with-dual*
⟨*proof*⟩

subclass *bounded-lattice-with-dual*⟨*proof*⟩

end

end

4 Properties of Orderings and Lattices

theory *Order-Lattice-Props*
imports *Order-Duality*

begin

4.1 Basic Definitions for Orderings and Lattices

The first definition is for order morphisms — isotone (order-preserving, monotone) functions. An order isomorphism is an order-preserving bijection. This should be defined in the class ord, but mono requires order.

definition *ord-homset* :: ('*a*::*order* ⇒ '*b*::*order*) *set* **where**
ord-homset = {*f*::'*a*::*order* ⇒ '*b*::*order*. *mono f*}

definition *ord-embed* :: ('*a*::*order* ⇒ '*b*::*order*) ⇒ *bool* **where**
ord-embed f = (∀ *x y*. *f x* ≤ *f y* ⟷ *x* ≤ *y*)

definition *ord-iso* :: (*'a*::*order* ⇒ *'b*::*order*) ⇒ *bool* **where**
ord-iso = *bij* ⊓ *mono* ⊓ (*mono* ∘ *the-inv*)

lemma *ord-embed-alt*: *ord-embed f* = (*mono f* ∧ (∀ *x y*. *f x* ≤ *f y* ⟶ *x* ≤ *y*))
⟨*proof*⟩

lemma *ord-embed-homset*: *ord-embed f* ⟹ *f* ∈ *ord-homset*
⟨*proof*⟩

lemma *ord-embed-inj*: *ord-embed f* ⟹ *inj f*
⟨*proof*⟩

lemma *ord-iso-ord-embed*: *ord-iso f* ⟹ *ord-embed f*
⟨*proof*⟩

lemma *ord-iso-alt*: *ord-iso f* = (*ord-embed f* ∧ *surj f*)
⟨*proof*⟩

lemma *ord-iso-the-inv*: *ord-iso f* ⟹ *mono* (*the-inv f*)
⟨*proof*⟩

lemma *ord-iso-inv1*: *ord-iso f* ⟹ (*the-inv f*) ∘ *f* = *id*
⟨*proof*⟩

lemma *ord-iso-inv2*: *ord-iso f* ⟹ *f* ∘ (*the-inv f*) = *id*
⟨*proof*⟩

typedef (**overloaded**) (*'a*,*'b*) *ord-homset* = *ord-homset*::(*'a*::*order* ⇒ *'b*::*order*) *set*
⟨*proof*⟩

setup-lifting *type-definition-ord-homset*

The next definition is for the set of fixpoints of a given function. It is important in the context of orders, for instance for proving Tarski's fixpoint theorem, but does not really belong here.

definition *Fix* :: (*'a* ⇒ *'a*) ⇒ *'a set* **where**
Fix f = {*x*. *f x* = *x*}

lemma *retraction-prop*: *f* ∘ *f* = *f* ⟹ *f x* = *x* ⟷ *x* ∈ *range f*
⟨*proof*⟩

lemma *retraction-prop-fix*: *f* ∘ *f* = *f* ⟹ *range f* = *Fix f*
⟨*proof*⟩

lemma *Fix-map-dual*: *Fix* ∘ ∂$_F$ = (‘) ∂ ∘ *Fix*
⟨*proof*⟩

lemma *Fix-map-dual-var*: $Fix\ (\partial_F\ f) = \partial\ `\ (Fix\ f)$
⟨*proof*⟩

lemma *gfp-dual*: $(\partial::'a::complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'a) \circ gfp = lfp \circ \partial_F$
⟨*proof*⟩

lemma *gfp-dual-var*:
fixes $f :: 'a::complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'a$
shows $\partial\ (gfp\ f) = lfp\ (\partial_F\ f)$
⟨*proof*⟩

lemma *gfp-to-lfp*: $gfp = (\partial::'a::complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'a) \circ lfp \circ \partial_F$
⟨*proof*⟩

lemma *gfp-to-lfp-var*:
fixes $f :: 'a::complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'a$
shows $gfp\ f = \partial\ (lfp\ (\partial_F\ f))$
⟨*proof*⟩

lemma *lfp-dual*: $(\partial::'a::complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'a) \circ lfp = gfp \circ \partial_F$
⟨*proof*⟩

lemma *lfp-dual-var*:
fixes $f :: 'a::complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'a$
shows $\partial\ (lfp\ f) = gfp\ (map\text{-}dual\ f)$
⟨*proof*⟩

lemma *lfp-to-gfp*: $lfp = (\partial::'a::complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'a) \circ gfp \circ \partial_F$
⟨*proof*⟩

lemma *lfp-to-gfp-var*:
fixes $f :: 'a::complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'a$
shows $lfp\ f = \partial\ (gfp\ (\partial_F\ f))$
⟨*proof*⟩

lemma *lfp-in-Fix*:
fixes $f :: 'a::complete\text{-}lattice \Rightarrow 'a$
shows $mono\ f \Longrightarrow lfp\ f \in Fix\ f$
⟨*proof*⟩

lemma *gfp-in-Fix*:
fixes $f :: 'a::complete\text{-}lattice \Rightarrow 'a$
shows $mono\ f \Longrightarrow gfp\ f \in Fix\ f$
⟨*proof*⟩

lemma *nonempty-Fix*:
fixes $f :: 'a::complete\text{-}lattice \Rightarrow 'a$
shows $mono\ f \Longrightarrow Fix\ f \neq \{\}$
⟨*proof*⟩

Next the minimal and maximal elements of an ordering are defined.

context *ord*
begin

definition *min-set* :: *'a set* ⇒ *'a set* **where**
min-set X = {*y* ∈ *X*. ∀ *x* ∈ *X*. *x* ≤ *y* ⟶ *x* = *y*}

definition *max-set* :: *'a set* ⇒ *'a set* **where**
max-set X = {*x* ∈ *X*. ∀ *y* ∈ *X*. *x* ≤ *y* ⟶ *x* = *y*}

end

context *ord-with-dual*
begin

lemma *min-max-set-dual*: (‘) ∂ ∘ *min-set* = *max-set* ∘ (‘) ∂
⟨*proof*⟩

lemma *min-max-set-dual-var*: ∂ ‘ (*min-set X*) = *max-set* (∂ ‘ *X*)
⟨*proof*⟩

lemma *max-min-set-dual*: (‘) ∂ ∘ *max-set* = *min-set* ∘ (‘) ∂
⟨*proof*⟩

lemma *min-to-max-set*: *min-set* = (‘) ∂ ∘ *max-set* ∘ (‘) ∂
⟨*proof*⟩

lemma *max-min-set-dual-var*: ∂ ‘ (*max-set X*) = *min-set* (∂ ‘ *X*)
⟨*proof*⟩

lemma *min-to-max-set-var*: *min-set X* = ∂ ‘ (*max-set* (∂ ‘ *X*))
⟨*proof*⟩

end

Next, directed and filtered sets, upsets, downsets, filters and ideals in posets are defined.

context *ord*
begin

definition *directed* :: *'a set* ⇒ *bool* **where**
directed X = (∀ *Y*. *finite Y* ∧ *Y* ⊆ *X* ⟶ (∃ *x* ∈ *X*. ∀ *y* ∈ *Y*. *y* ≤ *x*))

definition *filtered* :: *'a set* ⇒ *bool* **where**
filtered X = (∀ *Y*. *finite Y* ∧ *Y* ⊆ *X* ⟶ (∃ *x* ∈ *X*. ∀ *y* ∈ *Y*. *x* ≤ *y*))

definition *downset-set* :: *'a set* ⇒ *'a set* (‹⇓›) **where**
⇓*X* = {*y*. ∃ *x* ∈ *X*. *y* ≤ *x*}

definition *upset-set* :: *'a set* ⇒ *'a set* (‹⇑›) **where**
⇑X = {y. ∃ x ∈ X. x ≤ y}

definition *downset* :: *'a* ⇒ *'a set* (‹↓›) **where**
↓ = ⇓ ∘ η

definition *upset* :: *'a* ⇒ *'a set* (‹↑›) **where**
↑ = ⇑ ∘ η

definition *downsets* :: *'a set set* **where**
downsets = *Fix* ⇓

definition *upsets* :: *'a set set* **where**
upsets = *Fix* ⇑

definition *downclosed-set* X = (X ∈ *downsets*)

definition *upclosed-set* X = (X ∈ *upsets*)

definition *ideals* :: *'a set set* **where**
ideals = {X. X ≠ {} ∧ *downclosed-set* X ∧ *directed* X}

definition *filters* :: *'a set set* **where**
filters = {X. X ≠ {} ∧ *upclosed-set* X ∧ *filtered* X}

abbreviation *idealp* X ≡ X ∈ *ideals*

abbreviation *filterp* X ≡ X ∈ *filters*

end

These notions are pair-wise dual.

Filtered and directed sets are dual.

context *ord-with-dual*
begin

lemma *filtered-directed-dual*: *filtered* ∘ (‘) ∂ = *directed*
⟨*proof*⟩

lemma *directed-filtered-dual*: *directed* ∘ (‘) ∂ = *filtered*
⟨*proof*⟩

lemma *filtered-to-directed*: *filtered* X = *directed* (∂ ‘ X)
⟨*proof*⟩

Upsets and downsets are dual.

lemma *downset-set-upset-set-dual*: (‘) ∂ ∘ ⇓ = ⇑ ∘ (‘) ∂
⟨*proof*⟩

lemma *upset-set-downset-set-dual*: (‘) $\partial \circ \Uparrow = \Downarrow \circ$ (‘) ∂
⟨*proof*⟩

lemma *upset-set-to-downset-set*: $\Uparrow$ = (‘) $\partial \circ \Downarrow \circ$ (‘) ∂
⟨*proof*⟩

lemma *upset-set-to-downset-set2*: $\Uparrow X = \partial$ ‘ ($\Downarrow$ (∂ ‘ X))
⟨*proof*⟩

lemma *downset-upset-dual*: (‘) $\partial \circ \downarrow = \uparrow \circ \partial$
⟨*proof*⟩

lemma *upset-to-downset*: (‘) $\partial \circ \uparrow = \downarrow \circ \partial$
⟨*proof*⟩

lemma *upset-to-downset2*: $\uparrow$ = (‘) $\partial \circ \downarrow \circ \partial$
⟨*proof*⟩

lemma *upset-to-downset3*: $\uparrow x = \partial$ ‘ ($\downarrow$ (∂ x))
⟨*proof*⟩

lemma *downsets-upsets-dual*: ($X \in$ *downsets*) = (∂ ‘ $X \in$ *upsets*)
⟨*proof*⟩

lemma *downset-setp-upset-setp-dual*: *upclosed-set* $\circ$ (‘) ∂ = *downclosed-set*
⟨*proof*⟩

lemma *upsets-to-downsets*: ($X \in$ *upsets*) = (∂ ‘ $X \in$ *downsets*)
⟨*proof*⟩

lemma *upset-setp-downset-setp-dual*: *downclosed-set* $\circ$ (‘) ∂ = *upclosed-set*
⟨*proof*⟩

Filters and ideals are dual.

lemma *ideals-filters-dual*: ($X \in$ *ideals*) = ((∂ ‘ X) $\in$ *filters*)
⟨*proof*⟩

lemma *idealp-filterp-dual*: *idealp* = *filterp* $\circ$ (‘) ∂
⟨*proof*⟩

lemma *filters-to-ideals*: ($X \in$ *filters*) = ((∂ ‘ X) $\in$ *ideals*)
⟨*proof*⟩

lemma *filterp-idealp-dual*: *filterp* = *idealp* $\circ$ (‘) ∂
⟨*proof*⟩

end

4.2 Properties of Orderings

context *ord*
begin

lemma *directed-nonempty*: *directed* $X \Longrightarrow X \neq \{\}$
⟨*proof*⟩

lemma *directed-ub*: *directed* $X \Longrightarrow (\forall\, x \in X.\ \forall\, y \in X.\ \exists\, z \in X.\ x \leq z \wedge y \leq z)$
⟨*proof*⟩

lemma *downset-set-prop*: $\Downarrow = \mathit{Union} \circ (`) \downarrow$
⟨*proof*⟩

lemma *downset-set-prop-var*: $\Downarrow X = (\bigcup x \in X.\ \downarrow x)$
⟨*proof*⟩

lemma *downset-prop*: $\downarrow x = \{y.\ y \leq x\}$
⟨*proof*⟩

lemma *downset-prop2*: $y \leq x \Longrightarrow y \in \downarrow x$
⟨*proof*⟩

lemma *ideals-downsets*: $X \in \mathit{ideals} \Longrightarrow X \in \mathit{downsets}$
⟨*proof*⟩

lemma *ideals-directed*: $X \in \mathit{ideals} \Longrightarrow \mathit{directed}\ X$
⟨*proof*⟩

end

context *preorder*
begin

lemma *directed-prop*: $X \neq \{\} \Longrightarrow (\forall\, x \in X.\ \forall\, y \in X.\ \exists\, z \in X.\ x \leq z \wedge y \leq z)$
$\Longrightarrow \mathit{directed}\ X$
⟨*proof*⟩

lemma *directed-alt*: $\mathit{directed}\ X = (X \neq \{\} \wedge (\forall\, x \in X.\ \forall\, y \in X.\ \exists\, z \in X.\ x \leq z$
$\wedge\ y \leq z))$
⟨*proof*⟩

lemma *downset-set-prop-var2*: $x \in \Downarrow X \Longrightarrow y \leq x \Longrightarrow y \in \Downarrow X$
⟨*proof*⟩

lemma *downclosed-set-iff*: $\mathit{downclosed\text{-}set}\ X = (\forall\, x \in X.\ \forall\, y.\ y \leq x \longrightarrow y \in X)$
⟨*proof*⟩

lemma *downclosed-downset-set*: $\mathit{downclosed\text{-}set}\ (\Downarrow X)$
⟨*proof*⟩

lemma *downclosed-downset*: *downclosed-set* $(\downarrow x)$
⟨*proof*⟩

lemma *downset-set-ext*: $id \leq \Downarrow$
⟨*proof*⟩

lemma *downset-set-iso*: *mono* $\Downarrow$
⟨*proof*⟩

lemma *downset-set-idem* [*simp*]: $\Downarrow \circ \Downarrow = \Downarrow$
⟨*proof*⟩

lemma *downset-faithful*: $\downarrow x \subseteq \downarrow y \Longrightarrow x \leq y$
⟨*proof*⟩

lemma *downset-iso-iff*: $(\downarrow x \subseteq \downarrow y) = (x \leq y)$
⟨*proof*⟩

The following proof uses the Axiom of Choice.

lemma *downset-directed-downset-var* [*simp*]: *directed* $(\Downarrow X)$ = *directed* X
⟨*proof*⟩

lemma *downset-directed-downset* [*simp*]: *directed* $\circ \Downarrow$ = *directed*
⟨*proof*⟩

lemma *directed-downset-ideals*: *directed* $(\Downarrow X) = (\Downarrow X \in ideals)$
⟨*proof*⟩

lemma *downclosed-Fix*: *downclosed-set* $X = (\Downarrow X = X)$
⟨*proof*⟩

end

lemma *downset-iso*: *mono* $(\downarrow :: 'a :: order \Rightarrow 'a\ set)$
⟨*proof*⟩

lemma *mono-downclosed*:
fixes $f :: 'a :: order \Rightarrow 'b :: order$
assumes *mono* f
shows $\forall Y.$ *downclosed-set* $Y \longrightarrow$ *downclosed-set* $(f -` Y)$
⟨*proof*⟩

lemma
fixes $f :: 'a :: order \Rightarrow 'b :: order$
assumes *mono* f
shows $\forall Y.$ *downclosed-set* $X \longrightarrow$ *downclosed-set* $(f ` X)$
⟨*proof*⟩

lemma *downclosed-mono*:
fixes $f :: 'a::order \Rightarrow 'b::order$
assumes $\forall\, Y.\ downclosed\text{-}set\ Y \longrightarrow downclosed\text{-}set\ (f\ -`\ Y)$
shows *mono f*
⟨*proof*⟩

lemma *mono-downclosed-iff*: *mono f* = ($\forall\, Y.\ downclosed\text{-}set\ Y \longrightarrow downclosed\text{-}set\ (f\ -`\ Y)$)
⟨*proof*⟩

context *order*
begin

lemma *downset-inj*: *inj* $\downarrow$
⟨*proof*⟩

lemma $(X \subseteq Y) = (\Downarrow X \subseteq \Downarrow Y)$
⟨*proof*⟩

end

context *lattice*
begin

lemma *lat-ideals*: $X \in ideals = (X \neq \{\} \wedge X \in downsets \wedge (\forall\, x \in X.\ \forall\ y \in X.\ x \sqcup y \in X))$
⟨*proof*⟩

end

context *bounded-lattice*
begin

lemma *bot-ideal*: $X \in ideals \Longrightarrow \bot \in X$
⟨*proof*⟩

end

context *complete-lattice*
begin

lemma *Sup-downset-id* [*simp*]: *Sup* $\circ \downarrow = id$
⟨*proof*⟩

lemma *downset-Sup-id*: $id \leq \downarrow \circ$ *Sup*
⟨*proof*⟩

lemma *Inf-Sup-var*: $\bigsqcup(\bigcap x \in X.\ \downarrow x) = \bigsqcap X$

⟨*proof*⟩

lemma *Inf-pres-downset-var*: $(\bigcap x \in X.\ {\downarrow}x) = {\downarrow}(\bigsqcap X)$
⟨*proof*⟩

