Stone Algebras

Walter Guttmann

August 16, 2018

Abstract

A range of algebras between lattices and Boolean algebras generalise the notion of a complement. We develop a hierarchy of these pseudo-complemented algebras that includes Stone algebras. Independently of this theory we study filters based on partial orders. Both theories are combined to prove Chen and Grätzer’s construction theorem for Stone algebras. The latter involves extensive reasoning about algebraic structures in addition to reasoning in algebraic structures.

Contents

1 Synopsis and Motivation 2

2 Lattice Basics 3

3 Pseudocomplemented Algebras 12
   3.1 P-Algebras . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
      3.1.1 Pseudocomplemented Lattices . . . . . . . . . . . . . 13
      3.1.2 Pseudocomplemented Distributive Lattices . . . . . 21
   3.2 Stone Algebras . . . . . . . . . . . . . . . . . . . . . . . . 23
   3.3 Heyting Algebras . . . . . . . . . . . . . . . . . . . . . . . 27
      3.3.1 Heyting Semilattices . . . . . . . . . . . . . . . . . 27
      3.3.2 Heyting Lattices . . . . . . . . . . . . . . . . . . . . 31
      3.3.3 Heyting Algebras . . . . . . . . . . . . . . . . . . . . 33
      3.3.4 Brouwer Algebras . . . . . . . . . . . . . . . . . . . 34
   3.4 Boolean Algebras . . . . . . . . . . . . . . . . . . . . . . . 35

4 Filters 38
   4.1 Orders . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
   4.2 Lattices . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
   4.3 Distributive Lattices . . . . . . . . . . . . . . . . . . . . . 50
## 1 Synopsis and Motivation

This document describes the following four theory files:

- **Lattice Basics** is a small theory with basic definitions and facts extending Isabelle/HOL’s lattice theory. It is used by the following theories.

- **Pseudocomplemented Algebras** contains a hierarchy of algebraic structures between lattices and Boolean algebras. Many results of Boolean algebras can be derived from weaker axioms and are useful for more general models. In this theory we develop a number of algebraic structures with such weaker axioms. The theory has four parts. We first extend lattices and distributive lattices with a pseudocomplement operation to obtain (distributive) p-algebras. An additional axiom of the pseudocomplement operation yields Stone algebras. The third part studies a relative pseudocomplement operation which results in Heyting algebras and Brouwer algebras. We finally show that Boolean algebras instantiate all of the above structures.

- **Filters** contains an order-/lattice-theoretic development of filters. We prove the ultrafilter lemma in a weak setting, several results about the lattice structure of filters and a few further results from the literature. Our selection is due to the requirements of the following theory.

- **Construction of Stone Algebras** contains the representation of Stone algebras as triples and the corresponding isomorphisms [7, 21]. It is also a case study of reasoning about algebraic structures. Every Stone algebra is isomorphic to a triple comprising a Boolean algebra, a distributive lattice with a greatest element, and a bounded lattice homomorphism from the Boolean algebra to filters of the distributive
lattice. We carry out the involved constructions and explicitly state the functions defining the isomorphisms. A function lifting is used to work around the need for dependent types. We also construct an embedding of Stone algebras to inherit theorems using a technique of universal algebra.

Algebras with pseudocomplements in general, and Stone algebras in particular, appear widely in mathematical literature; for example, see [4, 5, 6, 17]. We apply Stone algebras to verify Prim’s minimum spanning tree algorithm in Isabelle/HOL in [20].

There are at least two Isabelle/HOL theories related to filters. The theory \texttt{HOL/Algebra/Ideal.thy} defines ring-theoretic ideals in locales with a carrier set. In the theory \texttt{HOL/Filter.thy} a filter is defined as a set of sets. Filters based on orders and lattices abstract from the inner set structure; this approach is used in many texts such as [4, 5, 6, 9, 17]. Moreover, it is required for the construction theorem of Stone algebras, whence our theory implements filters this way.

Besides proving the results involved in the construction of Stone algebras, we study how to reason about algebraic structures defined as Isabelle/HOL classes without carrier sets. The Isabelle/HOL theories \texttt{HOL/Algebra/*thys} use locales with a carrier set, which facilitates reasoning about algebraic structures but requires assumptions involving the carrier set in many places. Extensive libraries of algebraic structures based on classes without carrier sets have been developed and continue to be developed [1, 2, 3, 10, 11, 13, 14, 15, 16, 19, 22, 24, 25, 26]. It is unlikely that these libraries will be converted to carrier-based theories and that carrier-free and carrier-based implementations will be consistently maintained and evolved; certainly this has not happened so far and initial experiments suggest potential drawbacks for proof automation [12]. An improvement of the situation seems to require some form of automation or system support that makes the difference irrelevant.

In the present development, we use classes without carrier sets to reason about algebraic structures. To instantiate results derived in such classes, the algebras must be represented as Isabelle/HOL types. This is possible to a certain extent, but causes a problem if the definition of the underlying set depends on parameters introduced in a locale; this would require dependent types. For the construction theorem of Stone algebras we work around this restriction by a function lifting. If the parameters are known, the functions can be specialised to obtain a simple (non-dependent) type that can instantiate classes. For the construction theorem this specialisation can be done using an embedding. The extent to which this approach can be generalised to other settings remains to be investigated.
2 Lattice Basics

This theory provides notations, basic definitions and facts of lattice-related structures used throughout the subsequent development.

theory Lattice-Basics

imports Main

begin

The following results extend basic Isabelle/HOL facts.

lemma if-distrib-2:
  \( f (\text{if } c \text{ then } x \text{ else } y) \ (\text{if } c \text{ then } z \text{ else } w) = (\text{if } c \text{ then } f x z \text{ else } f y w) \)
  by simp

lemma left-invertible-inj:
  \( (\forall x. \ g \ (f x) = x) \implies \text{inj } f \)
  by (metis injI)

lemma invertible-bij:
  assumes \( \forall x. \ g \ (f x) = x \)
  and \( \forall y. \ f \ (g y) = y \)
  shows bij f
  by (metis assms bijI)

lemma finite-ne-subset-induct [consumes 3, case-names singleton insert]:
  assumes finite F
  and F \( \neq \) \{\}\n  and F \( \subseteq \) S
  and singleton: \( \forall x. \ P \ \{x\} \)
  and insert: \( \forall x F. \ \text{finite } F \implies F \neq \{\} \implies F \subseteq S \implies x \in S \implies x \notin F \)
  \( \implies P F \implies P \ (\text{insert } x \ F) \)
  shows P F
  using assms(1-3)
  apply (induct rule: finite-ne-induct)
  apply (simp add: singleton)
  by (simp add: insert)

lemma finite-set-of-finite-funs-pred:
  assumes finite \{ x::'a . True \}
  and finite \{ y::'b . P y \}
  shows finite \{ f . (\forall x::'a . P (f x)) \}
  using assms finite-set-of-finite-funs by force

We use the following notations for the join, meet and complement operations. Changing the precedence of the unary complement allows us to write terms like \(-\neg x\) instead of \(-(-x)\).

context sup
begin
notation \texttt{sup} (infixl \mathbin{\sqcup} 65)

\begin{definition}
\texttt{additive} :: \langle 'a \rightarrow 'a \rangle \Rightarrow \texttt{bool}
\begin{prooftree}
\text{where} \quad \texttt{additive} \, f \equiv \forall x \, y . \, f \, (x \sqcup y) = f \, x \sqcup f \, y
\end{prooftree}
\end{definition}
end

context \texttt{inf}
begin

notation \texttt{inf} (infixl \mathbin{\sqcap} 67)
end

context \texttt{uminus}
begin

\text{no-notation} \texttt{uminus} (\mathbin{-} [81] 80)

notation \texttt{uminus} (\mathbin{-} [80] 80)

end

We use the following definition of monotonicity for operations defined in classes. The standard \texttt{mono} places a sort constraint on the target type. We also give basic properties of Galois connections and lift orders to functions.

context \texttt{ord}
begin

\begin{definition}
\texttt{isotone} :: \langle 'a \rightarrow 'a \rangle \Rightarrow \texttt{bool}
\begin{prooftree}
\text{where} \quad \texttt{isotone} \, f \equiv \forall x \, y . \, x \leq y \rightarrow f \, x \leq f \, y
\end{prooftree}
\end{definition}

\begin{definition}
\texttt{galois} :: \langle 'a \Rightarrow 'a \rangle \Rightarrow \langle 'a \Rightarrow 'a \rangle \Rightarrow \texttt{bool}
\begin{prooftree}
\text{where} \quad \texttt{galois} \, l \, u \equiv \forall x \, y . \, l \, x \leq y \leftrightarrow x \leq u \, y
\end{prooftree}
\end{definition}

\begin{definition}
\texttt{lifted-less-eq} :: \langle 'a \Rightarrow 'a \rangle \Rightarrow \langle 'a \Rightarrow 'a \rangle \Rightarrow \texttt{bool} \, ((\mathbin{-} \leq \mathbin{-}) \, [51, 51] 50)
\begin{prooftree}
\text{where} \quad f \leq g \equiv \forall x . \, f \, x \leq g \, x
\end{prooftree}
\end{definition}
end

context \texttt{order}
begin

\text{lemma} \, \texttt{order-lesseq-imp}:
\langle \forall z . \, x \leq z \rightarrow y \leq z \rangle \leftrightarrow y \leq x
\begin{prooftree}
\text{using} \, \texttt{order-trans} \, \text{by} \, \texttt{blast}
\end{prooftree}

\text{lemma} \, \texttt{galois-char}:
\[
galois l u \iff (\forall x . x \leq u (l x)) \land (\forall x . l (u x) \leq x) \land \text{isotone } l \land \text{isotone } u
\]
apply (rule iffI)
apply (metis (full-types) galois-def isotone-def order-refl order-trans)
using galois-def isotone-def order-trans by blast

lemma galois-closure:
\[
galois l u \implies l x = l (u (l x)) \land u x = u (l (u x))
\]
by (simp add: galois-char isotone-def antisym)

lemma lifted-reflexive:
\[
f = g \implies f \leq g
\]
by (simp add: lifted-less-eq-def)

lemma lifted-transitive:
\[
f \leq g \implies g \leq h \implies f \leq h
\]
using lifted-less-eq-def order-trans by blast

lemma lifted-antisymmetric:
\[
f \leq g \implies g \leq f \implies f = g
\]
by (metis (full-types) antisym ext lifted-less-eq-def)

If the image of a finite non-empty set under \( f \) is a totally ordered, there is an element that minimises the value of \( f \).

lemma finite-set-minimal:
assumes finite \( s \)
and \( s \neq \{\} \)
and \( \forall x \in s . \forall y \in s . f x \leq f y \lor f y \leq f x \)
shows \( \exists m \in s . \forall z \in s . f m \leq f z \)
apply (rule finite-ne-subset-induct[where \( S = s \)])
apply (rule assms(1))
apply (rule assms(2))
apply simp
apply simp
by (metis assms(3) insert-iff order-trans set-mp)
end

The following are basic facts in semilattices.

context semilattice-sup
begin

lemma sup-left-isotone:
\[
x \leq y \implies x \sqcup z \leq y \sqcup z
\]
using sup.mono by blast

lemma sup-right-isotone:
\[
x \leq y \implies z \sqcup x \leq z \sqcup y
\]
using sup.mono by blast
lemma sup-left-divisibility:
x ≤ y ←→ (∃ z. x ⊔ z = y)
using sup.absorb2 sup.cobounded1 by blast

lemma sup-right-divisibility:
x ≤ y ←→ (∃ z. z ⊔ x = y)
by (metis sup.cobounded2 sup.orderE)

lemma sup-same-context:
x ≤ y ⊔ z =⇒ y ≤ x ⊔ z =⇒ x ⊔ z = y ⊔ z
by (simp add: le_iff_sup sup-left-commute)

lemma sup-relative-same-increasing:
x ≤ y =⇒ x ⊔ z = x ⊔ w =⇒ y ⊔ z = y ⊔ w
using sup.assoc sup-right-divisibility by auto

end

Every bounded semilattice is a commutative monoid. Finite sums defined in commutative monoids are available via the following sublocale.

context bounded-semilattice-sup-bot
begin
sublocale sup-monoid: comm-monoid-add where plus = sup and zero = bot
apply unfold-locales
apply (simp add: sup-assoc)
by simp

end

context semilattice-inf
begin
lemma inf-same-context:
x ≤ y ⊓ z =⇒ y ≤ x ⊓ z =⇒ x ⊓ z = y ⊓ z
using antisym by auto

end

The following class requires only the existence of upper bounds, which is a property common to bounded semilattices and (not necessarily bounded) lattices. We use it in our development of filters.

class directed-semilattice-inf = semilattice-inf +
assumes ub: ∃ z. x ≤ z ∧ y ≤ z

We extend the \textit{inf} sublocale, which dualises the order in semilattices, to bounded semilattices.

context bounded-semilattice-inf-top
begin

subclass directed-semilattice-inf
  apply unfold-locales
  using top-greatest by blast

sublocale inf: bounded-semilattice-sup-bot where sup = inf and less-eq =
greater-eq and less = greater and bot = top
  by unfold-locales (simp-all add: less-le-not-le)
end

context lattice
begin

subclass directed-semilattice-inf
  apply unfold-locales
  using sup-ge1 sup-ge2 by blast

definition dual-additive :: ('a ⇒ 'a) ⇒ bool
  where dual-additive f ≡ ∀ x y. f x ⊔ y = f x ⊓ f y
end