end

4.3 Dual Properties of Orderings

context *ord-with-dual*
begin

lemma *filtered-nonempty*: *filtered* $X \Longrightarrow X \neq \{\}$
⟨*proof*⟩

lemma *filtered-lb*: *filtered* $X \Longrightarrow (\forall x \in X.\ \forall y \in X.\ \exists z \in X.\ z \leq x \wedge z \leq y)$
⟨*proof*⟩

lemma *upset-set-prop-var*: $\Uparrow X = (\bigcup x \in X.\ \uparrow x)$
⟨*proof*⟩

lemma *upset-set-prop*: $\Uparrow = \mathit{Union} \circ (`) \uparrow$
⟨*proof*⟩

lemma *upset-prop*: $\uparrow x = \{y.\ x \leq y\}$
⟨*proof*⟩

lemma *upset-prop2*: $x \leq y \Longrightarrow y \in \uparrow x$
⟨*proof*⟩

lemma *filters-upsets*: $X \in \mathit{filters} \Longrightarrow X \in \mathit{upsets}$
⟨*proof*⟩

lemma *filters-filtered*: $X \in \mathit{filters} \Longrightarrow \mathit{filtered}\ X$
⟨*proof*⟩

end

context *preorder-with-dual*
begin

lemma *filtered-prop*: $X \neq \{\} \Longrightarrow (\forall x \in X.\ \forall y \in X.\ \exists z \in X.\ z \leq x \wedge z \leq y) \Longrightarrow$ *filtered* X
⟨*proof*⟩

lemma *filtered-alt*: *filtered* $X = (X \neq \{\} \wedge (\forall x \in X.\ \forall y \in X.\ \exists z \in X.\ z \leq x \wedge z \leq y))$
⟨*proof*⟩

lemma *up-set-prop-var2*: $x \in \Uparrow X \Longrightarrow x \leq y \Longrightarrow y \in \Uparrow X$
⟨*proof*⟩

lemma *upclosed-set-iff*: *upclosed-set* $X = (\forall\, x \in X.\ \forall\, y.\ x \leq y \longrightarrow y \in X)$
⟨*proof*⟩

lemma *upclosed-upset-set*: *upclosed-set* $(\Uparrow X)$
⟨*proof*⟩

lemma *upclosed-upset*: *upclosed-set* $(\uparrow x)$
⟨*proof*⟩

lemma *upset-set-ext*: $id \leq \Uparrow$
⟨*proof*⟩

lemma *upset-set-anti*: *mono* $\Uparrow$
⟨*proof*⟩

lemma *up-set-idem* [*simp*]: $\Uparrow \circ \Uparrow = \Uparrow$
⟨*proof*⟩

lemma *upset-faithful*: $\uparrow x \subseteq \uparrow y \Longrightarrow y \leq x$
⟨*proof*⟩

lemma *upset-anti-iff*: $(\uparrow y \subseteq \uparrow x) = (x \leq y)$
⟨*proof*⟩

lemma *upset-filtered-upset* [*simp*]: *filtered* $\circ \Uparrow$ = *filtered*
⟨*proof*⟩

lemma *filtered-upset-filters*: *filtered* $(\Uparrow X) = (\Uparrow X \in$ *filters*$)$
⟨*proof*⟩

lemma *upclosed-Fix*: *upclosed-set* $X = (\Uparrow X = X)$
⟨*proof*⟩

end

lemma *upset-anti*: *antimono* $(\uparrow :: 'a :: order\text{-}with\text{-}dual \Rightarrow 'a\ set)$
⟨*proof*⟩

lemma *mono-upclosed*:
fixes $f :: 'a :: order\text{-}with\text{-}dual \Rightarrow 'b :: order\text{-}with\text{-}dual$
assumes *mono f*
shows $\forall\, Y.$ *upclosed-set* $Y \longrightarrow$ *upclosed-set* $(f\ -`\ Y)$
⟨*proof*⟩

lemma *mono-upclosed*:
fixes $f :: 'a :: order\text{-}with\text{-}dual \Rightarrow 'b :: order\text{-}with\text{-}dual$

assumes *mono f*
shows $\forall\, Y.$ *upclosed-set X* $\longrightarrow$ *upclosed-set* (*f* ‘ *X*)
⟨*proof*⟩

lemma *upclosed-mono*:
fixes *f* :: *'a*::*order-with-dual* ⇒ *'b*::*order-with-dual*
assumes $\forall\, Y.$ *upclosed-set Y* $\longrightarrow$ *upclosed-set* (*f* −‘ *Y*)
shows *mono f*
⟨*proof*⟩

lemma *mono-upclosed-iff*:
fixes *f* :: *'a*::*order-with-dual* ⇒ *'b*::*order-with-dual*
shows *mono f* = ($\forall\, Y.$ *upclosed-set Y* $\longrightarrow$ *upclosed-set* (*f* −‘ *Y*))
⟨*proof*⟩

context *order-with-dual*
begin

lemma *upset-inj*: *inj* ↑
⟨*proof*⟩

lemma (*X* ⊆ *Y*) = (⇑*Y* ⊆ ⇑*X*)
⟨*proof*⟩

end

context *lattice-with-dual*
begin

lemma *lat-filters*: *X* ∈ *filters* = (*X* ≠ {} ∧ *X* ∈ *upsets* ∧ (∀ *x* ∈ *X*. ∀ *y* ∈ *X*. *x* ⊓ *y* ∈ *X*))
⟨*proof*⟩

end

context *bounded-lattice-with-dual*
begin

lemma *top-filter*: *X* ∈ *filters* ⟹ ⊤ ∈ *X*
⟨*proof*⟩

end

context *complete-lattice-with-dual*
begin

lemma *Inf-upset-id* [*simp*]: *Inf* ∘ ↑ = *id*
⟨*proof*⟩

lemma *upset-Inf-id*: $id \leq \uparrow \circ Inf$
⟨*proof*⟩

lemma *Sup-Inf-var*: $\bigsqcap(\bigcap x \in X.\ \uparrow x) = \bigsqcup X$
⟨*proof*⟩

lemma *Sup-dual-upset-var*: $(\bigcap x \in X.\ \uparrow x) = \uparrow(\bigsqcup X)$
⟨*proof*⟩

end

4.4 Properties of Complete Lattices

definition *Inf-closed-set* $X = (\forall\, Y \subseteq X.\ \bigsqcap Y \in X)$

definition *Sup-closed-set* $X = (\forall\, Y \subseteq X.\ \bigsqcup Y \in X)$

definition *inf-closed-set* $X = (\forall\, x \in X.\ \forall\, y \in X.\ x \sqcap y \in X)$

definition *sup-closed-set* $X = (\forall\, x \in X.\ \forall\, y \in X.\ x \sqcup y \in X)$

The following facts about complete lattices add to those in the Isabelle libraries.

context *complete-lattice*
begin

The translation between sup and Sup could be improved. The sup-theorems should be direct consequences of Sup-ones. In addition, duality between sup and inf is currently not exploited.

lemma *sup-Sup*: $x \sqcup y = \bigsqcup\{x,y\}$
⟨*proof*⟩

lemma *inf-Inf*: $x \sqcap y = \bigsqcap\{x,y\}$
⟨*proof*⟩

The next two lemmas are about Sups and Infs of indexed families. These are interesting for iterations and fixpoints.

lemma *fSup-unfold*: $(f{::}nat \Rightarrow 'a)\ 0 \sqcup (\bigsqcup n.\ f\ (Suc\ n)) = (\bigsqcup n.\ f\ n)$
⟨*proof*⟩

lemma *fInf-unfold*: $(f{::}nat \Rightarrow 'a)\ 0 \sqcap (\bigsqcap n.\ f\ (Suc\ n)) = (\bigsqcap n.\ f\ n)$
⟨*proof*⟩

end

lemma *Sup-sup-closed*: *Sup-closed-set* $(X{::}'a{::}complete\text{-}lattice\ set) \Longrightarrow$ *sup-closed-set* X
⟨*proof*⟩

lemma *Inf-inf-closed*: *Inf-closed-set* (X::'a::*complete-lattice set*) $\Longrightarrow$ *inf-closed-set* X
⟨*proof*⟩

4.5 Sup- and Inf-Preservation

Next, important notation for morphism between posets and lattices is introduced: sup-preservation, inf-preservation and related properties.

abbreviation *Sup-pres* :: ('a::*Sup* ⇒ 'b::*Sup*) ⇒ *bool* **where**
Sup-pres $f \equiv f \circ Sup = Sup \circ (`)\ f$

abbreviation *Inf-pres* :: ('a::*Inf* ⇒ 'b::*Inf*) ⇒ *bool* **where**
Inf-pres $f \equiv f \circ Inf = Inf \circ (`)\ f$

abbreviation *sup-pres* :: ('a::*sup* ⇒ 'b::*sup*) ⇒ *bool* **where**
sup-pres $f \equiv (\forall\, x\ y.\ f\ (x \sqcup y) = f\ x \sqcup f\ y)$

abbreviation *inf-pres* :: ('a::*inf* ⇒ 'b::*inf*) ⇒ *bool* **where**
inf-pres $f \equiv (\forall\, x\ y.\ f\ (x \sqcap y) = f\ x \sqcap f\ y)$

abbreviation *bot-pres* :: ('a::*bot* ⇒ 'b::*bot*) ⇒ *bool* **where**
bot-pres $f \equiv f\ \bot = \bot$

abbreviation *top-pres* :: ('a::*top* ⇒ 'b::*top*) ⇒ *bool* **where**
top-pres $f \equiv f\ \top = \top$

abbreviation *Sup-dual* :: ('a::*Sup* ⇒ 'b::*Inf*) ⇒ *bool* **where**
Sup-dual $f \equiv f \circ Sup = Inf \circ (`)\ f$

abbreviation *Inf-dual* :: ('a::*Inf* ⇒ 'b::*Sup*) ⇒ *bool* **where**
Inf-dual $f \equiv f \circ Inf = Sup \circ (`)\ f$

abbreviation *sup-dual* :: ('a::*sup* ⇒ 'b::*inf*) ⇒ *bool* **where**
sup-dual $f \equiv (\forall\, x\ y.\ f\ (x \sqcup y) = f\ x \sqcap f\ y)$

abbreviation *inf-dual* :: ('a::*inf* ⇒ 'b::*sup*) ⇒ *bool* **where**
inf-dual $f \equiv (\forall\, x\ y.\ f\ (x \sqcap y) = f\ x \sqcup f\ y)$

abbreviation *bot-dual* :: ('a::*bot* ⇒ 'b::*top*) ⇒ *bool* **where**
bot-dual $f \equiv f\ \bot = \top$

abbreviation *top-dual* :: ('a::*top* ⇒ 'b::*bot*) ⇒ *bool* **where**
top-dual $f \equiv f\ \top = \bot$

Inf-preservation and sup-preservation relate with duality.

lemma *Inf-pres-map-dual-var*:
Inf-pres f = *Sup-pres* ($\partial_F\ f$)

for f :: $'a$::*complete-lattice-with-dual* ⇒ $'b$::*complete-lattice-with-dual*
⟨*proof*⟩

lemma *Inf-pres-map-dual*: *Inf-pres* = *Sup-pres* ∘ (∂_F::($'a$::*complete-lattice-with-dual* ⇒ $'b$::*complete-lattice-with-dual*) ⇒ $'a$ ⇒ $'b$)
⟨*proof*⟩

lemma *Sup-pres-map-dual-var*:
fixes f :: $'a$::*complete-lattice-with-dual* ⇒ $'b$::*complete-lattice-with-dual*
shows *Sup-pres* f = *Inf-pres* (∂_F f)
⟨*proof*⟩

lemma *Sup-pres-map-dual*: *Sup-pres* = *Inf-pres* ∘ (∂_F::($'a$::*complete-lattice-with-dual* ⇒ $'b$::*complete-lattice-with-dual*) ⇒ $'a$ ⇒ $'b$)
⟨*proof*⟩

The following lemmas relate isotonicity of functions between complete lattices with weak (left) preservation properties of sups and infs.

lemma *fun-isol*: *mono* f ⟹ *mono* ((∘) f)
⟨*proof*⟩

lemma *fun-isor*: *mono* f ⟹ *mono* (λx. x ∘ f)
⟨*proof*⟩

lemma *Sup-sup-pres*:
fixes f :: $'a$::*complete-lattice* ⇒ $'b$::*complete-lattice*
shows *Sup-pres* f ⟹ *sup-pres* f
⟨*proof*⟩

lemma *Inf-inf-pres*:
fixes f :: $'a$::*complete-lattice* ⇒ $'b$::*complete-lattice*
shows*Inf-pres* f ⟹ *inf-pres* f
⟨*proof*⟩

lemma *Sup-bot-pres*:
fixes f :: $'a$::*complete-lattice* ⇒ $'b$::*complete-lattice*
shows *Sup-pres* f ⟹ *bot-pres* f
⟨*proof*⟩

lemma *Inf-top-pres*:
fixes f :: $'a$::*complete-lattice* ⇒ $'b$::*complete-lattice*
shows *Inf-pres* f ⟹ *top-pres* f
⟨*proof*⟩

lemma *Sup-sup-dual*:
fixes f :: $'a$::*complete-lattice* ⇒ $'b$::*complete-lattice*
shows *Sup-dual* f ⟹ *sup-dual* f
⟨*proof*⟩

lemma *Inf-inf-dual*:
fixes f :: $'a$::*complete-lattice* $\Rightarrow$ $'b$::*complete-lattice*
shows *Inf-dual f* $\Longrightarrow$ *inf-dual f*
⟨*proof*⟩

lemma *Sup-bot-dual*:
fixes f :: $'a$::*complete-lattice* $\Rightarrow$ $'b$::*complete-lattice*
shows *Sup-dual f* $\Longrightarrow$ *bot-dual f*
⟨*proof*⟩

lemma *Inf-top-dual*:
fixes f :: $'a$::*complete-lattice* $\Rightarrow$ $'b$::*complete-lattice*
shows *Inf-dual f* $\Longrightarrow$ *top-dual f*
⟨*proof*⟩

However, Inf-preservation does not imply top-preservation and Sup-preservation does not imply bottom-preservation.

lemma
fixes f :: $'a$::*complete-lattice* $\Rightarrow$ $'b$::*complete-lattice*
shows *Sup-pres f* $\Longrightarrow$ *top-pres f*
⟨*proof*⟩

lemma
fixes f :: $'a$::*complete-lattice* $\Rightarrow$ $'b$::*complete-lattice*
shows *Inf-pres f* $\Longrightarrow$ *bot-pres f*
⟨*proof*⟩

context *complete-lattice*
begin

lemma *iso-Inf-subdistl*:
fixes f :: $'a \Rightarrow$ $'b$::*complete-lattice*
shows *mono f* $\Longrightarrow$*f* $\circ$ *Inf* $\leq$ *Inf* $\circ$ (‘) *f*
⟨*proof*⟩

lemma *iso-Sup-supdistl*:
fixes f :: $'a \Rightarrow$ $'b$::*complete-lattice*
shows *mono f* $\Longrightarrow$ *Sup* $\circ$ (‘) *f* $\leq$ *f* $\circ$ *Sup*
⟨*proof*⟩

lemma *Inf-subdistl-iso*:
fixes f :: $'a \Rightarrow$ $'b$::*complete-lattice*
shows *f* $\circ$ *Inf* $\leq$ *Inf* $\circ$ (‘) *f* $\Longrightarrow$ *mono f*
⟨*proof*⟩

lemma *Sup-supdistl-iso*:
fixes f :: $'a \Rightarrow$ $'b$::*complete-lattice*
shows *Sup* $\circ$ (‘) *f* $\leq$ *f* $\circ$ *Sup* $\Longrightarrow$ *mono f*
⟨*proof*⟩

lemma *supdistl-iso*:
fixes $f :: 'a \Rightarrow 'b$::*complete-lattice*
shows $(Sup \circ (`)\ f \leq f \circ Sup) = mono\ f$
⟨*proof*⟩

lemma *subdistl-iso*:
fixes $f :: 'a \Rightarrow 'b$::*complete-lattice*
shows $(f \circ Inf \leq Inf \circ (`)\ f) = mono\ f$
⟨*proof*⟩

end

lemma *ord-iso-Inf-pres*:
fixes $f :: 'a$::*complete-lattice* $\Rightarrow 'b$::*complete-lattice*
shows *ord-iso* $f \Longrightarrow Inf \circ (`)\ f = f \circ Inf$
⟨*proof*⟩

lemma *ord-iso-Sup-pres*:
fixes $f :: 'a$::*complete-lattice* $\Rightarrow 'b$::*complete-lattice*
shows *ord-iso* $f \Longrightarrow Sup \circ (`)\ f = f \circ Sup$
⟨*proof*⟩

Right preservation of sups and infs is trivial.