Not every bounded lattice has complements, but two elements might still
be complements of each other as captured in the following definition. In this
situation we can apply, for example, the shunting property shown below. We
introduce most definitions using the abbreviation command.

context bounded-lattice
begin

abbreviation complement x y ≡ x ⊔ y = top ∧ x ⊓ y = bot

lemma complement-symmetric:
  complement x y =⇒ complement y x
  by (simp add: inf.commute sup.commute)

definition conjugate :: ('a ⇒ 'a) ⇒ ('a ⇒ 'a) ⇒ bool
  where conjugate f g ≡ ∀ x y. f x ⊓ y = bot =⇒ x ⊓ g y = bot
end

class dense-lattice = bounded-lattice +
  assumes bot-meet-irreducible: x ⊓ y = bot =⇒ x = bot ∨ y = bot

context distrib-lattice
begin

lemma relative-equality:
Distributive lattices with a greatest element are widely used in the construction theorem for Stone algebras.

class distrib-lattice-bot = bounded-lattice-bot + distrib-lattice

class distrib-lattice-top = bounded-lattice-top + distrib-lattice

class bounded-distrib-lattice = bounded-lattice + distrib-lattice

begin

subclass distrib-lattice-bot ..

subclass distrib-lattice-top ..

lemma complement-shunting:
  assumes complement z w
  shows z ∩ x ≤ y ⇐⇒ x ≤ w ⊔ y

proof
  assume 1: z ∩ x ≤ y
  have x = (z ⊔ w) ∩ x
    by (simp add: assms)
  also have ... ≤ y ⊔ (w ∩ x)
    using 1 sup commute sup.left-commute inf-sup-distrib2 sup-right-divisibility
  by fastforce
  also have ... ≤ w ⊔ y
    by (simp add: inf.coboundedI1)
  finally show x ≤ w ⊔ y
  .

next
  assume x ≤ w ⊔ y
  hence z ∩ x ≤ z ∩ (w ⊔ y)
    using inf.sup-right-isotone by auto
  also have ... = z ∩ y
    by (simp add: assms inf-sup-distrib1)
  also have ... ≤ y
    by simp
  finally show z ∩ x ≤ y
  .

qed

end

We next consider lattices with a linear order structure. In such lattices, join and meet are selective operations, which give the maximum and the minimum of two elements, respectively. Moreover, the lattice is automati-
cally distributive.

class bounded-linorder = linorder + order-bot + order-top

class linear-lattice = lattice + linorder

begin

lemma max-sup:
\[ \max x y = x \sqcup y \]
by (metis max.boundedI max.coboundedI max.cobounded2 sup-unique)

lemma min-inf:
\[ \min x y = x \sqcap y \]
by (simp add: inf.absorb1 inf.absorb2 min-def)

lemma sup-inf-selective:
\[ (x \sqcup y = x \land x \sqcap y = y) \lor (x \sqcup y = y \land x \sqcap y = x) \]
by (meson inf.absorb1 inf.absorb2 le-cases sup.absorb1 sup.absorb2)

lemma sup-selective:
\[ x \sqcup y = x \lor x \sqcup y = y \]
using sup-inf-selective by blast

lemma inf-selective:
\[ x \sqcap y = x \lor x \sqcap y = y \]
using sup-inf-selective by blast

subclass distrib-lattice
apply unfold-locales
by (metis inf-selective antisym distrib-sup-le inf.commute inf.le2)

lemma sup-less-eq:
\[ x \leq y \sqcup z \iff x \leq y \lor x \leq z \]
by (metis le-supI1 le-supI2 sup-selective)

lemma inf-less-eq:
\[ x \sqcap y \leq z \iff x \leq z \lor y \leq z \]
by (metis inf.coboundedI1 inf.coboundedI2 inf-selective)

lemma sup-inf-sup:
\[ x \sqcup y = (x \sqcup y) \sqcup (x \sqcap y) \]
by (metis sup-commute sup-inf-absorb sup-left-commute)

end

The following class derives additional properties if the linear order of the
lattice has a least and a greatest element.

class linear-bounded-lattice = bounded-lattice + linorder

begin
subclass linear-lattice ..

subclass bounded-linorder ..

subclass bounded-distrib-lattice ..

lemma sup-dense:
  \(x \neq \text{top} \implies y \neq \text{top} \implies x \sqcup y \neq \text{top}\)
  by (metis sup-selective)

lemma inf-dense:
  \(x \neq \text{bot} \implies y \neq \text{bot} \implies x \sqcap y \neq \text{bot}\)
  by (metis inf-selective)

lemma sup-not-bot:
  \(x \neq \text{bot} \implies x \sqcup y \neq \text{bot}\)
  by simp

lemma inf-not-top:
  \(x \neq \text{top} \implies x \sqcap y \neq \text{top}\)
  by simp

subclass dense-lattice
  apply unfold-locales
  using inf-dense by blast
end

Every bounded linear order can be expanded to a bounded lattice. Join and meet are maximum and minimum, respectively.

class linorder-lattice-expansion = bounded-linorder + sup + inf +
  assumes sup-def [simp]: \(x \sqcup y = \max x y\)
  assumes inf-def [simp]: \(x \sqcap y = \min x y\)
begin
subclass linear-bounded-lattice
  apply unfold-locales
  by auto
end

Some results, such as the existence of certain filters, require that the algebras are not trivial. This is not an assumption of the order and lattice classes that come with Isabelle/HOL; for example, \(\text{bot} = \text{top}\) may hold in bounded lattices.

class non-trivial =
  assumes consistent: \(\exists x y . x \neq y\)

class non-trivial-order = non-trivial + order
class non-trivial-order-bot = non-trivial-order + order-bot

class non-trivial-bounded-order = non-trivial-order-bot + order-top

begin

lemma bot-not-top:
  bot ≠ top
proof -
  from consistent obtain x y :: 'a where x ≠ y
    by auto
  thus ?thesis
    by (metis bot-less top.extremum-strict)
qed

end

end

3 Pseudocomplemented Algebras

This theory expands lattices with a pseudocomplement operation. In particular, we consider the following algebraic structures:

- pseudocomplemented lattices (p-algebras)
- pseudocomplemented distributive lattices (distributive p-algebras)
- Stone algebras
- Heyting semilattices
- Heyting lattices
- Heyting algebras
- Heyting-Stone algebras
- Brouwer algebras
- Boolean algebras

Most of these structures and many results in this theory are discussed in [4, 5, 6, 8, 17, 23].

theory P-Algebras

imports Lattice-Basics

begin
3.1 P-Algebras

In this section we add a pseudocomplement operation to lattices and to distributive lattices.