lemma *fSup-distr*: *Sup-pres* $(\lambda x.\ x \circ f)$
⟨*proof*⟩

lemma *fSup-distr-var*: $\bigsqcup F \circ g = (\bigsqcup f \in F.\ f \circ g)$
⟨*proof*⟩

lemma *fInf-distr*: *Inf-pres* $(\lambda x.\ x \circ f)$
⟨*proof*⟩

lemma *fInf-distr-var*: $\bigsqcap F \circ g = (\bigsqcap f \in F.\ f \circ g)$
⟨*proof*⟩

The next set of lemma revisits the preservation properties in the function space.

lemma *fSup-subdistl*:
assumes *mono* $(f::'a$::*complete-lattice* $\Rightarrow 'b$::*complete-lattice*$)$
shows $Sup \circ (`)\ ((\circ)\ f) \leq (\circ)\ f \circ Sup$
⟨*proof*⟩

lemma *fSup-subdistl-var*:
fixes $f :: 'a$::*complete-lattice* $\Rightarrow 'b$::*complete-lattice*
shows $mono\ f \Longrightarrow (\bigsqcup g \in G.\ f \circ g) \leq f \circ \bigsqcup G$
⟨*proof*⟩

lemma *fInf-subdistl*:

fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'b{::}complete\text{-}lattice$
shows $mono\ f \Longrightarrow (\circ)\ f \circ Inf \leq Inf \circ (`)\ ((\circ)\ f)$
⟨*proof*⟩

lemma *fInf-subdistl-var*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'b{::}complete\text{-}lattice$
shows $mono\ f \Longrightarrow f \circ \bigsqcap G \leq (\bigsqcap g \in G.\ f \circ g)$
⟨*proof*⟩

lemma *fSup-distl*: $Sup\text{-}pres\ f \Longrightarrow Sup\text{-}pres\ ((\circ)\ f)$
⟨*proof*⟩

lemma *fSup-distl-var*: $Sup\text{-}pres\ f \Longrightarrow f \circ \bigsqcup G = (\bigsqcup g \in G.\ f \circ g)$
⟨*proof*⟩

lemma *fInf-distl*: $Inf\text{-}pres\ f \Longrightarrow Inf\text{-}pres\ ((\circ)\ f)$
⟨*proof*⟩

lemma *fInf-distl-var*: $Inf\text{-}pres\ f \Longrightarrow f \circ \bigsqcap G = (\bigsqcap g \in G.\ f \circ g)$
⟨*proof*⟩

Downsets preserve infs whereas upsets preserve sups.

lemma *Inf-pres-downset*: $Inf\text{-}pres\ (\downarrow{::}'a{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'a\ set)$
⟨*proof*⟩

lemma *Sup-dual-upset*: $Sup\text{-}dual\ (\uparrow{::}'a{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'a\ set)$
⟨*proof*⟩

Images of Sup-morphisms are closed under Sups and images of Inf-morphisms are closed under Infs.

lemma *Sup-pres-Sup-closed*: $Sup\text{-}pres\ f \Longrightarrow Sup\text{-}closed\text{-}set\ (range\ f)$
⟨*proof*⟩

lemma *Inf-pres-Inf-closed*: $Inf\text{-}pres\ f \Longrightarrow Inf\text{-}closed\text{-}set\ (range\ f)$
⟨*proof*⟩

It is well known that functions into complete lattices form complete lattices. Here, such results are shown for the subclasses of isotone functions, where additional closure conditions must be respected.

typedef (**overloaded**) $'a\ iso = \{f{::}'a{::}order \Rightarrow 'a{::}order.\ mono\ f\}$
⟨*proof*⟩

setup-lifting *type-definition-iso*

instantiation *iso* :: (*complete-lattice*) *complete-lattice*
begin

lift-definition $Inf\text{-}iso :: 'a{::}complete\text{-}lattice\ iso\ set \Rightarrow 'a\ iso$ **is** *Sup*

⟨*proof*⟩

lift-definition *Sup-iso* :: *'a*::*complete-lattice iso set* ⇒ *'a iso* **is** *Inf*
⟨*proof*⟩

lift-definition *bot-iso* :: *'a*::*complete-lattice iso* **is** ⊤
⟨*proof*⟩

lift-definition *sup-iso* :: *'a*::*complete-lattice iso* ⇒ *'a iso* ⇒ *'a iso* **is** *inf*
⟨*proof*⟩

lift-definition *top-iso* :: *'a*::*complete-lattice iso* **is** ⊥
⟨*proof*⟩

lift-definition *inf-iso* :: *'a*::*complete-lattice iso* ⇒ *'a iso* ⇒ *'a iso* **is** *sup*
⟨*proof*⟩

lift-definition *less-eq-iso* :: *'a*::*complete-lattice iso* ⇒ *'a iso* ⇒ *bool* **is** (≥)⟨*proof*⟩

lift-definition *less-iso* :: *'a*::*complete-lattice iso* ⇒ *'a iso* ⇒ *bool* **is** (>)⟨*proof*⟩

instance
⟨*proof*⟩

end

Duality has been baked into this result because of its relevance for predicate transformers. A proof where Sups are mapped to Sups and Infs to Infs is certainly possible, but two instantiation of the same type and the same classes are unfortunately impossible. Interpretations could be used instead.

A corresponding result for Inf-preseving functions and Sup-lattices, is proved in components on transformers, as more advanced properties about Inf-preserving functions are needed.

4.6 Alternative Definitions for Complete Boolean Algebras

The current definitions of complete boolean algebras deviates from that in most textbooks in that a distributive law with infinite sups and infinite infs is used. There are interesting applications, for instance in topology, where weaker laws are needed — for instance for frames and locales.

class *complete-heyting-algebra* = *complete-lattice* +
assumes *ch-dist*: $x \sqcap \bigsqcup Y = (\bigsqcup y \in Y.\ x \sqcap y)$

Complete Heyting algebras are also known as frames or locales (they differ with respect to their morphisms).

class *complete-co-heyting-algebra* = *complete-lattice* +
assumes *co-ch-dist*: $x \sqcup \bigsqcap Y = (\bigsqcap y \in Y.\ x \sqcup y)$

class *complete-boolean-algebra-alt* = *complete-lattice* + *boolean-algebra*

instance *set* :: (*type*) *complete-boolean-algebra-alt*⟨*proof*⟩

context *complete-boolean-algebra-alt*
begin

subclass *complete-heyting-algebra*
⟨*proof*⟩

subclass *complete-co-heyting-algebra*
⟨*proof*⟩

lemma *de-morgan1*: $-(\bigsqcup X) = (\bigsqcap x \in X.\ -x)$
⟨*proof*⟩

lemma *de-morgan2*: $-(\bigsqcap X) = (\bigsqcup x \in X.\ -x)$
⟨*proof*⟩

end

class *complete-boolean-algebra-alt-with-dual* = *complete-lattice-with-dual* + *complete-boolean-algebra-alt*

instantiation *set* :: (*type*) *complete-boolean-algebra-alt-with-dual*
begin

definition *dual-set* :: *'a set* ⇒ *'a set* **where**
dual-set = *uminus*

instance
⟨*proof*⟩

end

context *complete-boolean-algebra-alt*
begin

sublocale *cba-dual*: *complete-boolean-algebra-alt-with-dual* - - - - - - - - *uminus* - -
⟨*proof*⟩

end

4.7 Atomic Boolean Algebras

Next, atomic boolean algebras are defined.

context *bounded-lattice*
begin

Atoms are covers of bottom.

definition *atom* $x = (x \neq \bot \wedge \neg(\exists\, y.\ \bot < y \wedge y < x))$

definition *atom-map* $x = \{y.\ atom\ y \wedge y \leq x\}$

lemma *atom-map-def-var*: *atom-map* $x = {\downarrow}x \cap Collect\ atom$
⟨*proof*⟩

lemma *atom-map-atoms*: $\bigcup(range\ atom\text{-}map) = Collect\ atom$
⟨*proof*⟩

end

typedef (**overloaded**) $'a\ atoms = range\ (atom\text{-}map::'a::bounded\text{-}lattice \Rightarrow 'a\ set)$
⟨*proof*⟩

setup-lifting *type-definition-atoms*

definition *at-map* :: $'a::bounded\text{-}lattice \Rightarrow 'a\ atoms$ **where**
at-map = *Abs-atoms* ∘ *atom-map*

class *atomic-boolean-algebra* = *boolean-algebra* +
assumes *atomicity*: $x \neq \bot \Longrightarrow (\exists\, y.\ atom\ y \wedge y \leq x)$

class *complete-atomic-boolean-algebra* = *complete-lattice* + *atomic-boolean-algebra*

begin

subclass *complete-boolean-algebra-alt*⟨*proof*⟩

end

Here are two equivalent definitions for atoms; first in boolean algebras, and then in complete boolean algebras.

context *boolean-algebra*
begin

The following two conditions are taken from Koppelberg's book [6].

lemma *atom-neg*: $atom\ x \Longrightarrow x \neq \bot \wedge (\forall\, y\ z.\ x \leq y \vee x \leq -y)$
⟨*proof*⟩

lemma *atom-sup*: $(\forall\, y.\ x \leq y \vee x \leq -y) \Longrightarrow (\forall\, y\ z.\ (x \leq y \vee x \leq z) = (x \leq y \sqcup z))$
⟨*proof*⟩

lemma *sup-atom*: $x \neq \bot \Longrightarrow (\forall\, y\ z.\ (x \leq y \vee x \leq z) = (x \leq y \sqcup z)) \Longrightarrow atom\ x$
⟨*proof*⟩

lemma *atom-sup-iff*: $atom\ x = (x \neq \bot \wedge (\forall\, y\ z.\ (x \leq y \vee x \leq z) = (x \leq y \sqcup z)))$
⟨*proof*⟩

lemma *atom-neg-iff*: $atom\ x = (x \neq \bot \wedge (\forall\, y\ z.\ x \leq y \vee x \leq -y))$
⟨*proof*⟩

lemma *atom-map-bot-pres*: $atom\text{-}map\ \bot = \{\}$
⟨*proof*⟩

lemma *atom-map-top-pres*: $atom\text{-}map\ \top = Collect\ atom$
⟨*proof*⟩

end

context *complete-boolean-algebra-alt*
begin

lemma *atom-Sup*: $\bigwedge Y.\ x \neq \bot \Longrightarrow (\forall\, y.\ x \leq y \vee x \leq -y) \Longrightarrow ((\exists\, y \in Y.\ x \leq y) = (x \leq \bigsqcup Y))$
⟨*proof*⟩

lemma *Sup-atom*: $x \neq \bot \Longrightarrow (\forall\, Y.\ (\exists\, y \in Y.\ x \leq y) = (x \leq \bigsqcup Y)) \Longrightarrow atom\ x$
⟨*proof*⟩

lemma *atom-Sup-iff*: $atom\ x = (x \neq \bot \wedge (\forall\, Y.\ (\exists\, y \in Y.\ x \leq y) = (x \leq \bigsqcup Y)))$
⟨*proof*⟩

end

end

5 Representation Theorems for Orderings and Lattices

theory *Representations*
imports *Order-Lattice-Props*

begin

5.1 Representation of Posets

The isomorphism between partial orders and downsets with set inclusion is well known. It forms the basis of Priestley and Stone duality. I show it not only for objects, but also order morphisms, hence establish equivalences and isomorphisms between categories.

typedef (**overloaded**) $'a\ downset = range\ (\downarrow :: 'a :: ord \Rightarrow 'a\ set)$
⟨*proof*⟩

setup-lifting *type-definition-downset*

The map ds yields the isomorphism between the set and the powerset level if its range is restricted to downsets.

definition *ds* :: *'a*::*ord* ⇒ *'a downset* **where**
ds = *Abs-downset* ∘ ↓

In a complete lattice, its inverse is Sup.

definition *SSup* :: *'a*::*complete-lattice downset* ⇒ *'a* **where**
SSup = *Sup* ∘ *Rep-downset*

lemma *ds-SSup-inv*: *ds* ∘ *SSup* = (*id*::*'a*::*complete-lattice downset* ⇒ *'a downset*)
⟨*proof*⟩

lemma *SSup-ds-inv*: *SSup* ∘ *ds* = (*id*::*'a*::*complete-lattice* ⇒ *'a*)
⟨*proof*⟩

instantiation *downset* :: (*ord*) *order*
begin

lift-definition *less-eq-downset* :: *'a downset* ⇒ *'a downset* ⇒ *bool* **is** (λ*X Y*. *Rep-downset X* ⊆ *Rep-downset Y*) ⟨*proof*⟩

lift-definition *less-downset* :: *'a downset* ⇒ *'a downset* ⇒ *bool* **is** (λ*X Y*. *Rep-downset X* ⊂ *Rep-downset Y*) ⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma *ds-iso*: *mono ds*
⟨*proof*⟩

lemma *ds-faithful*: *ds x* ≤ *ds y* ⟹ *x* ≤ (*y*::*'a*::*order*)
⟨*proof*⟩

lemma *ds-inj*: *inj* (*ds*::*'a*::*order* ⇒ *'a downset*)
⟨*proof*⟩

lemma *ds-surj*: *surj ds*
⟨*proof*⟩

lemma *ds-bij*: *bij* (*ds*::*'a*::*order* ⇒ *'a downset*)
⟨*proof*⟩

lemma *ds-ord-iso*: *ord-iso ds*
⟨*proof*⟩

The morphishms between orderings and downsets are isotone functions. One can define functors mapping back and forth between these.

definition *map-ds* :: ('*a*::*complete-lattice* ⇒ '*b*::*complete-lattice*) ⇒ ('*a downset* ⇒ '*b downset*) **where**
map-ds f = *ds* ∘ *f* ∘ *SSup*

This definition is actually contrived. We have shown that a function f between posets P and Q is isotone if and only if the inverse image of f maps downclosed sets in Q to downclosed sets in P. There is the following duality: ds is a natural transformation between the identity functor and the preimage functor as a contravariant functor from P to Q. Hence orderings with isotone maps and downsets with downset-preserving maps are dual, which is a first step towards Stone duality. I don't see a way of proving this with Isabelle, as the types of the preimage of f are the wrong way and I don't see how I could capture opposition with what I have.

lemma *map-ds-prop*:
fixes *f* :: '*a*::*complete-lattice* ⇒ '*b*::*complete-lattice*
shows *map-ds f* ∘ *ds* = *ds* ∘ *f*
⟨*proof*⟩

lemma *map-ds-prop2*:
fixes *f* :: '*a*::*complete-lattice* ⇒ '*b*::*complete-lattice*
shows *map-ds f* ∘ *ds* = *ds* ∘ *id f*
⟨*proof*⟩

This is part of showing that map-ds is naturally isomorphic to the identity functor, ds being the natural isomorphism.

definition *map-SSup* :: ('*a downset* ⇒ '*b downset*) ⇒ ('*a*::*complete-lattice* ⇒ '*b*::*complete-lattice*) **where**
map-SSup F = *SSup* ∘ *F* ∘ *ds*

lemma *map-ds-iso-pres*:
fixes *f* :: '*a*::*complete-lattice* ⇒ '*b*::*complete-lattice*
shows *mono f* ⟹ *mono* (*map-ds f*)
⟨*proof*⟩

lemma *map-SSup-iso-pres*:
fixes *F* :: '*a*::*complete-lattice downset* ⇒ '*b*::*complete-lattice downset*
shows *mono F* ⟹ *mono* (*map-SSup F*)
⟨*proof*⟩

lemma *map-SSup-prop*:
fixes *F* :: '*a*::*complete-lattice downset* ⇒ '*b*::*complete-lattice downset*
shows *ds* ∘ *map-SSup F* = *F* ∘ *ds*
⟨*proof*⟩

lemma *map-SSup-prop2*:

fixes $F :: 'a::complete\text{-}lattice\ downset \Rightarrow 'b::complete\text{-}lattice\ downset$
shows $ds \circ map\text{-}SSup\ F = id\ F \circ ds$
⟨*proof*⟩

lemma *map-ds-func1*: $map\text{-}ds\ id = (id::'a::complete\text{-}lattice\ downset \Rightarrow 'a\ downset)$
⟨*proof*⟩

lemma *map-ds-func2*:
fixes $g :: 'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice$
shows $map\text{-}ds\ (f \circ g) = map\text{-}ds\ f \circ map\text{-}ds\ g$
⟨*proof*⟩

lemma *map-SSup-func1*: $map\text{-}SSup\ (id::'a::complete\text{-}lattice\ downset \Rightarrow 'a\ downset) = id$
⟨*proof*⟩

lemma *map-SSup-func2*:
fixes $F :: 'c::complete\text{-}lattice\ downset \Rightarrow 'b::complete\text{-}lattice\ downset$
and $G :: 'a::complete\text{-}lattice\ downset \Rightarrow 'c\ downset$
shows $map\text{-}SSup\ (F \circ G) = map\text{-}SSup\ F \circ map\text{-}SSup\ G$
⟨*proof*⟩

lemma *map-SSup-map-ds-inv*: $map\text{-}SSup \circ map\text{-}ds = (id::('a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice) \Rightarrow ('a \Rightarrow 'b))$
⟨*proof*⟩

lemma *map-ds-map-SSup-inv*: $map\text{-}ds \circ map\text{-}SSup = (id::('a::complete\text{-}lattice\ downset \Rightarrow 'b::complete\text{-}lattice\ downset) \Rightarrow ('a\ downset \Rightarrow 'b\ downset))$
⟨*proof*⟩

lemma *inj-map-ds*: $inj\ (map\text{-}ds::('a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice) \Rightarrow ('a\ downset \Rightarrow 'b\ downset))$
⟨*proof*⟩

lemma *inj-map-SSup*: $inj\ (map\text{-}SSup::('a::complete\text{-}lattice\ downset \Rightarrow 'b::complete\text{-}lattice\ downset) \Rightarrow ('a \Rightarrow 'b))$
⟨*proof*⟩

lemma *map-ds-map-SSup-iff*:
fixes $g :: 'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice$
shows $(f = map\text{-}ds\ g) = (map\text{-}SSup\ f = g)$
⟨*proof*⟩