3.1.1 Pseudocomplemented Lattices

The pseudocomplement of an element \( y \) is the greatest element whose meet with \( y \) is the least element of the lattice.

```haskell
class p-algebra = bounded-lattice + uminus +
  assumes pseudo-complement: x \sqcap y = bot \iff x \leq -y
begin
  Regular elements and dense elements are frequently used in pseudocomplemented algebras.

  abbreviation regular x \equiv x = -x
  abbreviation dense x \equiv -x = bot
  abbreviation complemented x \equiv \exists y . x \sqcap y = bot \land x \sqcup y = top
  abbreviation in-p-image x \equiv \exists y . x = -y
  abbreviation selection s x \equiv s = -s \sqcap x

  abbreviation dense-elements \equiv \{ x . dense x \}
  abbreviation regular-elements \equiv \{ x . in-p-image x \}

lemma p-bot [simp]:
  \(-bot = top\)
  using inf-top.left-neutral pseudo-complement top-unique by blast

lemma p-top [simp]:
  \(-top = bot\)
  by (metis eq-refl inf-top.comm-neutral pseudo-complement)

  The pseudocomplement satisfies the following half of the requirements of a complement.

lemma inf-p [simp]:
  \(x \sqcap -x = bot\)
  using inf.commute pseudo-complement by fastforce

lemma p-inf [simp]:
  \(-x \sqcap x = bot\)
  by (simp add: inf-commute)

lemma pp-inf-p:
  \(-x \sqcap -x = bot\)
  by simp

  The double complement is a closure operation.

lemma pp-increasing:
```
using \( \inf-p \) pseudo-complement by blast

\textbf{lemma} ppp [simp]:
\[
-x = -x
\]
by (metis antisym inf.commute order-trans pseudo-complement pp-increasing)

\textbf{lemma} pp-idempotent:
\[
-x = -x
\]
by simp

\textbf{lemma} regular-in-p-image-iff:
regular \( x \) \( \iff \) in-p-image \( x \)
by auto

\textbf{lemma} pseudo-complement-pp:
\[
x \land y = \bot \iff -x \leq -y
\]
by (metis inf-commute pseudo-complement pp)

\textbf{lemma} p-antitone:
\[
x \leq y \implies -y \leq -x
\]
by (metis inf-commute order-trans pseudo-complement pp-increasing)

\textbf{lemma} p-antitone-sup:
\[
-(x \lor y) \leq -x
\]
by (simp add: p-antitone)

\textbf{lemma} p-antitone-inf:
\[
-x \leq -(x \land y)
\]
by (simp add: p-antitone)

\textbf{lemma} p-antitone-iff:
\[
x \leq -y \iff y \leq -x
\]
using order-lesseq-imp p-antitone pp-increasing by blast

\textbf{lemma} pp-isotone:
\[
x \leq y \implies --x \leq --y
\]
by (simp add: p-antitone)

\textbf{lemma} pp-isotone-sup:
\[
--x \leq --(x \lor y)
\]
by (simp add: p-antitone)

\textbf{lemma} pp-isotone-inf:
\[
--(x \land y) \leq --x
\]
by (simp add: p-antitone)

One of De Morgan’s laws holds in pseudocomplemented lattices.
\[-(x \sqcup y) = -x \sqcap -y\]

**apply** (rule antisym)

**apply** (simp add: p-antitone)

**using** inf-le1 inf-le2 le-sup-iff p-antitone-iff by blast

**lemma** p-supdist-inf:
\[-x \sqcup -y \leq -(x \sqcap y)\]

**by** (simp add: p-antitone)

**lemma** pp-dist-pp-sup [simp]:
\[-(-x \sqcup -y) = -(-(x \sqcap y))\]

**by** simp

**lemma** p-sup-p [simp]:
\[-(x \sqcap -x) = \bot\]

**by** simp

**lemma** pp-sup-p [simp]:
\[--(x \sqcap -x) = \top\]

**by** simp

**lemma** dense-pp:
\[\text{dense } x \iff -x = \top\]

**by** (metis p-bot p-top ppp)

**lemma** dense-sup-p:
\[\text{dense } (x \sqcup -x)\]

**by** simp

**lemma** regular-char:
\[\text{regular } x \iff \exists y . x = -y\]

**by** auto

**lemma** pp-inf-bot-iff:
\[x \sqcap y = \bot \iff -x \sqcap y = \bot\]

**by** (simp add: pseudo-complement-pp)

Weak forms of the shunting property hold. Most require a pseudocomplemented element on the right-hand side.

**lemma** p-shunting-swap:
\[x \sqcap y \leq -z \iff x \sqcap z \leq -y\]

**by** (metis inf-assoc inf-commute pseudo-complement)

**lemma** pp-inf-below-iff:
\[x \sqcap y \leq -z \iff -x \sqcap y \leq -z\]

**by** (simp add: inf-commute p-shunting-swap)

**lemma** p-inf-pp [simp]:
\[-(x \sqcap -y) = -(x \sqcap y)\]
apply (rule antisym)
apply (simp add: inf.coboundedI2 p-antitone pp-increasing)
using inf-commute p-antitone-iff pp-inf-below-iff by auto

lemma p-inf-pp-pp [simp]:
  \(-(-x \cap -y) = -(x \cap y)\)
by (simp add: inf-commute)

lemma regular-closed-inf:
  regular x \implies regular y \implies regular (x \cap y)
by (metis p-dist-sup ppp)

lemma regular-closed-p:
  regular (-x)
by simp

lemma regular-closed-pp:
  regular (-x)
by simp

lemma regular-closed-bot:
  regular bot
by simp

lemma regular-closed-top:
  regular top
by simp

lemma pp-dist-inf [simp]:
  \(-(-x \cap y) = -(-x \cap -y)\)
by (metis p-dist-sup p-inf-pp-pp ppp)

lemma inf-import-p [simp]:
  x \cap -(x \cap y) = x \cap -y
apply (rule antisym)
using p-shunting-swap apply fastforce
using inf.sup-right-isotone p-antitone by auto

Pseudocomplements are unique.

lemma p-unique:
  (\forall x . x \cap y = bot \longleftrightarrow x \leq z) \implies z = -y
using inf.eq-iff pseudo-complement by auto

lemma maddux-3-5:
  x \sqcup x = x \sqcup -(y \sqcup -y)
by simp

lemma shunting-1-pp:
  x \leq -y \longleftrightarrow x \cap -y = bot
by \((simp \ add: \ pseudo-complement)\)

**lemma** \(pp\text{-}pp\text{-}inf\text{-}bot\text{-}iff\):
\[ x \cap y = bot \iff --x \cap --y = bot \]
by \((simp \ add: \ pseudo-complement\text{-}pp)\)