This gives an isomorphism between categories.

lemma *surj-map-ds*: $surj\ (map\text{-}ds::('a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice) \Rightarrow ('a\ downset \Rightarrow 'b\ downset))$
⟨*proof*⟩

lemma *surj-map-SSup*: $surj\ (map\text{-}SSup::('a::complete\text{-}lattice\text{-}with\text{-}dual\ downset \Rightarrow$

$'b$::*complete-lattice-with-dual downset*) ⇒ ($'a$ ⇒ $'b$))
⟨*proof*⟩

There is of course a dual result for upsets with the reverse inclusion ordering. Once again, it seems impossible to capture the "real" duality that uses the inverse image functor.

typedef (**overloaded**) $'a$ *upset* = *range* (↑::$'a$::*ord* ⇒ $'a$ *set*)
⟨*proof*⟩

setup-lifting *type-definition-upset*

definition *us* :: $'a$::*ord* ⇒ $'a$ *upset* **where**
us = *Abs-upset* ∘ ↑

definition *IInf* :: $'a$::*complete-lattice upset* ⇒ $'a$ **where**
IInf = *Inf* ∘ *Rep-upset*

lemma *us-ds*: *us* = *Abs-upset* ∘ (‘) ∂ ∘ *Rep-downset* ∘ *ds* ∘ (∂::$'a$::*ord-with-dual* ⇒ $'a$)
⟨*proof*⟩

lemma *IInf-SSup*: *IInf* = ∂ ∘ *SSup* ∘ *Abs-downset* ∘ (‘) (∂::$'a$::*complete-lattice-with-dual* ⇒ $'a$) ∘ *Rep-upset*
⟨*proof*⟩

lemma *us-IInf-inv*: *us* ∘ *IInf* = (*id*::$'a$::*complete-lattice-with-dual upset* ⇒ $'a$ *upset*)
⟨*proof*⟩

lemma *IInf-us-inv*: *IInf* ∘ *us* = (*id*::$'a$::*complete-lattice-with-dual* ⇒ $'a$)
⟨*proof*⟩

instantiation *upset* :: (*ord*) *order*
begin

lift-definition *less-eq-upset* :: $'a$ *upset* ⇒ $'a$ *upset* ⇒ *bool* **is** (λ*X Y*. *Rep-upset X* ⊇ *Rep-upset Y*) ⟨*proof*⟩

lift-definition *less-upset* :: $'a$ *upset* ⇒ $'a$ *upset* ⇒ *bool* **is** (λ*X Y*. *Rep-upset X* ⊃ *Rep-upset Y*) ⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma *us-iso*: *x* ≤ *y* ⟹ *us x* ≤ *us* (*y*::$'a$::*order-with-dual*)
⟨*proof*⟩

lemma *us-faithful*: *us x* ≤ *us y* ⟹ *x* ≤ (*y*::'*a*::*order-with-dual*)
⟨*proof*⟩

lemma *us-inj*: *inj* (*us*::'*a*::*order-with-dual* ⇒ '*a upset*)
⟨*proof*⟩

lemma *us-surj*: *surj* (*us*::'*a*::*order-with-dual* ⇒ '*a upset*)
⟨*proof*⟩

lemma *us-bij*: *bij* (*us*::'*a*::*order-with-dual* ⇒ '*a upset*)
⟨*proof*⟩

lemma *us-ord-iso*: *ord-iso* (*us*::'*a*::*order-with-dual* ⇒ '*a upset*)
⟨*proof*⟩

definition *map-us* :: ('*a*::*complete-lattice* ⇒ '*b*::*complete-lattice*) ⇒ ('*a upset* ⇒ '*b upset*) **where**
map-us f = *us* ∘ *f* ∘ *IInf*

lemma *map-us-prop*: *map-us f* ∘ (*us*::'*a*::*complete-lattice-with-dual* ⇒ '*a upset*) = *us* ∘ *id f*
⟨*proof*⟩

definition *map-IInf* :: ('*a upset* ⇒ '*b upset*) ⇒ ('*a*::*complete-lattice* ⇒ '*b*::*complete-lattice*) **where**
map-IInf F = *IInf* ∘ *F* ∘ *us*

lemma *map-IInf-prop*: (*us*::'*a*::*complete-lattice-with-dual* ⇒ '*a upset*) ∘ *map-IInf F* = *id F* ∘ *us*
⟨*proof*⟩

lemma *map-us-func1*: *map-us id* = (*id*::'*a*::*complete-lattice-with-dual upset* ⇒ '*a upset*)
⟨*proof*⟩

lemma *map-us-func2*:
fixes *f* :: '*c*::*complete-lattice-with-dual* ⇒ '*b*::*complete-lattice-with-dual*
and *g* :: '*a*::*complete-lattice-with-dual* ⇒ '*c*
shows *map-us* (*f* ∘ *g*) = *map-us f* ∘ *map-us g*
⟨*proof*⟩

lemma *map-IInf-func1*: *map-IInf id* = (*id*::'*a*::*complete-lattice-with-dual* ⇒ '*a*)
⟨*proof*⟩

lemma *map-IInf-func2*:
fixes *F* :: '*c*::*complete-lattice-with-dual upset* ⇒ '*b*::*complete-lattice-with-dual upset*
and *G* :: '*a*::*complete-lattice-with-dual upset* ⇒ '*c upset*
shows *map-IInf* (*F* ∘ *G*) = *map-IInf F* ∘ *map-IInf G*

⟨*proof*⟩

lemma *map-IInf-map-us-inv*: *map-IInf* ∘ *map-us* = (*id*::('*a*::*complete-lattice-with-dual* ⇒ '*b*::*complete-lattice-with-dual*) ⇒ ('*a* ⇒ '*b*))
⟨*proof*⟩

lemma *map-us-map-IInf-inv*: *map-us* ∘ *map-IInf* = (*id*::('*a*::*complete-lattice-with-dual upset* ⇒ '*b*::*complete-lattice-with-dual upset*) ⇒ ('*a upset* ⇒ '*b upset*))
⟨*proof*⟩

lemma *inj-map-us*: *inj* (*map-us*::('*a*::*complete-lattice-with-dual* ⇒ '*b*::*complete-lattice-with-dual*) ⇒ ('*a upset* ⇒ '*b upset*))
⟨*proof*⟩

lemma *inj-map-IInf*: *inj* (*map-IInf*::('*a*::*complete-lattice-with-dual upset* ⇒ '*b*::*complete-lattice-with-dual upset*) ⇒ ('*a* ⇒ '*b*))
⟨*proof*⟩

lemma *map-us-map-IInf-iff*:
fixes *g* :: '*a*::*complete-lattice-with-dual* ⇒ '*b*::*complete-lattice-with-dual*
shows (*f* = *map-us g*) = (*map-IInf f* = *g*)
⟨*proof*⟩

lemma *map-us-mono-pres*:
fixes *f* :: '*a*::*complete-lattice-with-dual* ⇒ '*b*::*complete-lattice-with-dual*
shows *mono f* ⟹ *mono* (*map-us f*)
⟨*proof*⟩

lemma *map-IInf-mono-pres*:
fixes *F* :: '*a*::*complete-lattice-with-dual upset* ⇒ '*b*::*complete-lattice-with-dual upset*
shows *mono F* ⟹ *mono* (*map-IInf F*)
⟨*proof*⟩

lemma *surj-map-us*: *surj* (*map-us*::('*a*::*complete-lattice-with-dual* ⇒ '*b*::*complete-lattice-with-dual*) ⇒ ('*a upset* ⇒ '*b upset*))
⟨*proof*⟩

lemma *surj-map-IInf*: *surj* (*map-IInf*::('*a*::*complete-lattice-with-dual upset* ⇒ '*b*::*complete-lattice-with-dual upset*) ⇒ ('*a* ⇒ '*b*))
⟨*proof*⟩

Hence we have again an isomorphism — or rather equivalence — between categories. Here, however, duality is not consistently picked up.

5.2 Stone's Theorem in the Presence of Atoms

Atom-map is a boolean algebra morphism.

context *boolean-algebra*

begin

lemma *atom-map-compl-pres*: *atom-map* $(-x)$ = *Collect atom* − *atom-map x* ⟨*proof*⟩

lemma *atom-map-sup-pres*: *atom-map* $(x \sqcup y)$ = *atom-map x* ∪ *atom-map y* ⟨*proof*⟩

lemma *atom-map-inf-pres*: *atom-map* $(x \sqcap y)$ = *atom-map x* ∩ *atom-map y* ⟨*proof*⟩

lemma *atom-map-minus-pres*: *atom-map* $(x - y)$ = *atom-map x* − *atom-map y* ⟨*proof*⟩

end

The homomorphic images of boolean algebras under atom-map are boolean algebras — in fact powerset boolean algebras.

instantiation *atoms* :: (*boolean-algebra*) *boolean-algebra*
begin

lift-definition *minus-atoms* :: *'a atoms* ⇒ *'a atoms* ⇒ *'a atoms* **is** λ*x y*. *Abs-atoms* (*Rep-atoms x* − *Rep-atoms y*)⟨*proof*⟩

lift-definition *uminus-atoms* :: *'a atoms* ⇒ *'a atoms* **is** λ*x*. *Abs-atoms* (*Collect atom* − *Rep-atoms x*)⟨*proof*⟩

lift-definition *bot-atoms* :: *'a atoms* **is** *Abs-atoms* {}⟨*proof*⟩

lift-definition *sup-atoms* :: *'a atoms* ⇒ *'a atoms* ⇒ *'a atoms* **is** λ*x y*. *Abs-atoms* (*Rep-atoms x* ∪ *Rep-atoms y*)⟨*proof*⟩

lift-definition *top-atoms* :: *'a atoms* **is** *Abs-atoms* (*Collect atom*)⟨*proof*⟩

lift-definition *inf-atoms* :: *'a atoms* ⇒ *'a atoms* ⇒ *'a atoms* **is** λ*x y*. *Abs-atoms* (*Rep-atoms x* ∩ *Rep-atoms y*)⟨*proof*⟩

lift-definition *less-eq-atoms* :: *'a atoms* ⇒ *'a atoms* ⇒ *bool* **is** (λ*x y*. *Rep-atoms x* ⊆ *Rep-atoms y*)⟨*proof*⟩

lift-definition *less-atoms* :: *'a atoms* ⇒ *'a atoms* ⇒ *bool* **is** (λ*x y*. *Rep-atoms x* ⊂ *Rep-atoms y*)⟨*proof*⟩

instance
⟨*proof*⟩

end

The homomorphism atom-map can then be restricted in its output type to

the powerset boolean algebra.

lemma *at-map-bot-pres*: *at-map* $\bot = \bot$
⟨*proof*⟩

lemma *at-map-top-pres*: *at-map* $\top = \top$
⟨*proof*⟩

lemma *at-map-compl-pres*: *at-map* $\circ$ *uminus* = *uminus* $\circ$ *at-map*
⟨*proof*⟩

lemma *at-map-sup-pres*: *at-map* $(x \sqcup y)$ = *at-map* $x \sqcup$ *at-map* y
⟨*proof*⟩

lemma *at-map-inf-pres*: *at-map* $(x \sqcap y)$ = *at-map* $x \sqcap$ *at-map* y
⟨*proof*⟩

lemma *at-map-minus-pres*: *at-map* $(x - y)$ = *at-map* $x -$ *at-map* y
⟨*proof*⟩

context *atomic-boolean-algebra*
begin

In atomic boolean algebras, atom-map is an embedding that maps atoms of the boolean algebra to those of the powerset boolean algebra. Analogous properties hold for at-map.

lemma *inj-atom-map*: *inj atom-map*
⟨*proof*⟩

lemma *atom-map-atom-pres*: *atom* $x \Longrightarrow$ *atom-map* $x = \{x\}$
⟨*proof*⟩

lemma *atom-map-atom-pres2*: *atom* $x \Longrightarrow$ *atom* (*atom-map* x)
⟨*proof*⟩

end

lemma *inj-at-map*: *inj* (*at-map*::'*a*::*atomic-boolean-algebra* $\Rightarrow$ '*a atoms*)
⟨*proof*⟩

lemma *at-map-atom-pres*: *atom* (x::'*a*::*atomic-boolean-algebra*) $\Longrightarrow$ *at-map* x = *Abs-atoms* $\{x\}$
⟨*proof*⟩

lemma *at-map-atom-pres2*: *atom* (x::'*a*::*atomic-boolean-algebra*) $\Longrightarrow$ *atom* (*at-map* x)
⟨*proof*⟩

Homomorphic images of atomic boolean algebras under atom-map are there-

fore atomic (rather obviously).

instance *atoms* :: (*atomic-boolean-algebra*) *atomic-boolean-algebra*
⟨*proof*⟩

context *complete-boolean-algebra-alt*
begin

In complete boolean algebras, atom-map is surjective; more precisely it is the left inverse of Sup, at least for sets of atoms. Below, this statement is made more explicit for at-map.

lemma *surj-atom-map*: $Y \subseteq$ *Collect atom* $\Longrightarrow$ $Y =$ *atom-map* $(\bigsqcup Y)$
⟨*proof*⟩

In this setting, atom-map is a complete boolean algebra morphism.

lemma *atom-map-Sup-pres*: *atom-map* $(\bigsqcup X) = (\bigcup x \in X.$ *atom-map* $x)$
⟨*proof*⟩

lemma *atom-map-Sup-pres-var*: *atom-map* $\circ$ *Sup* $=$ *Sup* $\circ$ (‘) *atom-map*
⟨*proof*⟩

For Inf-preservation, it is important that Infs are restricted to homomorphic images; hence they need to be pushed into the set of all atoms.

lemma *atom-map-Inf-pres*: *atom-map* $(\bigsqcap X) =$ *Collect atom* $\cap$ $(\bigcap x \in X.$ *atom-map* $x)$
⟨*proof*⟩

end

It follows that homomorphic images of complete boolean algebras under atom-map form complete boolean algebras.

instantiation *atoms* :: (*complete-boolean-algebra-alt*) *complete-boolean-algebra-alt*
begin

lift-definition *Inf-atoms* :: *'a*::*complete-boolean-algebra-alt atoms set* ⇒ *'a*::*complete-boolean-algebra-alt atoms* **is** $\lambda X.$ *Abs-atoms* (*Collect atom* ∩ *Inter* ((‘) *Rep-atoms X*))⟨*proof*⟩

lift-definition *Sup-atoms* :: *'a*::*complete-boolean-algebra-alt atoms set* ⇒ *'a*::*complete-boolean-algebra-alt atoms* **is** $\lambda X.$ *Abs-atoms* (*Union* ((‘) *Rep-atoms X*))⟨*proof*⟩

instance
⟨*proof*⟩

end

Once more, properties proved above can now be restricted to at-map.

lemma *surj-at-map-var*: *at-map* $\circ$ *Sup* $\circ$ *Rep-atoms* $=$ (*id*::*'a*::*complete-boolean-algebra-alt atoms* ⇒ *'a atoms*)

⟨*proof*⟩

lemma *surj-at-map*: *surj* (*at-map*::′*a*::*complete-boolean-algebra-alt* ⇒ ′*a atoms*)
⟨*proof*⟩

lemma *at-map-Sup-pres*: *at-map* ∘ *Sup* = *Sup* ∘ (‘) (*at-map*::′*a*::*complete-boolean-algebra-alt* ⇒ ′*a atoms*)
⟨*proof*⟩

lemma *at-map-Sup-pres-var*: *at-map* (⨆ *X*) = (⨆ (*x*::′*a*::*complete-boolean-algebra-alt*) ∈ *X*. (*at-map x*))
⟨*proof*⟩

lemma *at-map-Inf-pres*: *at-map* (⨅ *X*) = *Abs-atoms* (*Collect atom* ⊓ (⨅ *x* ∈ *X*. (*Rep-atoms* (*at-map* (*x*::′*a*::*complete-boolean-algebra-alt*)))))
⟨*proof*⟩

lemma *at-map-Inf-pres-var*: *at-map* ∘ *Inf* = *Inf* ∘ (‘) (*at-map*::′*a*::*complete-boolean-algebra-alt* ⇒ ′*a atoms*)
⟨*proof*⟩

Finally, on complete atomic boolean algebras (CABAs), at-map is an isomorphism, that is, a bijection that preserves the complete boolean algebra operations. Thus every CABA is isomorphic to a powerset boolean algebra and every powerset boolean algebra is a CABA. The bijective pair is given by at-map and Sup (defined on the powerset algebra). This theorem is a little version of Stone's theorem. In the general case, ultrafilters play the role of atoms.

lemma *Sup* ∘ *atom-map* = (*id*::′*a*::*complete-atomic-boolean-algebra* ⇒ ′*a*)
⟨*proof*⟩

lemma *inj-at-map-var*: *Sup* ∘ *Rep-atoms* ∘ *at-map* = (*id* ::′*a*::*complete-atomic-boolean-algebra* ⇒ ′*a*)

⟨*proof*⟩

lemma *bij-at-map*: *bij* (*at-map*::′*a*::*complete-atomic-boolean-algebra* ⇒ ′*a atoms*)
⟨*proof*⟩

instance *atoms* :: (*complete-atomic-boolean-algebra*) *complete-atomic-boolean-algebra*⟨*proof*⟩

A full consideration of Stone duality is left for future work.

end

6 Galois Connections

theory *Galois-Connections*
imports *Order-Lattice-Props*

begin

6.1 Definitions and Basic Properties

The approach follows the Compendium of Continuous Lattices [3], without attempting completeness. First, left and right adjoints of a Galois connection are defined.

definition *adj* :: ('*a*::*ord* ⇒ '*b*::*ord*) ⇒ ('*b* ⇒ '*a*) ⇒ *bool* (**infixl** ‹⊣› *70*) **where**
(*f* ⊣ *g*) = (∀ *x y*. (*f x* ≤ *y*) = (*x* ≤ *g y*))

definition *ladj* (*g*::'*a*::*Inf* ⇒ '*b*::*ord*) = (λ*x*. ⨅{*y*. *x* ≤ *g y*})

definition *radj* (*f*::'*a*::*Sup* ⇒ '*b*::*ord*) = (λ*y*. ⨆{*x*. *f x* ≤ *y*})

lemma *ladj-radj-dual*:
fixes *f* :: '*a*::*complete-lattice-with-dual* ⇒ '*b*::*ord-with-dual*
shows *ladj f x* = ∂ (*radj* (∂$_F$ *f*) (∂ *x*))
⟨*proof*⟩

lemma *radj-ladj-dual*:
fixes *f* :: '*a*::*complete-lattice-with-dual* ⇒ '*b*::*ord-with-dual*
shows *radj f x* = ∂ (*ladj* (∂$_F$ *f*) (∂ *x*))
⟨*proof*⟩

lemma *ladj-prop*:
fixes *g* :: '*b*::*Inf* ⇒ '*a*::*ord-with-dual*
shows *ladj g* = *Inf* ∘ (− ‘) *g* ∘ ↑
⟨*proof*⟩

lemma *radj-prop*:
fixes *f* :: '*b*::*Sup* ⇒ '*a*::*ord*
shows *radj f* = *Sup* ∘ (− ‘) *f* ∘ ↓
⟨*proof*⟩

The first set of properties holds without any sort assumptions.

lemma *adj-iso1*: *f* ⊣ *g* ⟹ *mono f*
⟨*proof*⟩

lemma *adj-iso2*: *f* ⊣ *g* ⟹ *mono g*
⟨*proof*⟩

lemma *adj-comp*: *f* ⊣ *g* ⟹ *adj h k* ⟹ (*f* ∘ *h*) ⊣ (*k* ∘ *g*)
⟨*proof*⟩

lemma *adj-dual*:
fixes *f* :: '*a*::*ord-with-dual* ⇒ '*b*::*ord-with-dual*
shows *f* ⊣ *g* = (∂$_F$ *g*) ⊣ (∂$_F$ *f*)
⟨*proof*⟩

6.2 Properties for (Pre)Orders

The next set of properties holds in preorders or orders.

lemma *adj-cancel1*:
fixes $f :: 'a::preorder \Rightarrow 'b::ord$
shows $f \dashv g \Longrightarrow f \circ g \leq id$
⟨*proof*⟩

lemma *adj-cancel2*:
fixes $f :: 'a::ord \Rightarrow 'b::preorder$
shows $f \dashv g \Longrightarrow id \leq g \circ f$
⟨*proof*⟩

lemma *adj-prop*:
fixes $f :: 'a::preorder \Rightarrow 'a$
shows $f \dashv g \Longrightarrow f \circ g \leq g \circ f$
⟨*proof*⟩

lemma *adj-cancel-eq1*:
fixes $f :: 'a::preorder \Rightarrow 'b::order$
shows $f \dashv g \Longrightarrow f \circ g \circ f = f$
⟨*proof*⟩

lemma *adj-cancel-eq2*:
fixes $f :: 'a::order \Rightarrow 'b::preorder$
shows $f \dashv g \Longrightarrow g \circ f \circ g = g$
⟨*proof*⟩

lemma *adj-idem1*:
fixes $f :: 'a::preorder \Rightarrow 'b::order$
shows $f \dashv g \Longrightarrow (f \circ g) \circ (f \circ g) = f \circ g$
⟨*proof*⟩

lemma *adj-idem2*:
fixes $f :: 'a::order \Rightarrow 'b::preorder$
shows $f \dashv g \Longrightarrow (g \circ f) \circ (g \circ f) = g \circ f$
⟨*proof*⟩

lemma *adj-iso3*:
fixes $f :: 'a::order \Rightarrow 'b::order$
shows $f \dashv g \Longrightarrow mono\ (f \circ g)$
⟨*proof*⟩

lemma *adj-iso4*:
fixes $f :: 'a::order \Rightarrow 'b::order$
shows $f \dashv g \Longrightarrow mono\ (g \circ f)$
⟨*proof*⟩

lemma *adj-canc1*:

fixes $f :: 'a{::}order \Rightarrow 'b{::}ord$
shows $f \dashv g \Longrightarrow ((f \circ g)\ x = (f \circ g)\ y \longrightarrow g\ x = g\ y)$
⟨*proof*⟩

lemma *adj-canc2*:
fixes $f :: 'a{::}ord \Rightarrow 'b{::}order$
shows $f \dashv g \Longrightarrow ((g \circ f)\ x = (g \circ f)\ y \longrightarrow f\ x = f\ y)$
⟨*proof*⟩

lemma *adj-sur-inv*:
fixes $f :: 'a{::}preorder \Rightarrow 'b{::}order$
shows $f \dashv g \Longrightarrow ((surj\ f) = (f \circ g = id))$
⟨*proof*⟩

lemma *adj-surj-inj*:
fixes $f :: 'a{::}order \Rightarrow 'b{::}order$
shows $f \dashv g \Longrightarrow ((surj\ f) = (inj\ g))$
⟨*proof*⟩

lemma *adj-inj-inv*:
fixes $f :: 'a{::}preorder \Rightarrow 'b{::}order$
shows $f \dashv g \Longrightarrow ((inj\ f) = (g \circ f = id))$
⟨*proof*⟩

lemma *adj-inj-surj*:
fixes $f :: 'a{::}order \Rightarrow 'b{::}order$
shows $f \dashv g \Longrightarrow ((inj\ f) = (surj\ g))$
⟨*proof*⟩

lemma *surj-id-the-inv*: $surj\ f \Longrightarrow g \circ f = id \Longrightarrow g = the\text{-}inv\ f$
⟨*proof*⟩

lemma *inj-id-the-inv*: $inj\ f \Longrightarrow f \circ g = id \Longrightarrow f = the\text{-}inv\ g$
⟨*proof*⟩

6.3 Properties for Complete Lattices

The next laws state that a function between complete lattices preserves infs if and only if it has a lower adjoint.

lemma *radj-Inf-pres*:
fixes $g :: 'b{::}complete\text{-}lattice \Rightarrow 'a{::}complete\text{-}lattice$
shows $(\exists f.\ f \dashv g) \Longrightarrow Inf\text{-}pres\ g$
⟨*proof*⟩

lemma *ladj-Sup-pres*:
fixes $f :: 'a{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'b{::}complete\text{-}lattice\text{-}with\text{-}dual$
shows $(\exists g.\ f \dashv g) \Longrightarrow Sup\text{-}pres\ f$
⟨*proof*⟩

lemma *radj-adj*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'b{::}complete\text{-}lattice$
shows $f \dashv g \Longrightarrow g = (radj\ f)$
⟨*proof*⟩

lemma *ladj-adj*:
fixes $g :: 'b{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'a{::}complete\text{-}lattice\text{-}with\text{-}dual$
shows $f \dashv g \Longrightarrow f = (ladj\ g)$
⟨*proof*⟩

lemma *Inf-pres-radj-aux*:
fixes $g :: 'a{::}complete\text{-}lattice \Rightarrow 'b{::}complete\text{-}lattice$
shows $Inf\text{-}pres\ g \Longrightarrow (ladj\ g) \dashv g$
⟨*proof*⟩

lemma *Sup-pres-ladj-aux*:
fixes $f :: 'a{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'b{::}complete\text{-}lattice\text{-}with\text{-}dual$
shows $Sup\text{-}pres\ f \Longrightarrow f \dashv (radj\ f)$
⟨*proof*⟩

lemma *Inf-pres-radj*:
fixes $g :: 'b{::}complete\text{-}lattice \Rightarrow 'a{::}complete\text{-}lattice$
shows $Inf\text{-}pres\ g \Longrightarrow (\exists f.\ f \dashv g)$
⟨*proof*⟩

lemma *Sup-pres-ladj*:
fixes $f :: 'a{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'b{::}complete\text{-}lattice\text{-}with\text{-}dual$
shows $Sup\text{-}pres\ f \Longrightarrow (\exists g.\ f \dashv g)$
⟨*proof*⟩

lemma *Inf-pres-upper-adj-eq*:
fixes $g :: 'b{::}complete\text{-}lattice \Rightarrow 'a{::}complete\text{-}lattice$
shows $(Inf\text{-}pres\ g) = (\exists f.\ f \dashv g)$
⟨*proof*⟩

lemma *Sup-pres-ladj-eq*:
fixes $f :: 'a{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'b{::}complete\text{-}lattice\text{-}with\text{-}dual$
shows $(Sup\text{-}pres\ f) = (\exists g.\ f \dashv g)$
⟨*proof*⟩

lemma *Sup-downset-adj*: $(Sup{::}'a{::}complete\text{-}lattice\ set \Rightarrow 'a) \dashv \downarrow$
⟨*proof*⟩

lemma *Sup-downset-adj-var*: $(Sup\ (X{::}'a{::}complete\text{-}lattice\ set) \leq y) = (X \subseteq \downarrow y)$
⟨*proof*⟩

Once again many statements arise by duality, which Isabelle usually picks up.

end

7 Fixpoint Fusion

theory *Fixpoint-Fusion*
imports *Galois-Connections*

begin

Least and greatest fixpoint fusion laws for adjoints in a Galois connection, including some variants, are proved in this section. Again, the laws for least and greatest fixpoints are duals.

lemma *lfp-Fix*:
fixes $f :: 'a$::*complete-lattice-with-dual* $\Rightarrow 'a$
shows *mono* $f \Longrightarrow$ *lfp* $f = \bigsqcap$ (*Fix* f)
⟨*proof*⟩

lemma *gfp-Fix*:
fixes $f :: 'a$::*complete-lattice-with-dual* $\Rightarrow 'a$
shows *mono* $f \Longrightarrow$ *gfp* $f = \bigsqcup$ (*Fix* f)
⟨*proof*⟩

lemma *gfp-little-fusion*:
fixes $f :: 'a$::*complete-lattice* $\Rightarrow 'a$
and $g :: 'b$::*complete-lattice* $\Rightarrow 'b$
assumes *mono* f
assumes $h \circ f \leq g \circ h$
shows h (*gfp* f) $\leq$ *gfp* g
⟨*proof*⟩

lemma *lfp-little-fusion*:
fixes $f :: 'a$::*complete-lattice-with-dual* $\Rightarrow 'a$
and $g :: 'b$::*complete-lattice-with-dual* $\Rightarrow 'b$
assumes *mono* f
assumes $g \circ h \leq h \circ f$
shows *lfp* $g \leq h$ (*lfp* f)
⟨*proof*⟩

lemma *gfp-fusion*:
fixes $f :: 'a$::*complete-lattice* $\Rightarrow 'a$
and $g :: 'b$::*complete-lattice* $\Rightarrow 'b$
assumes $\exists f.\ f \dashv h$
and *mono* f
and *mono* g
and $h \circ f = g \circ h$
shows h (*gfp* f) = *gfp* g
⟨*proof*⟩

lemma *lfp-fusion*:
fixes $f :: 'a$::*complete-lattice-with-dual* $\Rightarrow 'a$
and $g :: 'b$::*complete-lattice-with-dual* $\Rightarrow 'b$

assumes $\exists f.\ h \dashv f$
and *mono f*
and *mono g*
and $h \circ f = g \circ h$
shows $h\ (lfp\ f) = lfp\ g$
⟨*proof*⟩

lemma *gfp-fusion-inf-pres*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
and $g :: 'b{::}complete\text{-}lattice \Rightarrow 'b$
assumes *Inf-pres h*
and *mono f*
and *mono g*
and $h \circ f = g \circ h$
shows $h\ (gfp\ f) = gfp\ g$
⟨*proof*⟩

lemma *lfp-fusion-sup-pres*:
fixes $f :: 'a{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'a$
and $g :: 'b{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'b$
assumes *Sup-pres h*
and *mono f*
and *mono g*
and $h \circ f = g \circ h$
shows $h\ (lfp\ f) = lfp\ g$
⟨*proof*⟩

The following facts are usueful for the semantics of isotone predicate transformers. A dual statement for least fixpoints can be proved, but is not spelled out here.

lemma *k-adju*:
fixes $k :: 'a{::}order \Rightarrow 'b{::}complete\text{-}lattice$
shows $\exists F.\forall x.\ (F{::}'b \Rightarrow 'a \Rightarrow 'b) \dashv (\lambda k.\ k\ y)$
⟨*proof*⟩

lemma *k-adju-var*: $\exists F.\ \forall x.\forall f{::}'a{::}order \Rightarrow 'b{::}complete\text{-}lattice.\ (F\ x \leq f) = (x \leq (\lambda k.\ k\ y)\ f)$
⟨*proof*⟩

lemma *gfp-fusion-var*:
fixes $F :: ('a{::}order \Rightarrow 'b{::}complete\text{-}lattice) \Rightarrow 'a \Rightarrow 'b$
and $g :: 'b \Rightarrow 'b$
assumes *mono F*
and *mono g*
and $\forall h.\ F\ h\ x = g\ (h\ x)$
shows $gfp\ F\ x = gfp\ g$
⟨*proof*⟩

This time, Isabelle is picking up dualities rather inconsistently.

end

8 Closure and Co-Closure Operators

theory *Closure-Operators*
imports *Galois-Connections*

begin

8.1 Closure Operators

Closure and coclosure operators in orders and complete lattices are defined in this section, and some basic properties are proved. Isabelle infers the appropriate types. Facts are taken mainly from the Compendium of Continuous Lattices [3] and Rosenthal's book on quantales [10].

definition *clop* :: ('*a*::*order* ⇒ '*a*) ⇒ *bool* **where**
clop f = (*id* ≤ *f* ∧ *mono f* ∧ *f* ∘ *f* ≤ *f*)

lemma *clop-extensive*: *clop f* ⟹ *id* ≤ *f*
⟨*proof*⟩

lemma *clop-extensive-var*: *clop f* ⟹ *x* ≤ *f x*
⟨*proof*⟩

lemma *clop-iso*: *clop f* ⟹ *mono f*
⟨*proof*⟩

lemma *clop-iso-var*: *clop f* ⟹ *x* ≤ *y* ⟹ *f x* ≤ *f y*
⟨*proof*⟩

lemma *clop-idem*: *clop f* ⟹ *f* ∘ *f* = *f*
⟨*proof*⟩

lemma *clop-Fix-range*: *clop f* ⟹ (*Fix f* = *range f*)
⟨*proof*⟩

lemma *clop-idem-var*: *clop f* ⟹ *f* (*f x*) = *f x*
⟨*proof*⟩

lemma *clop-Inf-closed-var*:
fixes *f* :: '*a*::*complete-lattice* ⇒ '*a*
shows *clop f* ⟹ *f* ∘ *Inf* ∘ (‘) *f* = *Inf* ∘ (‘) *f*
⟨*proof*⟩

lemma *clop-top*:
fixes *f* :: '*a*::*complete-lattice* ⇒ '*a*
shows *clop f* ⟹ *f* ⊤ = ⊤
⟨*proof*⟩

lemma *clop* (f::$'a$::*complete-lattice* $\Rightarrow$ $'a$) $\Longrightarrow$ f ($\bigsqcup x \in X.\ f\ x$) = ($\bigsqcup x \in X.\ f\ x$)
⟨*proof*⟩

lemma *clop* (f::$'a$::*complete-lattice* $\Rightarrow$ $'a$) $\Longrightarrow$ f ($f\ x \sqcup f\ y$) = $f\ x \sqcup f\ y$
⟨*proof*⟩

lemma *clop* (f::$'a$::*complete-lattice* $\Rightarrow$ $'a$) $\Longrightarrow$ $f\ \bot = \bot$
⟨*proof*⟩

lemma *clop* (f::$'a$ *set* $\Rightarrow$ $'a$ *set*) $\Longrightarrow$ f ($\bigsqcup x \in X.\ f\ x$) = ($\bigsqcup x \in X.\ f\ x$)
⟨*proof*⟩

lemma *clop* (f::$'a$ *set* $\Rightarrow$ $'a$ *set*) $\Longrightarrow$ f ($f\ x \sqcup f\ y$) = $f\ x \sqcup f\ y$
⟨*proof*⟩

lemma *clop* (f::$'a$ *set* $\Rightarrow$ $'a$ *set*) $\Longrightarrow$ $f\ \bot = \bot$
⟨*proof*⟩

lemma *clop-closure*: *clop* f $\Longrightarrow$ ($x \in$ *range* f) = ($f\ x = x$)
⟨*proof*⟩

lemma *clop-closure-set*: *clop* f $\Longrightarrow$ *range* f = *Fix* f
⟨*proof*⟩

lemma *clop-closure-prop*: (*clop*::($'a$::*complete-lattice-with-dual*$\Rightarrow$ $'a$) $\Rightarrow$ *bool*) (*Inf* $\circ$ $\uparrow$)
⟨*proof*⟩

lemma *clop-closure-prop-var*: *clop* (λx::$'a$::*complete-lattice*. $\bigsqcap\{y.\ x \leq y\}$)
⟨*proof*⟩

lemma *clop-alt*: (*clop* f) = ($\forall\, x\ y.\ x \leq f\ y \longleftrightarrow f\ x \leq f\ y$)
⟨*proof*⟩