**lemma** \(inf\text{-}pp\text{-}semi\text{-}commute\):
\[ x \cap --y \leq --(x \cap y) \]
using \(inf\text{-}eq\text{-}refl \ p\text{-}antitone\text{-}iff \ pp\text{-}inf\text{-}pp \ by \ presburger\)

**lemma** \(inf\text{-}pp\text{-}commute\):
\[ --(--x \cap y) = --x \cap --y \]
by \(simp\)

**lemma** \(sup\text{-}pp\text{-}semi\text{-}commute\):
\[ x \sqcup --y \leq --(x \sqcup y) \]
by \((simp \ add: \ p\text{-}antitone\text{-}iff)\)

**lemma** \(regular\text{-}sup\):
regular \(z\) \(\Rightarrow\) \((x \leq z \land y \leq z \iff --(x \sqcup y) \leq z)\)
apply \((rule \ iffI)\)
apply \((metis \ le\text{-}supI \ pp\text{-}isotone)\)
using \(dual\text{-}order\text{-}trans \ sup\text{-}ge2 \ pp\text{-}increasing \ pp\text{-}isotone\text{-}sup \ by \ blast\)

**lemma** \(dense\text{-}closed\text{-}inf\):
\[ dense \ x \Rightarrow dense \ y \Rightarrow dense \ (x \cap y) \]
by \((simp \ add: \ dense\text{-}pp)\)

**lemma** \(dense\text{-}closed\text{-}sup\):
\[ dense \ x \Rightarrow dense \ y \Rightarrow dense \ (x \sqcup y) \]
by \(simp\)

**lemma** \(dense\text{-}closed\text{-}pp\):
\[ dense \ x \Rightarrow dense \ (--x) \]
by \(simp\)

**lemma** \(dense\text{-}closed\text{-}top\):
\[ dense \ top \]
by \(simp\)

**lemma** \(dense\text{-}up\text{-}closed\):
\[ dense \ x \Rightarrow x \leq y \Rightarrow dense \ y \]
using \(dense\text{-}pp \ top\text{-}le \ pp\text{-}isotone \ by \ auto\)

**lemma** \(regular\text{-}dense\text{-}top\):
\[ regular \ x \Rightarrow dense \ x \Rightarrow x = top \]
using \(p\text{-}bot \ by \ blast\)

**lemma** \(selection\text{-}char\):

\[ 17 \]
Conjugates are usually studied for Boolean algebras, however, some of their properties generalise to pseudocomplemented algebras.

lemma conjugate-unique-p:
assumes conjugate \( f \) \( g \)
and conjugate \( f \) \( h \)
shows \( \uminus \circ g = \uminus \circ h \)

proof –
have \( \forall x \ y. \ x \cap g y = \bot \longleftrightarrow x \cap h y = \bot \)
  using assms conjugate-def inf-commute by simp
hence \( \forall x \ y. \ x \leq -(g y) \longleftrightarrow x \leq -(h y) \)
  using inf-commute pseudo-complement by simp
hence \( \forall y. \ -(g y) = -(h y) \)
  using eq-iff by blast
thus \( \text{thesis} \)
by auto
qed

lemma conjugate-symmetric:
conjugate \( f \) \( g \) \( \Rightarrow \) conjugate \( g \) \( f \)
by (simp add: conjugate-def inf-commute)

lemma additive-isotone:
additive \( f \) \( \Rightarrow \) isotone \( f \)
by (metis additive-def isotone-def le-iff-sup)

lemma dual-additive-antitone:
assumes dual-additive \( f \)
shows isotone \( (\uminus \circ f) \)
proof –
have \( \forall x \ y. \ f (x \sqcup y) \leq f x \)
using assms dual-additive-def by simp

hence ∀ x y . x ≤ y → f y ≤ f x
  by (metis sup-absorb2)

hence ∀ x y . x ≤ y → −(f x) ≤ −(f y)
  by (simp add: p-antitone)

thus ?thesis
  by (simp add: isotone-def)

qed

lemma conjugate-dual-additive:
  assumes conjugate f g
  shows dual-additive (uminus ◦ f)
proof –
  have 1: ∀ x y z . −z ≤ −(f (x ⊔ y)) ↔ −z ≤ −(f x) ∧ −z ≤ −(f y)
    proof (intro allI)
      fix x y z
      have (−z ≤ −(f (x ⊔ y))) = (f (x ⊔ y) ∩ −z = bot)
        by (simp add: p-antitone-iff pseudo-complement)
      also have ... = (x ⊔ y) ∩ g(−z) = bot
        using assms conjugate-def by auto
      also have ... = (x ⊔ y ≤ −(g(−z)))
        by (simp add: pseudo-complement)
      also have ... = (x ≤ −(g(−z))) ∧ y ≤ −(g(−z))
        by (simp add: le-sup-iff)
      also have ... = (f x ∩ −z = bot ∧ f y ∩ −z = bot)
        by (simp add: pseudo-complement)
      also have ... = (f x ∩ −z = bot ∧ y ∩ −z = bot)
        using assms conjugate-def by auto
      also have ... = (−z ≤ −(f x) ∧ −z ≤ −(f y))
        by (simp add: p-antitone-iff pseudo-complement)
      finally show −z ≤ −(f (x ⊔ y)) ↔ −z ≤ −(f x) ∧ −z ≤ −(f y)
        by simp
    qed

  have ∀ x y . −(f (x ⊔ y)) = −(f x) ∩ −(f y)
    proof (intro allI)
      fix x y
      have −(f x) ∩ −(f y) = −−(−(f x) ∩ −(f y))
        by simp
      hence −(f x) ∩ −(f y) ≤ −(f (x ⊔ y))
        using 1 by (metis inf-le1 inf-le2)
      thus −(f (x ⊔ y)) = −(f x) ∩ −(f y)
        using 1 antisym by fastforce
    qed

thus ?thesis
  using dual-additive-def by simp

qed

lemma conjugate-isotone-pp:
  conjugate f g ⟷ isotone (uminus ◦ uminus ◦ f)
by (simp add: comp-assoc conjugate-dual-additive dual-additive-antitone)

lemma conjugate-char-1-pp:
conjugate $f$ $g$ $\iff$ $(\forall x y . f(x \cap -(g y)) \leq --f x \cap -y \land g(y \cap -(f x)) \leq --g y \cap -x)$

proof
assumes 1: conjugate $f$ $g$
shows $(\forall x y . f(x \cap -(g y)) \leq --f x \cap -y \land g(y \cap -(f x)) \leq --g y \cap -x)$
proof (intro allI)
fix $x$ $y$
have 2: $f(x \cap -(g y)) \leq -y$
using 1 (simp add: conjugate-def pseudo-complement)
have $f(x \cap -(g y)) \leq -f(x \cap -(g y))$
by (simp add: pp-increasing)
also have ... $\leq --f x$
using 1 conjugate-isotone-pp isotone-def by simp
finally have 3: $f(x \cap -(g y)) \leq --f x \cap -y$
using 2 by simp
have 4: isotone $(uminus \circ uminus \circ g)\circ l$ using 1 conjugate-isotone-pp conjugate-symmetric by auto
have 5: $g(y \cap -(f x)) \leq -x$
using 1 by (metis conjugate-def inf.order.2 inf-commute)