Finally it is shown that adjoints in a Galois connection yield closure operators.

lemma *clop-adj*:
fixes f :: $'a$::*order* $\Rightarrow$ $'b$::*order*
shows $f \dashv g \Longrightarrow$ *clop* ($g \circ f$)
⟨*proof*⟩

Closure operators are monads for posets, and monads arise from adjunctions. This fact is not formalised at this point. But here is the first step: every function can be decomposed into a surjection followed by an injection.

definition *surj-on* $f\ Y$ = ($\forall\, y \in Y.\ \exists\, x.\ y = f\ x$)

lemma *surj-surj-on*: *surj* f $\Longrightarrow$ *surj-on* $f\ Y$
⟨*proof*⟩

lemma *fun-surj-inj*: $\exists\, g\ h.\ f = g \circ h \wedge$ *surj-on* h (*range* f) $\wedge$ *inj-on* g (*range* f)
⟨*proof*⟩

Connections between downsets, upsets and closure operators are outlined next.

lemma *preorder-clop*: *clop* (⇓::'*a*::*preorder set* ⇒ '*a set*)
⟨*proof*⟩

lemma *clop-preorder-aux*: *clop* $f \Longrightarrow (x \in f\ \{y\} \longleftrightarrow f\ \{x\} \subseteq f\ \{y\})$
⟨*proof*⟩

lemma *clop-preorder*: *clop* $f \Longrightarrow$ *class.preorder* $(\lambda x\ y.\ f\ \{x\} \subseteq f\ \{y\})$ $(\lambda x\ y.\ f\ \{x\} \subset f\ \{y\})$
⟨*proof*⟩

lemma *preorder-clop-dual*: *clop* (⇑::'*a*::*preorder-with-dual set* ⇒ '*a set*)
⟨*proof*⟩

The closed elements of any closure operator over a complete lattice form an Inf-closed set (a Moore family).

lemma *clop-Inf-closed*:
fixes f :: '*a*::*complete-lattice* ⇒ '*a*
shows *clop* $f \Longrightarrow$ *Inf-closed-set* (*Fix* f)
⟨*proof*⟩

lemma *clop-top-Fix*:
fixes f :: '*a*::*complete-lattice* ⇒ '*a*
shows *clop* $f \Longrightarrow \top \in$ *Fix* f
⟨*proof*⟩

Conversely, every Inf-closed subset of a complete lattice is the set of fixpoints of some closure operator.

lemma *Inf-closed-clop*:
fixes X :: '*a*::*complete-lattice set*
shows *Inf-closed-set* $X \Longrightarrow$ *clop* $(\lambda y.\ \bigsqcap\{x \in X.\ y \leq x\})$
⟨*proof*⟩

lemma *Inf-closed-clop-var*:
fixes X :: '*a*::*complete-lattice set*
shows *clop* $f \Longrightarrow \forall\, x \in X.\ x \in$ *range* $f \Longrightarrow \bigsqcap X \in$ *range* f
⟨*proof*⟩

It is well known that downsets and upsets over an ordering form subalgebras of the complete powerset lattice.

typedef (**overloaded**) '*a downsets* = *range* (⇓::'*a*::*order set* ⇒ '*a set*)
⟨*proof*⟩

setup-lifting *type-definition-downsets*

typedef (**overloaded**) *'a upsets = range* (⇑::*'a*::*order set* ⇒ *'a set*)
⟨*proof*⟩

setup-lifting *type-definition-upsets*

instantiation *downsets* :: (*order*) *Inf-lattice*
begin

lift-definition *Inf-downsets* :: *'a downsets set* ⇒ *'a downsets* **is** *Abs-downsets* ∘ *Inf* ∘ (‘) *Rep-downsets*⟨*proof*⟩

lift-definition *less-eq-downsets* :: *'a downsets* ⇒ *'a downsets* ⇒ *bool* **is** λ*X Y*. *Rep-downsets X* ⊆ *Rep-downsets Y*⟨*proof*⟩

lift-definition *less-downsets* :: *'a downsets* ⇒ *'a downsets* ⇒ *bool* **is** λ*X Y*. *Rep-downsets X* ⊂ *Rep-downsets Y*⟨*proof*⟩

instance
⟨*proof*⟩

end

instantiation *upsets* :: (*order-with-dual*) *Inf-lattice*
begin

lift-definition *Inf-upsets* :: *'a upsets set* ⇒ *'a upsets* **is** *Abs-upsets* ∘ *Inf* ∘ (‘) *Rep-upsets*⟨*proof*⟩

lift-definition *less-eq-upsets* :: *'a upsets* ⇒ *'a upsets* ⇒ *bool* **is** λ*X Y*. *Rep-upsets X* ⊆ *Rep-upsets Y*⟨*proof*⟩

lift-definition *less-upsets* :: *'a upsets* ⇒ *'a upsets* ⇒ *bool* **is** λ*X Y*. *Rep-upsets X* ⊂ *Rep-upsets Y*⟨*proof*⟩

instance
⟨*proof*⟩

end

It has already been shown in the section on representations that the map ds, which maps elements of the order to its downset, is an order embedding. However, the duality between the underlying ordering and the lattices of up- and down-closed sets as categories can probably not be expressed, as there is no easy access to contravariant functors.

8.2 Co-Closure Operators

Next, the co-closure (or kernel) operation satisfies dual laws.

definition *coclop* :: (*'a*::*order* ⇒ *'a*::*order*) ⇒ *bool* **where**
coclop f = (*f* ≤ *id* ∧ *mono f* ∧ *f* ≤ *f* ∘ *f*)

lemma *coclop-dual*: (*coclop*::(*'a*::*order-with-dual* ⇒ *'a*) ⇒ *bool*) = *clop* ∘ ∂$_F$
⟨*proof*⟩

lemma *coclop-dual-var*:
fixes *f* :: *'a*::*order-with-dual* ⇒ *'a*
shows *coclop f* = *clop* (∂$_F$ *f*)
⟨*proof*⟩

lemma *clop-dual*: (*clop*::(*'a*::*order-with-dual* ⇒ *'a*) ⇒ *bool*) = *coclop* ∘ ∂$_F$
⟨*proof*⟩

lemma *clop-dual-var*:
fixes *f* :: *'a*::*order-with-dual* ⇒ *'a*
shows *clop f* = *coclop* (∂$_F$ *f*)
⟨*proof*⟩

lemma *coclop-coextensive*: *coclop f* ⟹ *f* ≤ *id*
⟨*proof*⟩

lemma *coclop-coextensive-var*: *coclop f* ⟹ *f x* ≤ *x*
⟨*proof*⟩

lemma *coclop-iso*: *coclop f* ⟹ *mono f*
⟨*proof*⟩

lemma *coclop-iso-var*: *coclop f* ⟹ (*x* ≤ *y* ⟶ *f x* ≤ *f y*)
⟨*proof*⟩

lemma *coclop-idem*: *coclop f* ⟹ *f* ∘ *f* = *f*
⟨*proof*⟩

lemma *coclop-closure*: *coclop f* ⟹ (*x* ∈ *range f*) = (*f x* = *x*)
⟨*proof*⟩

lemma *coclop-Fix-range*: *coclop f* ⟹ (*Fix f* = *range f*)
⟨*proof*⟩

lemma *coclop-idem-var*: *coclop f* ⟹ *f* (*f x*) = *f x*
⟨*proof*⟩

lemma *coclop-Sup-closed-var*:
fixes *f* :: *'a*::*complete-lattice-with-dual* ⇒ *'a*
shows *coclop f* ⟹ *f* ∘ *Sup* ∘ (‘) *f* = *Sup* ∘ (‘) *f*

⟨*proof*⟩

lemma *Sup-closed-coclop-var*:
fixes $X :: 'a::complete\text{-}lattice\ set$
shows $coclop\ f \Longrightarrow \forall x \in X.\ x \in range\ f \Longrightarrow \bigsqcup X \in range\ f$
⟨*proof*⟩

lemma *coclop-bot*:
fixes $f :: 'a::complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'a$
shows $coclop\ f \Longrightarrow f\ \bot = \bot$
⟨*proof*⟩

lemma $coclop\ (f::'a::complete\text{-}lattice \Rightarrow 'a) \Longrightarrow f\ (\bigsqcap x \in X.\ f\ x) = (\bigsqcap x \in X.\ f\ x)$
⟨*proof*⟩

lemma $coclop\ (f::'a::complete\text{-}lattice \Rightarrow 'a) \Longrightarrow f\ (f\ x \sqcap f\ y) = f\ x \sqcap f\ y$
⟨*proof*⟩

lemma $coclop\ (f::'a::complete\text{-}lattice \Rightarrow 'a) \Longrightarrow f\ \top = \top$
⟨*proof*⟩

lemma $coclop\ (f::'a\ set \Rightarrow 'a\ set) \Longrightarrow f\ (\bigsqcap x \in X.\ f\ x) = (\bigsqcap x \in X.\ f\ x)$
⟨*proof*⟩

lemma $coclop\ (f::'a\ set \Rightarrow 'a\ set) \Longrightarrow f\ (f\ x \sqcap f\ y) = f\ x \sqcap f\ y$
⟨*proof*⟩

lemma $coclop\ (f::'a\ set \Rightarrow 'a\ set) \Longrightarrow f\ \top = \top$
⟨*proof*⟩

lemma *coclop-coclosure*: $coclop\ f \Longrightarrow f\ x = x \longleftrightarrow x \in range\ f$
⟨*proof*⟩

lemma *coclop-coclosure-set*: $coclop\ f \Longrightarrow range\ f = Fix\ f$
⟨*proof*⟩

lemma *coclop-coclosure-prop*: $(coclop::('a::complete\text{-}lattice \Rightarrow 'a) \Rightarrow bool)\ (Sup \circ \downarrow)$
⟨*proof*⟩

lemma *coclop-coclosure-prop-var*: $coclop\ (\lambda x::'a::complete\text{-}lattice.\ \bigsqcup\{y.\ y \leq x\})$
⟨*proof*⟩

lemma *coclop-alt*: $(coclop\ f) = (\forall x\ y.\ f\ x \leq y \longleftrightarrow f\ x \leq f\ y)$
⟨*proof*⟩

lemma *coclop-adj*:
fixes $f :: 'a::order \Rightarrow 'b::order$

shows $f \dashv g \Longrightarrow$ *coclop* $(f \circ g)$
⟨*proof*⟩

Finally, a subset of a complete lattice is Sup-closed if and only if it is the set of fixpoints of some co-closure operator.

lemma *coclop-Sup-closed*:
fixes $f :: 'a$::*complete-lattice* $\Rightarrow 'a$
shows *coclop* $f \Longrightarrow$ *Sup-closed-set* (*Fix* f)
⟨*proof*⟩

lemma *Sup-closed-coclop*:
fixes $X :: 'a$::*complete-lattice set*
shows *Sup-closed-set* $X \Longrightarrow$ *coclop* $(\lambda y. \bigsqcup \{x \in X.\ x \leq y\})$
⟨*proof*⟩

8.3 Complete Lattices of Closed Elements

The machinery developed allows showing that the closed elements in a complete lattice (with respect to some closure operation) form themselves a complete lattice.

class *cl-op* = *ord* +
fixes *cl-op* :: $'a \Rightarrow 'a$
assumes *clop-ext*: $x \leq$ *cl-op* x
and *clop-iso*: $x \leq y \Longrightarrow$ *cl-op* $x \leq$ *cl-op* y
and *clop-wtrans*: *cl-op* (*cl-op* x) $\leq$ *cl-op* x

class *clattice-with-clop* = *complete-lattice* + *cl-op*

begin

lemma *clop-cl-op*: *clop cl-op*
⟨*proof*⟩

lemma *clop-idem* [*simp*]: *cl-op* $\circ$ *cl-op* = *cl-op*
⟨*proof*⟩

lemma *clop-idem-var* [*simp*]: *cl-op* (*cl-op* x) = *cl-op* x
⟨*proof*⟩

lemma *clop-range-Fix*: *range cl-op* = *Fix cl-op*
⟨*proof*⟩

lemma *Inf-closed-cl-op-var*:
fixes $X :: 'a$ *set*
shows $\forall x \in X.\ x \in$ *range cl-op* $\Longrightarrow \bigsqcap X \in$ *range cl-op*
⟨*proof*⟩

lemma *inf-closed-cl-op-var*: $x \in range\ cl\text{-}op \Longrightarrow y \in range\ cl\text{-}op \Longrightarrow x \sqcap y \in range\ cl\text{-}op$
⟨*proof*⟩

end

typedef (**overloaded**) $'a$::*clattice-with-clop cl-op-im* = *range* (*cl-op*::$'a \Rightarrow 'a$)
⟨*proof*⟩

setup-lifting *type-definition-cl-op-im*

lemma *cl-op-prop* [*iff*]: (*cl-op* ($x \sqcup y$) = *cl-op* y) = (*cl-op* (x::$'a$::*clattice-with-clop*) $\leq$ *cl-op* y)
⟨*proof*⟩

lemma *cl-op-prop-var* [*iff*]: (*cl-op* ($x \sqcup$ *cl-op* y) = *cl-op* y) = (*cl-op* (x::$'a$::*clattice-with-clop*) $\leq$ *cl-op* y)
⟨*proof*⟩

instantiation *cl-op-im* :: (*clattice-with-clop*) *complete-lattice*
begin

lift-definition *Inf-cl-op-im* :: $'a$ *cl-op-im set* $\Rightarrow$ $'a$ *cl-op-im* **is** *Inf*
⟨*proof*⟩

lift-definition *Sup-cl-op-im* :: $'a$ *cl-op-im set* $\Rightarrow$ $'a$ *cl-op-im* **is** $\lambda X.$ *cl-op* ($\bigsqcup X$)
⟨*proof*⟩

lift-definition *inf-cl-op-im* :: $'a$ *cl-op-im* $\Rightarrow$ $'a$ *cl-op-im* $\Rightarrow$ $'a$ *cl-op-im* **is** *inf*
⟨*proof*⟩

lift-definition *sup-cl-op-im* :: $'a$ *cl-op-im* $\Rightarrow$ $'a$ *cl-op-im* $\Rightarrow$ $'a$ *cl-op-im* **is** $\lambda x\ y.$ *cl-op* ($x \sqcup y$)
⟨*proof*⟩

lift-definition *less-eq-cl-op-im* :: $'a$ *cl-op-im* $\Rightarrow$ $'a$ *cl-op-im* $\Rightarrow$ *bool* **is** ($\leq$)⟨*proof*⟩

lift-definition *less-cl-op-im* :: $'a$ *cl-op-im* $\Rightarrow$ $'a$ *cl-op-im* $\Rightarrow$ *bool* **is** ($<$)⟨*proof*⟩

lift-definition *bot-cl-op-im* :: $'a$ *cl-op-im* **is** *cl-op* $\bot$
⟨*proof*⟩

lift-definition *top-cl-op-im* :: $'a$ *cl-op-im* **is** $\top$
⟨*proof*⟩

instance
⟨*proof*⟩

end

This statement is perhaps less useful as it might seem, because it is difficult to make it cooperate with concrete closure operators, which one would not generally like to define within a type class. Alternatively, a sublocale statement could perhaps be given. It would also have been nice to prove this statement for Sup-lattices—this would have cut down the number of proof obligations significantly. But this would require a tighter integration of these structures. A similar statement could have been proved for co-closure operators. But this would not lead to new insights.