have $g(y \cap -(f x)) \leq --g(y \cap -(f x))$
by (simp add: pp-increasing)
also have ... $\leq --g y$
using 4 isotone-def by auto
finally have 6: $f(x \cap -(g y)) \leq --f x \cap -y \land g(y \cap -(f x)) \leq --g y \cap -x$
using 3 by simp
thus $(\forall x y . f(x \cap -(g y)) \leq --f x \cap -y \land g(y \cap -(f x)) \leq --g y \cap -x)$
using 3 by simp
qed

next
assumes 7: $(\forall x y . f(x \cap -(g y)) \leq --f x \cap -y \land g(y \cap -(f x)) \leq --g y \cap -x)$
have $(\forall x y . f(x \cap -(g y)) \leq --f x \cap -y \land g(y \cap -(f x)) \leq --g y \cap -x)$
by (metis inf.order.2 sup.order.2 inf-sup iff sup-sup iff inf-commute pseudo-complement)
thus conjugate $f$ $g$
using 7 conjugate-def by auto
qed

lemma conjugate-char-1-isotone:
conjugate $f$ $g \Rightarrow$ isotone $f \Rightarrow$ isotone $g \Rightarrow$ $f(x \cap -(g y)) \leq f(x \cap -(f x)) \leq g(y \cap -(f x)) \leq g(y \cap -(g y))$
by (simp add: conjugate-char-1-pp ord.isotone-def)

lemma dense-lattice-char-1:

20
\[(\forall x y . x \sqcap y = \bot \rightarrow x = \bot \lor y = \bot) \iff (\forall x . x \neq \bot \rightarrow \text{dense } x)\]

by (metis inf-top.left-neutral p-bot p-inf pp-inf-bot-iff)

**lemma** dense-lattice-char-2:
\[(\forall x y . x \sqcap y = \bot \rightarrow x = \bot \lor y = \bot) \iff (\forall x . \text{regular } x \rightarrow x = \bot \lor x = \top)\]

by (metis dense-lattice-char-1 inf-top.left-neutral p-inf regular-closed-p regular-closed-top)

**lemma** restrict-below-Rep-eq:
\[x \sqcap -y \leq z \implies x \sqcap y = x \sqcap z \sqcap y\]

by (metis inf.absorb2 inf.commute inf.left-commute pp-increasing)

end

The following class gives equational axioms for the pseudocomplement operation.

**class** p-algebra-eq = bounded-lattice + uminus +

assumes p-bot-eq: \[-\bot = \top\]

and p-top-eq: \[-\top = \bot\]

and inf-import-p-eq: \[x \sqcap -(x \sqcap y) = x \sqcap y\]

begin

**lemma** inf-p-eq:
\[x \sqcap -x = \bot\]

by (metis inf-bot-right inf-import-p-eq inf-top-right p-top-eq)

subclass p-algebra

apply unfold-locales

apply (rule iffI)

apply (metis inf.orderI inf-import-p-eq inf-top-right-neutral p-bot-eq)

by (metis (full-types) inf.left-commute inf.orderE inf-bot-right inf-commute inf-p-eq)

end

3.1.2 Pseudocomplemented Distributive Lattices

We obtain further properties if we assume that the lattice operations are distributive.

**class** pd-algebra = p-algebra + bounded-distrib-lattice

begin

**lemma** p-inf-sup-below:
\[-x \sqcap (x \sqcup y) \leq y\]

by (simp add: inf-sup-distrib1)
lemma pp-inf-sup-p [simp]:
\[ -x \sqcap (x \sqcup -x) = x \]
using inf.absorb2 inf-sup-distrib1 pp-increasing by auto

lemma complement-p:
x \sqcap y = bot \implies x \sqcup y = top \implies -x = y
by (metis pseudo-complement inf.commute inf-top.left-neutral sup.absorb-iff1
sup.commute sup-bot.right-neutral sup-inf-distrib2 p-inf)

lemma complemented-regular:
complemented x \implies regular x
using complement-p inf.commute sup by fastforce

lemma regular-inf-dense:
\exists y z . regular y \land dense z \land x = y \sqcap z
by (metis pp-inf-sup-p dense-sup-p ppp)

lemma maddux-3-12 [simp]:
\[ (x \sqcup -y) \sqcap (x \sqcup y) = x \]
by (simp add: inf-sup-distrib2)

lemma maddux-3-13 [simp]:
\[ (x \sqcup y) \sqcap -x = y \sqcap -x \]
by (simp add: inf-sup-distrib2)

lemma maddux-3-20:
\[ ((v \sqcap w) \sqcup (-v \sqcap x)) \sqcap -((v \sqcap y) \sqcup (-v \sqcap z)) = (v \sqcap w \sqcap -y) \sqcup (-v \sqcap x \sqcap -z) \]
proof -
have v \sqcap w \sqcap -v \sqcap y \sqcap -v \sqcap z = v \sqcap w \sqcap -v \sqcap y
by (meson inf.cobounded1 inf-absorb1 le-infI1 p-antitone-iff)
also have ... = v \sqcap w \sqcap -y
using inf.sup-relative-same-increasing inf-import-p inf-le1 by blast
finally have 1: v \sqcap w \sqcap -v \sqcap z = v \sqcap w \sqcap -y
.
have -v \sqcap x \sqcap -v \sqcap y \sqcap -v \sqcap z = -v \sqcap x \sqcap -v \sqcap z
by (simp add: inf.absorb1 le-infI1 p-antitone-inf)
also have ... = -v \sqcap x \sqcap -z
by (simp add: inf.sup-relative-same-increasing inf-import-p inf-le1 by blast)
finally have 2: -v \sqcap x \sqcap -v \sqcap y \sqcap -v \sqcap z = -v \sqcap x \sqcap -z
.
have ((v \sqcap w) \sqcup (-v \sqcap x)) \sqcap -((v \sqcap y) \sqcup (-v \sqcap z)) = (v \sqcap w \sqcap -v \sqcap y \sqcap -v \sqcap z) \sqcup (-v \sqcap x \sqcap -v \sqcap y \sqcap -v \sqcap z)
by (simp add: inf-sup-distrib2)
also have ... = (v \sqcap w \sqcap -y) \sqcup (-v \sqcap x \sqcap -z)
using 1 2 by simp
finally show \( \)thesis
.
qed
lemma order-char-1:
\[ x \leq y \iff x \leq y \sqcup -x \]
by (metis inf.sup-left-isotone inf-sup-absorb le-supI1 maddux-3-12 sup-commute)

lemma order-char-2:
\[ x \leq y \iff x \sqcup -x \leq y \sqcup -x \]
using order-char-1 by auto

end

3.2 Stone Algebras

A Stone algebra is a distributive lattice with a pseudocomplement that satisfies the following equation. We thus obtain the other half of the requirements of a complement at least for the regular elements.