Next I show that for every surjective Sup-preserving function between complete lattices there is a closure operator such that the set of closed elements is isomorphic to the range of the surjection.

lemma *surj-Sup-pres-id*:
fixes $f :: 'a{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'b{::}complete\text{-}lattice\text{-}with\text{-}dual$
assumes *surj f*
and *Sup-pres f*
shows $f \circ (radj\ f) = id$
⟨*proof*⟩

lemma *surj-Sup-pres-inj*:
fixes $f :: 'a{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'b{::}complete\text{-}lattice\text{-}with\text{-}dual$
assumes *surj f*
and *Sup-pres f*
shows *inj* (*radj f*)
⟨*proof*⟩

lemma *surj-Sup-pres-inj-on*:
fixes $f :: 'a{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'b{::}complete\text{-}lattice\text{-}with\text{-}dual$
assumes *surj f*
and *Sup-pres f*
shows *inj-on f* (*range* (*radj f* ∘ *f*))
⟨*proof*⟩

lemma *surj-Sup-pres-bij-on*:
fixes $f :: 'a{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'b{::}complete\text{-}lattice\text{-}with\text{-}dual$
assumes *surj f*
and *Sup-pres f*
shows *bij-betw f* (*range* (*radj f* ∘ *f*)) *UNIV*
⟨*proof*⟩

Thus the restriction of f to the set of closed elements is indeed a bijection. The final fact shows that it preserves Sups of closed elements, and hence is an isomorphism of complete lattices.

lemma *surj-Sup-pres-iso*:
fixes $f :: 'a{::}complete\text{-}lattice\text{-}with\text{-}dual \Rightarrow 'b{::}complete\text{-}lattice\text{-}with\text{-}dual$
assumes *surj f*

and *Sup-pres f*
shows $f\ ((\mathit{radj}\ f \circ f)\ (\bigsqcup X)) = (\bigsqcup x \in X.\ f\ x)$
⟨*proof*⟩

8.4 A Quick Example: Dedekind-MacNeille Completions

I only outline the basic construction. Additional facts about join density, and that the completion yields the least complete lattice that contains all Sups and Infs of the underlying posets, are left for future consideration.

abbreviation $\mathit{dm} \equiv \mathit{lb\text{-}set} \circ \mathit{ub\text{-}set}$

lemma *up-set-prop*: $(X::'a::\mathit{preorder}\ \mathit{set}) \neq \{\} \Longrightarrow \mathit{ub\text{-}set}\ X = \bigcap \{\uparrow x\ |x.\ x \in X\}$
⟨*proof*⟩

lemma *lb-set-prop*: $(X::'a::\mathit{preorder}\ \mathit{set}) \neq \{\} \Longrightarrow \mathit{lb\text{-}set}\ X = \bigcap \{\downarrow x\ |x.\ x \in X\}$
⟨*proof*⟩

lemma *dm-downset-var*: $\mathit{dm}\ \{x\} = \downarrow(x::'a::\mathit{preorder})$
⟨*proof*⟩

lemma *dm-downset*: $\mathit{dm} \circ \eta = (\downarrow::'a::\mathit{preorder} \Rightarrow 'a\ \mathit{set})$
⟨*proof*⟩

lemma *dm-inj*: $\mathit{inj}\ ((\mathit{dm}::'a::\mathit{order}\ \mathit{set} \Rightarrow 'a\ \mathit{set}) \circ \eta)$
⟨*proof*⟩

lemma *clop* $(\mathit{lb\text{-}set} \circ \mathit{ub\text{-}set})$
⟨*proof*⟩

end

9 Locale-Based Duality

theory *Order-Lattice-Props-Loc*
imports *Main*
begin

unbundle *lattice-syntax*

This section explores order and lattice duality based on locales. Used within the context of a class or locale, this is very effective, though more opaque than the previous approach. Outside of such a context, however, it apparently cannot be used for dualising theorems. Examples are properties of functions between orderings or lattices.

definition $\mathit{Fix} :: ('a \Rightarrow 'a) \Rightarrow 'a\ \mathit{set}$ **where**
$\mathit{Fix}\ f = \{x.\ f\ x = x\}$

context *ord*
begin

definition *min-set* :: *'a set* ⇒ *'a set* **where**
min-set X = {*y* ∈ *X*. ∀ *x* ∈ *X*. *x* ≤ *y* ⟶ *x* = *y*}

definition *max-set* :: *'a set* ⇒ *'a set* **where**
max-set X = {*x* ∈ *X*. ∀ *y* ∈ *X*. *x* ≤ *y* ⟶ *x* = *y*}

definition *directed* :: *'a set* ⇒ *bool* **where**
directed X = (∀ *Y*. *finite Y* ∧ *Y* ⊆ *X* ⟶ (∃ *x* ∈ *X*. ∀ *y* ∈ *Y*. *y* ≤ *x*))

definition *filtered* :: *'a set* ⇒ *bool* **where**
filtered X = (∀ *Y*. *finite Y* ∧ *Y* ⊆ *X* ⟶ (∃ *x* ∈ *X*. ∀ *y* ∈ *Y*. *x* ≤ *y*))

definition *downset-set* :: *'a set* ⇒ *'a set* (‹⇓›) **where**
⇓*X* = {*y*. ∃ *x* ∈ *X*. *y* ≤ *x*}

definition *upset-set* :: *'a set* ⇒ *'a set* (‹⇑›) **where**
⇑*X* = {*y*. ∃ *x* ∈ *X*. *x* ≤ *y*}

definition *downset* :: *'a* ⇒ *'a set* (‹↓›) **where**
↓ = ⇓ ∘ (λ*x*. {*x*})

definition *upset* :: *'a* ⇒ *'a set* (‹↑›) **where**
↑ = ⇑ ∘ (λ*x*. {*x*})

definition *downsets* :: *'a set set* **where**
downsets = *Fix* ⇓

definition *upsets* :: *'a set set* **where**
upsets = *Fix* ⇑

abbreviation *downset-setp X* ≡ *X* ∈ *downsets*

abbreviation *upset-setp X* ≡ *X* ∈ *upsets*

definition *ideals* :: *'a set set* **where**
ideals = {*X*. *X* ≠ {} ∧ *downset-setp X* ∧ *directed X*}

definition *filters* :: *'a set set* **where**
filters = {*X*. *X* ≠ {} ∧ *upset-setp X* ∧ *filtered X*}

abbreviation *idealp X* ≡ *X* ∈ *ideals*

abbreviation *filterp X* ≡ *X* ∈ *filters*

end

abbreviation *Sup-pres* :: $('a{::}Sup \Rightarrow 'b{::}Sup) \Rightarrow bool$ **where**
Sup-pres f $\equiv f \circ Sup = Sup \circ (`)\ f$

abbreviation *Inf-pres* :: $('a{::}Inf \Rightarrow 'b{::}Inf) \Rightarrow bool$ **where**
Inf-pres f $\equiv f \circ Inf = Inf \circ (`)\ f$

abbreviation *sup-pres* :: $('a{::}sup \Rightarrow 'b{::}sup) \Rightarrow bool$ **where**
sup-pres f $\equiv (\forall\, x\ y.\ f\ (x \sqcup y) = f\ x \sqcup f\ y)$

abbreviation *inf-pres* :: $('a{::}inf \Rightarrow 'b{::}inf) \Rightarrow bool$ **where**
inf-pres f $\equiv (\forall\, x\ y.\ f\ (x \sqcap y) = f\ x \sqcap f\ y)$

abbreviation *bot-pres* :: $('a{::}bot \Rightarrow 'b{::}bot) \Rightarrow bool$ **where**
bot-pres f $\equiv f\ \bot = \bot$

abbreviation *top-pres* :: $('a{::}top \Rightarrow 'b{::}top) \Rightarrow bool$ **where**
top-pres f $\equiv f\ \top = \top$

abbreviation *Sup-dual* :: $('a{::}Sup \Rightarrow 'b{::}Inf) \Rightarrow bool$ **where**
Sup-dual f $\equiv f \circ Sup = Inf \circ (`)\ f$

abbreviation *Inf-dual* :: $('a{::}Inf \Rightarrow 'b{::}Sup) \Rightarrow bool$ **where**
Inf-dual f $\equiv f \circ Inf = Sup \circ (`)\ f$

abbreviation *sup-dual* :: $('a{::}sup \Rightarrow 'b{::}inf) \Rightarrow bool$ **where**
sup-dual f $\equiv (\forall\, x\ y.\ f\ (x \sqcup y) = f\ x \sqcap f\ y)$

abbreviation *inf-dual* :: $('a{::}inf \Rightarrow 'b{::}sup) \Rightarrow bool$ **where**
inf-dual f $\equiv (\forall\, x\ y.\ f\ (x \sqcap y) = f\ x \sqcup f\ y)$

abbreviation *bot-dual* :: $('a{::}bot \Rightarrow 'b{::}top) \Rightarrow bool$ **where**
bot-dual f $\equiv f\ \bot = \top$

abbreviation *top-dual* :: $('a{::}top \Rightarrow 'b{::}bot) \Rightarrow bool$ **where**
top-dual f $\equiv f\ \top = \bot$

9.1 Duality via Locales

sublocale *ord* $\subseteq$ *dual-ord*: *ord* $(\geq)$ $(>)$
rewrites *dual-max-set*: *max-set* = *dual-ord.min-set*
and *dual-filtered*: *filtered* = *dual-ord.directed*
and *dual-upset-set*: *upset-set* = *dual-ord.downset-set*
and *dual-upset*: *upset* = *dual-ord.downset*
and *dual-upsets*: *upsets* = *dual-ord.downsets*
and *dual-filters*: *filters* = *dual-ord.ideals*
⟨*proof*⟩

sublocale *preorder* $\subseteq$ *dual-preorder*: *preorder* $(\geq)$ $(>)$
⟨*proof*⟩

sublocale *order* $\subseteq$ *dual-order*: *order* $(\geq)$ $(>)$
⟨*proof*⟩

sublocale *lattice* $\subseteq$ *dual-lattice*: *lattice sup* $(\geq)$ $(>)$ *inf*
⟨*proof*⟩

sublocale *bounded-lattice* $\subseteq$ *dual-bounded-lattice*: *bounded-lattice sup* $(\geq)$ $(>)$ *inf* $\top$ $\bot$
⟨*proof*⟩

sublocale *boolean-algebra* $\subseteq$ *dual-boolean-algebra*: *boolean-algebra* $\lambda x\ y.\ x \sqcup -y$ *uminus sup* $(\geq)$ $(>)$ *inf* $\top$ $\bot$
⟨*proof*⟩

sublocale *complete-lattice* $\subseteq$ *dual-complete-lattice*: *complete-lattice Sup Inf sup* $(\geq)$ $(>)$ *inf* $\top$ $\bot$
rewrites *dual-gfp*: *gfp* = *dual-complete-lattice.lfp*
⟨*proof*⟩

context *ord*
begin

lemma *dual-min-set*: *min-set* = *dual-ord.max-set*
⟨*proof*⟩

lemma *dual-directed*: *directed* = *dual-ord.filtered*
⟨*proof*⟩

lemma *dual-downset*: *downset* = *dual-ord.upset*
⟨*proof*⟩

lemma *dual-downset-set*: *downset-set* = *dual-ord.upset-set*
⟨*proof*⟩

lemma *dual-downsets*: *downsets* = *dual-ord.upsets*
⟨*proof*⟩

lemma *dual-ideals*: *ideals* = *dual-ord.filters*
⟨*proof*⟩

end

context *complete-lattice*
begin

lemma *dual-lfp*: *lfp* = *dual-complete-lattice.gfp*
⟨*proof*⟩

end

9.2 Properties of Orderings, Again

context *ord*
begin

lemma *directed-nonempty*: *directed* $X \Longrightarrow X \neq \{\}$
⟨*proof*⟩

lemma *directed-ub*: *directed* $X \Longrightarrow (\forall\, x \in X.\ \forall\, y \in X.\ \exists\, z \in X.\ x \leq z \wedge y \leq z)$
⟨*proof*⟩

lemma *downset-set-prop*: $\Downarrow = \mathit{Union} \circ (`) \downarrow$
⟨*proof*⟩

lemma *downset-set-prop-var*: $\Downarrow X = (\bigcup x \in X.\ \downarrow x)$
⟨*proof*⟩

lemma *downset-prop*: $\downarrow x = \{y.\ y \leq x\}$
⟨*proof*⟩

end

context *preorder*
begin

lemma *directed-prop*: $X \neq \{\} \Longrightarrow (\forall\, x \in X.\ \forall\, y \in X.\ \exists\, z \in X.\ x \leq z \wedge y \leq z)$
$\Longrightarrow$ *directed* X
⟨*proof*⟩

lemma *directed-alt*: *directed* $X = (X \neq \{\} \wedge (\forall\, x \in X.\ \forall\, y \in X.\ \exists\, z \in X.\ x \leq z$
$\wedge\ y \leq z))$
⟨*proof*⟩

lemma *downset-set-ext*: $\mathit{id} \leq \Downarrow$
⟨*proof*⟩

lemma *downset-set-iso*: *mono* $\Downarrow$
⟨*proof*⟩

lemma *downset-set-idem* [*simp*]: $\Downarrow \circ \Downarrow = \Downarrow$
⟨*proof*⟩

lemma *downset-faithful*: $\downarrow x \subseteq \downarrow y \Longrightarrow x \leq y$
⟨*proof*⟩

lemma *downset-iso-iff*: $(\downarrow x \subseteq \downarrow y) = (x \leq y)$
⟨*proof*⟩

lemma *downset-directed-downset-var* [*simp*]: *directed* ($\Downarrow X$) = *directed X*
⟨*proof*⟩

lemma *downset-directed-downset* [*simp*]: *directed* ∘ $\Downarrow$ = *directed*
⟨*proof*⟩

lemma *directed-downset-ideals*: *directed* ($\Downarrow X$) = ($\Downarrow X \in$ *ideals*)
⟨*proof*⟩

end

lemma *downset-iso*: *mono* ($\downarrow$::'*a*::*order* ⇒ '*a set*)
⟨*proof*⟩

context *order*
begin

lemma *downset-inj*: *inj* $\downarrow$
⟨*proof*⟩

end

context *lattice*
begin

lemma *lat-ideals*: $X \in$ *ideals* = ($X \neq \{\} \wedge X \in$ *downsets* $\wedge$ ($\forall\, x \in X.\ \forall\ y \in X.\ x \sqcup y \in X$))
⟨*proof*⟩

end

context *bounded-lattice*
begin

lemma *bot-ideal*: $X \in$ *ideals* $\Longrightarrow \bot \in X$
⟨*proof*⟩

end

context *complete-lattice*
begin

lemma *Sup-downset-id* [*simp*]: *Sup* ∘ $\downarrow$ = *id*
⟨*proof*⟩

lemma *downset-Sup-id*: *id* $\leq \downarrow \circ$ *Sup*
⟨*proof*⟩

lemma *Inf-Sup-var*: $\bigsqcup(\bigcap x \in X.\ \downarrow x) = \bigsqcap X$
⟨*proof*⟩

lemma *Inf-pres-downset-var*: $(\bigcap x \in X.\ \downarrow x) = \downarrow(\bigsqcap X)$
⟨*proof*⟩

end

lemma *lfp-in-Fix*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $mono\ f \Longrightarrow lfp\ f \in Fix\ f$
⟨*proof*⟩

lemma *gfp-in-Fix*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $mono\ f \Longrightarrow gfp\ f \in Fix\ f$
⟨*proof*⟩

lemma *nonempty-Fix*:
fixes $f :: 'a{::}complete\text{-}lattice \Rightarrow 'a$
shows $mono\ f \Longrightarrow Fix\ f \neq \{\}$
⟨*proof*⟩

9.3 Dual Properties of Orderings from Locales

These properties can be proved very smoothly overall. But only within the context of a class or locale!