class stone-algebra = pd-algebra +
  assumes stone [simp]: \(-x \sqcup -x = \top\)
begin
  As a consequence, we obtain both De Morgan’s laws for all elements.

lemma p-dist-inf [simp]:
\(- (x \cap y) = -x \sqcup -y\)
proof (rule p-unique[THEN sym], rule allI, rule iffI)
  fix w
  assume \(w \cap (x \cap y) =\bot\)
  hence \(w \sqcap -x \cap y = \bot\)
    using inf-commute inf-left-commute pseudo-complement by auto
  hence 1: \(w \sqcap -x \leq -y\)
    by (simp add: pseudo-complement)
  have \(w = (w \cap -x) \sqcup (w \cap -x)\)
    using distrib-imp2 sup-inf-distrib1 by auto
  thus \(w \leq -x \sqcup -y\)
    using 1 by (metis inf-le2 sup.mono)
next
  fix w
  assume \(w \leq -x \sqcup -y\)
  thus \(w \cap (x \cap y) = \bot\)
    using order-trans p-supdist-inf pseudo-complement by blast
qed

lemma pp-dist-sup [simp]:
\(-(-x \sqcup y) = --x \sqcup --y\)
by simp

lemma regular-closed-sup:
The regular elements are precisely the ones having a complement.

**Lemma** `regular-complemented-iff`:

`regular x ⇔ complemented x`

**Proof**

by (metis `inf-p` `stone` `complemented-regular`)

**Lemma** `selection-closed-sup`:

`selection s x ⊔ selection t x = selection (s ⊔ t) x`

by (simp add: `inf-sup-distrib2`)

**Lemma** `huntington-3-pp` [simp]:

`¬(¬x ⊔ ¬y) ⊔¬(¬x ⊔ y) = ¬¬x`

by (metis `p-dist-inf` `p-inf` `sup-bot-left` `sup-inf-distrib1`)

**Lemma** `maddux-3-3` [simp]:

`¬(x ⊔ y) ⊔¬(x ⊔ ¬y) = ¬x`

by (simp add: `sup-commute` `sup-inf-distrib1`)

**Lemma** `maddux-3-11-pp`:

`(x ⊓¬ y) ⊔(x ⊓¬¬ y) = x`

by (metis `inf-sup-distrib1` `inf-top-right` `stone`)

**Lemma** `maddux-3-19-pp`:

`(¬x ⊓ y) ⊔(¬x ⊓ z) = (¬x ⊓ y) ⊔ (¬z ⊓ ¬y)`

**Proof**

have `(¬x ⊓ y) ⊔ (¬z ⊓ ¬y) = (¬x ⊓ z) ⊔ (y ⊓ ¬x) ⊔ (y ⊓ z) ⊔ (¬x ⊓ ¬y)`

by (simp add: `inf-commute` `inf-sup-distrib1` `sup-assoc`)

also have ... = `(¬x ⊓ z) ⊔ (y ⊓ ¬x) ⊔ (y ⊓ z) ⊔ (¬x ⊓ ¬y)`

by simp

also have ... = `(¬x ⊓ z) ⊔ (y ⊓ ¬x) ⊔ (y ⊓ z) ⊔ (¬x ⊓ ¬y)`

using `inf-sup-distrib1` `sup-assoc` `inf-commute` `inf-assoc` by presburger

also have ... = `(¬x ⊓ z) ⊔ (y ⊓ ¬x) ⊔ (y ⊓ z) ⊔ (¬x ⊓ ¬y)`

by simp

also have ... = `(¬x ⊓ z) ⊔ (y ⊓ ¬x) ⊔ (y ⊓ z) ⊔ (¬x ⊓ ¬y)`

by simp

finally show ?thesis

by (simp add: `inf-commute` `sup-commute`)

qed

**Lemma** `compl-inter-eq-pp`:

`¬¬x ⊓ y = ¬¬x ⊓ z =⇒ ¬x ⊓ y = ¬x ⊓ z =⇒ y = z`

by (metis `inf-commute` `inf-p` `inf-sup-distrib1` `inf-top-right-neutral` `p-bot` `p-dist-inf`)

**Lemma** `maddux-3-21-pp` [simp]:
\[ -x \cup (-x \cap y) = -x \cup y \]
by \((\text{simp add: sup.commute sup-inf-distrib1})\)

**lemma** shunting-2-pp:
\[ x \leq -y \iff -x \cup -y = \top \]
by \((\text{metis inf-top-left p-bot p-dist-inf pseudo-complement})\)

**lemma** shunting-p:
\[ x \cap y \leq -z \iff x \leq -z \cup -y \]
by \((\text{metis inf.assoc p-dist-inf p-shunting-swap pseudo-complement})\)

The following weak shunting property is interesting as it does not require the element \(z\) on the right-hand side to be regular.

**lemma** shunting-var-p:
\[ x \cap -y \leq z \iff x \leq -z \cup -y \]
proof
\begin{align*}
\text{assume} & \quad x \cap -y \leq z \\
\text{hence} & \quad z \cup -y = -y \cup (z \cup x \cap -y) \\
\text{by} & \quad (\text{simp add: sup.absorb1 sup.commute}) \\
\text{thus} & \quad x \leq z \cup -y \\
\text{by} & \quad (\text{metis inf-commute maddux-3-21 pp sup.commute sup.left-commute sup-left-divisibility})
\end{align*}
next
\begin{align*}
\text{assume} & \quad x \leq z \cup -y \\
\text{thus} & \quad x \cap -y \leq z \\
& \quad \text{by} \quad (\text{metis inf.mono maddux-3-12 sup.ge2})
\end{align*}
qed