context *ord*
begin

lemma *filtered-nonempty*: $filtered\ X \Longrightarrow X \neq \{\}$
⟨*proof*⟩

lemma *filtered-lb*: $filtered\ X \Longrightarrow (\forall x \in X.\ \forall y \in X.\ \exists z \in X.\ z \leq x \wedge z \leq y)$
⟨*proof*⟩

lemma *upset-set-prop*: $\Uparrow = Union \circ (`) \uparrow$
⟨*proof*⟩

lemma *upset-set-prop-var*: $\Uparrow X = (\bigcup x \in X.\ \uparrow x)$
⟨*proof*⟩

lemma *upset-prop*: $\uparrow x = \{y.\ x \leq y\}$
⟨*proof*⟩

end

context *preorder*
begin

lemma *filtered-prop*: $X \neq \{\} \Longrightarrow (\forall\, x \in X.\ \forall\, y \in X.\ \exists\, z \in X.\ z \leq x \wedge z \leq y) \Longrightarrow$ *filtered* X
⟨*proof*⟩

lemma *filtered-alt*: *filtered* $X = (X \neq \{\} \wedge (\forall\, x \in X.\ \forall\, y \in X.\ \exists\, z \in X.\ z \leq x \wedge z \leq y))$
⟨*proof*⟩

lemma *upset-set-ext*: $id \leq \Uparrow$
⟨*proof*⟩

lemma *upset-set-anti*: *mono* $\Uparrow$
⟨*proof*⟩

lemma *up-set-idem* [*simp*]: $\Uparrow \circ \Uparrow = \Uparrow$
⟨*proof*⟩

lemma *upset-faithful*: $\uparrow x \subseteq \uparrow y \Longrightarrow y \leq x$
⟨*proof*⟩

lemma *upset-anti-iff*: $(\uparrow y \subseteq \uparrow x) = (x \leq y)$
⟨*proof*⟩

lemma *upset-filtered-upset* [*simp*]: *filtered* $\circ \Uparrow =$ *filtered*
⟨*proof*⟩

lemma *filtered-upset-filters*: *filtered* $(\Uparrow X) = (\Uparrow X \in$ *filters*$)$
⟨*proof*⟩

end

context *order*
begin

lemma *upset-inj*: *inj* $\uparrow$
⟨*proof*⟩

end

context *lattice*
begin

lemma *lat-filters*: $X \in$ *filters* $= (X \neq \{\} \wedge X \in$ *upsets* $\wedge (\forall\, x \in X.\ \forall\ y \in X.\ x \sqcap y \in X))$
⟨*proof*⟩

end

context *bounded-lattice*
begin

lemma *top-filter*: $X \in \mathit{filters} \Longrightarrow \top \in X$
⟨*proof*⟩

end

context *complete-lattice*
begin

lemma *Inf-upset-id* [*simp*]: $\mathit{Inf} \circ \uparrow = \mathit{id}$
⟨*proof*⟩

lemma *upset-Inf-id*: $\mathit{id} \leq \uparrow \circ \mathit{Inf}$
⟨*proof*⟩

lemma *Sup-Inf-var*: $\bigsqcap(\bigsqcap x \in X.\ \uparrow x) = \bigsqcup X$
⟨*proof*⟩

lemma *Sup-dual-upset-var*: $(\bigsqcap x \in X.\ \uparrow x) = \uparrow(\bigsqcup X)$
⟨*proof*⟩

end

9.4 Examples that Do Not Dualise

lemma *upset-anti*: $\mathit{antimono}\ (\uparrow :: {'a} :: \mathit{order} \Rightarrow {'a}\ \mathit{set})$
⟨*proof*⟩

context *complete-lattice*
begin

lemma *fSup-unfold*: $(f :: \mathit{nat} \Rightarrow {'a})\ 0 \sqcup (\bigsqcup n.\ f\ (\mathit{Suc}\ n)) = (\bigsqcup n.\ f\ n)$
⟨*proof*⟩

lemma *fInf-unfold*: $(f :: \mathit{nat} \Rightarrow {'a})\ 0 \sqcap (\bigsqcap n.\ f\ (\mathit{Suc}\ n)) = (\bigsqcap n.\ f\ n)$
⟨*proof*⟩

end

lemma *fun-isol*: $\mathit{mono}\ f \Longrightarrow \mathit{mono}\ ((\circ)\ f)$
⟨*proof*⟩

lemma *fun-isor*: $\mathit{mono}\ f \Longrightarrow \mathit{mono}\ (\lambda x.\ x \circ f)$
⟨*proof*⟩

lemma *Sup-sup-pres*:

fixes f :: $'a$::*complete-lattice* ⇒ $'b$::*complete-lattice*
shows *Sup-pres f* ⟹ *sup-pres f*
⟨*proof*⟩

lemma *Inf-inf-pres*:
fixes f :: $'a$::*complete-lattice* ⇒ $'b$::*complete-lattice*
shows*Inf-pres f* ⟹ *inf-pres f*
⟨*proof*⟩

lemma *Sup-bot-pres*:
fixes f :: $'a$::*complete-lattice* ⇒ $'b$::*complete-lattice*
shows *Sup-pres f* ⟹ *bot-pres f*
⟨*proof*⟩

lemma *Inf-top-pres*:
fixes f :: $'a$::*complete-lattice* ⇒ $'b$::*complete-lattice*
shows *Inf-pres f* ⟹ *top-pres f*
⟨*proof*⟩

context *complete-lattice*
begin

lemma *iso-Inf-subdistl*:
assumes *mono* (f::$'a$ ⇒ $'b$::*complete-lattice*)
shows f ∘ *Inf* ≤ *Inf* ∘ (‘) f
⟨*proof*⟩

lemma *iso-Sup-supdistl*:
assumes *mono* (f::$'a$ ⇒ $'b$::*complete-lattice*)
shows *Sup* ∘ (‘) f ≤ f ∘ *Sup*
⟨*proof*⟩

lemma *Inf-subdistl-iso*:
fixes f :: $'a$ ⇒ $'b$::*complete-lattice*
shows f ∘ *Inf* ≤ *Inf* ∘ (‘) f ⟹ *mono f*
⟨*proof*⟩

lemma *Sup-supdistl-iso*:
fixes f :: $'a$ ⇒ $'b$::*complete-lattice*
shows *Sup* ∘ (‘) f ≤ f ∘ *Sup* ⟹ *mono f*
⟨*proof*⟩

lemma *supdistl-iso*:
fixes f :: $'a$ ⇒ $'b$::*complete-lattice*
shows (*Sup* ∘ (‘) f ≤ f ∘ *Sup*) = *mono f*
⟨*proof*⟩

lemma *subdistl-iso*:
fixes f :: $'a$ ⇒ $'b$::*complete-lattice*

shows $(f \circ Inf \leq Inf \circ (`)\ f) = mono\ f$
⟨*proof*⟩

end

lemma *fSup-distr*: *Sup-pres* $(\lambda x.\ x \circ f)$
⟨*proof*⟩

lemma *fSup-distr-var*: $\bigsqcup F \circ g = (\bigsqcup f \in F.\ f \circ g)$
⟨*proof*⟩

lemma *fInf-distr*: *Inf-pres* $(\lambda x.\ x \circ f)$
⟨*proof*⟩

lemma *fInf-distr-var*: $\bigsqcap F \circ g = (\bigsqcap f \in F.\ f \circ g)$
⟨*proof*⟩

lemma *fSup-subdistl*:
assumes *mono* $(f::'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice)$
shows $Sup \circ (`)\ ((\circ)\ f) \leq (\circ)\ f \circ Sup$
⟨*proof*⟩

lemma *fSup-subdistl-var*:
fixes $f :: 'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice$
shows $mono\ f \Longrightarrow (\bigsqcup g \in G.\ f \circ g) \leq f \circ \bigsqcup G$
⟨*proof*⟩

lemma *fInf-subdistl*:
fixes $f :: 'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice$
shows $mono\ f \Longrightarrow (\circ)\ f \circ Inf \leq Inf \circ (`)\ ((\circ)\ f)$
⟨*proof*⟩

lemma *fInf-subdistl-var*:
fixes $f :: 'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice$
shows $mono\ f \Longrightarrow f \circ \bigsqcap G \leq (\bigsqcap g \in G.\ f \circ g)$
⟨*proof*⟩

lemma *Inf-pres-downset*: *Inf-pres* $(\downarrow::'a::complete\text{-}lattice \Rightarrow 'a\ set)$
⟨*proof*⟩

lemma *Sup-dual-upset*: *Sup-dual* $(\uparrow::'a::complete\text{-}lattice \Rightarrow 'a\ set)$
⟨*proof*⟩

This approach could probably be combined with the explicit functor-based one. This may be good for proofs, but seems conceptually rather ugly.

end

10 Duality Based on a Data Type

theory *Order-Lattice-Props-Wenzel*
imports *Main*
begin

unbundle *lattice-syntax*

10.1 Wenzel's Approach Revisited

This approach is similar to, but inferior to the explicit class-based one. The main caveat is that duality is not involutive with this approach, and this allows dualising less theorems.

I copy Wenzel's development [11] in this subsection and extend it with additional properties. I show only the most important properties.

datatype $'a$ *dual* = *dual* (*un-dual*: $'a$) (‹∂›)

notation *un-dual* (‹∂^-›)

lemma *dual-inj*: *inj* ∂
⟨*proof*⟩

lemma *dual-surj*: *surj* ∂
⟨*proof*⟩

lemma *dual-bij*: *bij* ∂
⟨*proof*⟩

Dual is not idempotent, and I see no way of imposing this condition. Yet at least an inverse exists — namely un-dual..

lemma *dual-inv1* [*simp*]: $\partial^- \circ \partial = id$
⟨*proof*⟩

lemma *dual-inv2* [*simp*]: $\partial \circ \partial^- = id$
⟨*proof*⟩

lemma *dual-inv-inj*: *inj* ∂^-
⟨*proof*⟩

lemma *dual-inv-surj*: *surj* ∂^-
⟨*proof*⟩

lemma *dual-inv-bij*: *bij* ∂^-
⟨*proof*⟩

lemma *dual-iff*: $(\partial\ x = y) \longleftrightarrow (x = \partial^-\ y)$
⟨*proof*⟩

Isabelle data types come with a number of generic functions.

The functor map-dual lifts functions to dual types. Isabelle's generic definition is not straightforward to understand and use. Yet conceptually it can be explained as follows.

lemma *map-dual-def-var* [*simp*]: (*map-dual*::('*a* ⇒ '*b*) ⇒ '*a dual* ⇒ '*b dual*) *f* = ∂ ∘ *f* ∘ ∂⁻
⟨*proof*⟩

lemma *map-dual-def-var2*: ∂⁻ ∘ *map-dual f* = *f* ∘ ∂⁻
⟨*proof*⟩

lemma *map-dual-func1*: *map-dual* (*f* ∘ *g*) = *map-dual f* ∘ *map-dual g*
⟨*proof*⟩

lemma *map-dual-func2* : *map-dual id* = *id*
⟨*proof*⟩

The functor map-dual has an inverse functor as well.

definition *map-dual-inv* :: ('*a dual* ⇒ '*b dual*) => ('*a* => '*b*) **where**
map-dual-inv f = ∂⁻ ∘ *f* ∘ ∂

lemma *map-dual-inv-func1*: *map-dual-inv id* = *id*
⟨*proof*⟩

lemma *map-dual-inv-func2*: *map-dual-inv* (*f* ∘ *g*) = *map-dual-inv f* ∘ *map-dual-inv g*
⟨*proof*⟩

lemma *map-dual-inv1*: *map-dual* ∘ *map-dual-inv* = *id*
⟨*proof*⟩

lemma *map-dual-inv2*: *map-dual-inv* ∘ *map-dual* = *id*
⟨*proof*⟩

Hence dual is an isomorphism between categories.

lemma *subset-dual*: (∂ ' *X* = *Y*) ⟷ (*X* = ∂⁻ ' *Y*)
⟨*proof*⟩

lemma *subset-dual1*: (*X* ⊆ *Y*) ⟷ (∂ ' *X* ⊆ ∂ ' *Y*)
⟨*proof*⟩

lemma *dual-ball*: (∀ *x* ∈ *X*. *P* (∂ *x*)) ⟷ (∀ *y* ∈ ∂ ' *X*. *P y*)
⟨*proof*⟩

lemma *dual-inv-ball*: (∀ *x* ∈ *X*. *P* (∂⁻ *x*)) ⟷ (∀ *y* ∈ ∂⁻ ' *X*. *P y*)
⟨*proof*⟩

lemma *dual-all*: $(\forall\, x.\ P\ (\partial\ x)) \longleftrightarrow (\forall\, y.\ P\ y)$
⟨*proof*⟩

lemma *dual-inv-all*: $(\forall\, x.\ P\ (\partial^{-}\ x)) \longleftrightarrow (\forall\, y.\ P\ y)$
⟨*proof*⟩

lemma *dual-ex*: $(\exists\, x.\ P\ (\partial\ x)) \longleftrightarrow (\exists\, y.\ P\ y)$
⟨*proof*⟩

lemma *dual-inv-ex*: $(\exists\, x.\ P\ (\partial^{-}\ x)) \longleftrightarrow (\exists\, y.\ P\ y)$
⟨*proof*⟩

lemma *dual-Collect*: $\{\partial\ x\ |x.\ P\ (\partial\ x)\} = \{y.\ P\ y\}$
⟨*proof*⟩

lemma *dual-inv-Collect*: $\{\partial^{-}\ x\ |x.\ P\ (\partial^{-}\ x)\} = \{y.\ P\ y\}$
⟨*proof*⟩

lemma *fun-dual1*: $(f \circ \partial = g) \longleftrightarrow (f = g \circ \partial^{-})$
⟨*proof*⟩

lemma *fun-dual2*: $(\partial \circ f = g) \longleftrightarrow (f = \partial^{-} \circ g)$
⟨*proof*⟩

lemma *fun-dual3*: $(f \circ (\,`\,) \ \partial = g) \longleftrightarrow (f = g \circ (\,`\,) \ \partial^{-})$
⟨*proof*⟩

lemma *fun-dual4*: $(f = \partial^{-} \circ g \circ (\,`\,) \ \partial) \longleftrightarrow (\partial \circ f \circ (\,`\,) \ \partial^{-} = g)$
⟨*proof*⟩

The next facts show incrementally that the dual of a complete lattice is a complete lattice. This follows once again Wenzel.

instantiation *dual* :: (*ord*) *ord*
begin

definition *less-eq-dual-def*: $(\leq) = \mathit{rel\text{-}dual}\ (\geq)$

definition *less-dual-def*: $(<) = \mathit{rel\text{-}dual}\ (>)$

instance⟨*proof*⟩

end

lemma *less-eq-dual-def-var*: $(x \leq y) = (\partial^{-}\ y \leq \partial^{-}\ x)$
⟨*proof*⟩

lemma *less-dual-def-var*: $(x < y) = (\partial^{-}\ y < \partial^{-}\ x)$
⟨*proof*⟩

instance *dual* :: (*preorder*) *preorder*
⟨*proof*⟩

instance *dual* :: (*order*) *order*
⟨*proof*⟩

lemma *dual-anti*: $x \leq y \Longrightarrow \partial\ y \leq \partial\ x$
⟨*proof*⟩

lemma *dual-anti-iff*: $(x \leq y) = (\partial\ y \leq \partial\ x)$
⟨*proof*⟩

map-dual does not map isotone functions to antitone ones. It simply lifts the type!

lemma *mono f* $\Longrightarrow$ *mono* (*map-dual f*)
⟨*proof*⟩

instantiation *dual* :: (*lattice*) *lattice*
begin

definition *inf-dual-def*: $x \sqcap y = \partial\ (\partial^-\ x \sqcup \partial^-\ y)$

definition *sup-dual-def*: $x \sqcup y = \partial\ (\partial^-\ x \sqcap \partial^-\ y)$

instance
⟨*proof*⟩

end

instantiation *dual* :: (*complete-lattice*) *complete-lattice*
begin

definition *Inf-dual-def*: $Inf = \partial \circ Sup \circ (`)\ \partial^-$

definition *Sup-dual-def*: $Sup = \partial \circ Inf \circ (`)\ \partial^-$

definition *bot-dual-def*: $\bot = \partial\ \top$

definition *top-dual-def*: $\top = \partial\ \bot$

instance
⟨*proof*⟩

end

Next, directed and filtered sets, upsets, downsets, filters and ideals in posets are defined.

context *ord*
begin

definition *directed* :: *'a set* ⇒ *bool* **where**
directed X = (∀ *Y*. *finite Y* ∧ *Y* ⊆ *X* ⟶ (∃ *x* ∈ *X*. ∀ *y* ∈ *Y*. *y* ≤ *x*))

definition *filtered* :: *'a set* ⇒ *bool* **where**
filtered X = (∀ *Y*. *finite Y* ∧ *Y* ⊆ *X* ⟶ (∃ *x* ∈ *X*. ∀ *y* ∈ *Y*. *x* ≤ *y*))

definition *downset-set* :: *'a set* ⇒ *'a set* (‹⇓›) **where**
⇓*X* = {*y*. ∃ *x* ∈ *X*. *y* ≤ *x*}

definition *upset-set* :: *'a set* ⇒ *'a set* (‹⇑›) **where**
⇑*X* = {*y*. ∃ *x* ∈ *X*. *x* ≤ *y*}

end

10.2 Examples that Do Not Dualise

Filtered and directed sets are dual.

Proofs could be simplified if dual was idempotent.

lemma *filtered-directed-dual*: *filtered* ∘ (‘) ∂ = *directed*
⟨*proof*⟩

lemma *directed-filtered-dual*: *directed* ∘ (‘) ∂ = *filtered*
⟨*proof*⟩

This example illustrates the deficiency of the approach. In the class-based approach the second proof is trivial.

The next example shows that this is a systematic problem.

lemma *downset-set-upset-set-dual*: (‘) ∂ ∘ ⇓ = ⇑ ∘ (‘) ∂
⟨*proof*⟩

lemma *upset-set-downset-set-dual*: (‘) ∂ ∘ ⇑ = ⇓ ∘ (‘) ∂
⟨*proof*⟩

end

References

[1] A. Armstrong and G. Struth. Automated reasoning in higher-order regular algebra. In *RAMiCS 2012*, volume 7560 of *LNCS*, pages 66–81. Springer, 2012.

[2] C. Ballarin. The Isabelle/HOL algebra library. https://isabelle.in.tum.de/dist/library/HOL/HOL-Algebra/index.html.

[3] G. Gierz, K. H. Hofmann, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer, 1980.

[4] V. B. F. Gomes and G. Struth. Residuated lattices. *Archive of Formal Proofs*, 2015.

[5] P. T. Johnstone. *Stone Spaces*. Cambridge University Press, 1982.

[6] S. Koppelberg. *Handbook of Boolean Algebras*. North-Holland, 1989.

[7] O. Kuncar and A. Popescu. From types to sets by local type definitions in higher-order logic. In *ITP 2016*, volume 9807 of *LNCS*, pages 200–218. Springer, 2016.

[8] V. Preoteasa. Algebra of monotonic boolean transformers. *Archive of Formal Proofs*, 2011.

[9] V. Preoteasa. Lattice properties. *Archive of Formal Proofs*, 2011.

[10] K. I. Rosenthal. *Quantales and their Applications*. Longman Scientific & Technical, 1990.

[11] M. Wenzel. Session HOL-Lattice. https://isabelle.in.tum.de/dist/library/HOL/HOL-Lattice/index.html.