**lemma** conjugate-char-2-pp:
\begin{align*}
\text{conjugate } f g & \iff f \bot \land g \bot = \bot \land (\forall x y . f x \cap y \leq --(f(x \cap --(g y)))) \land g y \cap x \leq --(g(y \cap --(f x))) \\
\text{proof} & \\
\text{assume} & \quad 1: \text{conjugate } f g \\
\text{hence} & \quad 2: \text{dual-additive } (\text{uminus} \circ g) \\
& \quad \text{using} \quad \text{conjugate-symmetric conjugate-dual-additive} \quad \text{by auto} \\
\text{show} & \quad f \bot = \bot \land g \bot = \bot \land (\forall x y . f x \cap y \leq --(f(x \cap --(g y)))) \land g y \cap x \leq --(g(y \cap --(f x))) \\
\text{proof} & \quad (\text{intro conjI}) \\
& \quad \text{show} f \bot = \bot \\
& \quad \text{using} f \text{ by} \quad (\text{metis conjugate-def inf-idem inf-bot-left})
\end{align*}
next
\begin{align*}
\text{show} & \quad g \bot = \bot \\
& \quad \text{using} f \text{ by} \quad (\text{metis conjugate-def inf-idem inf-bot-right})
\end{align*}
next
\begin{align*}
\text{show} & \quad \forall x y . f x \cap y \leq --(f(x \cap --(g y))) \land g y \cap x \leq --(g(y \cap --(f x))) \\
\text{proof} & \quad (\text{intro allI}) \\
& \quad \text{fix} x y \\
& \quad \text{have} \quad 3: \quad y \leq --(f(x \cap --(g y)))
\end{align*}
proof

next

lemma conjugate-char-2-pp-additive:

assumes conjugate \( f \) \( g \)

and additive \( f \)

and additive \( g \)

shows \( f \ x \cap y \leq f(x \cap -\neg(g \ y)) \land g \ y \cap x \leq g(y \cap -\neg(f \ x)) \)

proof

have \( f \ x \cap y = f((x \cap -\neg g \ y) \cup (x \cap -g \ y)) \cap y \)

by (simp add: sup.commute sup-inf-distrib1)

also have \( (...) = f((x \cap -\neg g \ y) \cup (x \cap -g \ y)) \cap y \)

using assms(2) additive-def inf-sup-distrib2 by auto

also have \( (...) = f((x \cap -\neg g \ y) \cap y) \)

by (metis assms(1) conjugate-def inf-le2 pseudo-complement sup-bot.right-neutral)

finally have \( 2: f \ x \cap y \leq f(x \cap -\neg g \ y) \)
by simp
have \( g \ y \cap x = g \ ((y \cap -f \ x) \cup (y \cap f \ x)) \cap x \)
by (simp add: sup.commute sup-inf-distrib1)
also have \( ... = (g \ (y \cap -f \ x) \cap x) \cup (g \ (y \cap f \ x) \cap x) \)
using assms(3) additive-def inf-sup-distrib2 by auto
also have \( ... = g \ (y \cap -f \ x) \cap x \)
by (metis assms(1) conjugate-def inf.cobounded2 pseudo-complement
sup-bot.right-neutral inf-commute)
finally have \( g \ y \cap x \leq g \ (y \cap -f \ x) \)
by simp
thus \(?thesis\)
using 2 by simp
qed

end

Every bounded linear order can be expanded to a Stone algebra. The
pseudocomplement takes \( \text{bot} \) to the \( \text{top} \) and every other element to \( \text{bot} \).

class linorder-stone-algebra-expansion = linorder-lattice-expansion + uminus +
assumes uminus-def [simp]: \(-x = (if \ x = \text{bot} \ then \ \text{top} \ else \ \text{bot})\)
begin

subclass stone-algebra
apply unfold-locales
using bot-unique min-def top-le by auto

The regular elements are the least and greatest elements. All elements
except the least element are dense.

lemma regular-bot-top:
regular \( x \leftrightarrow x = \text{bot} \lor x = \text{top} \)
by simp

lemma not-bot-dense:
\( x \neq \text{bot} \rightarrow - -x = \text{top} \)
by simp

end

3.3 Heyting Algebras
In this section we add a relative pseudocomplement operation to semilattices
and to lattices.

3.3.1 Heyting Semilattices
The pseudocomplement of an element \( y \) relative to an element \( z \) is the least
element whose meet with \( y \) is below \( z \). This can be stated as a Galois
connection. Specialising \( z = \text{bot} \) gives (non-relative) pseudocomplements. Many properties can already be shown if the underlying structure is just a semilattice.

class implies =
    fixes implies :: \( 'a \Rightarrow 'a \Rightarrow 'a \) (infixl \( \Rightarrow \) 65)

class Heyting-semilattice = semilattice-inf + implies +
    assumes implies-galois: \( x \cap y \leq z \iff x \leq y \Rightarrow z \)

begin

lemma implies-below-eq [simp]:
    \( y \cap (x \Rightarrow y) = y \)
  using implies-galois inf.absorb-iff1 inf.cobounded1 by blast

lemma implies-increasing:
    \( x \leq y \Rightarrow x \)
  by (simp add: inf.orderI)

lemma implies-galois-swap:
    \( x \leq y \Rightarrow z \iff y \leq x \Rightarrow z \)
  by (metis implies-galois inf-commute)

lemma implies-galois-var:
    \( x \cap y \leq z \iff y \leq x \Rightarrow z \)
  by (simp add: implies-galois-swap implies-galois)

lemma implies-galois-increasing:
    \( x \leq y \Rightarrow (x \cap y) \)
  using implies-galois by blast

lemma implies-galois-decreasing:
    \( (y \Rightarrow x) \cap y \leq x \)
  using implies-galois by blast

lemma implies-mp-below:
    \( x \cap (x \Rightarrow y) \leq y \)
  using implies-galois-decreasing inf-commute by auto

lemma implies-isotone:
    \( x \leq y \Rightarrow z \Rightarrow x \leq z \leq y \)
  using implies-galois order-trans by blast

lemma implies-antitone:
    \( x \leq y \Rightarrow y \Rightarrow z \leq x \Rightarrow z \)
  by (meson implies-galois-swap order-leq-imp)

lemma implies-isotone-inf:
    \( x \Rightarrow (y \cap z) \leq x \Rightarrow y \)
  by (simp add: implies-isotone)
lemma implies-antitone-inf:
\[ x \rightsquigarrow z \leq (x \cap y) \rightsquigarrow z \]
by (simp add: implies-antitone)

lemma implies-curry:
\[ x \rightsquigarrow (y \rightsquigarrow z) = (x \cap y) \rightsquigarrow z \]
by (metis implies-galois-decreasing implies-galois inf-assoc antisym)

lemma implies-curry-flip:
\[ x \rightsquigarrow (y \rightsquigarrow z) = y \rightsquigarrow (x \rightsquigarrow z) \]
by (simp add: implies-curry inf-commute)

lemma triple-implies [simp]:
\[(x \rightsquigarrow y) \rightsquigarrow y = x \rightsquigarrow y\]
using implies-antitone implies-galois-swap eq-iff by auto

lemma implies-dist-implies:
\[ x \rightsquigarrow (y \rightsquigarrow z) \leq (x \rightsquigarrow y) \rightsquigarrow (x \rightsquigarrow z) \]
using implies-curry implies-curry-flip by auto

lemma implies-import-inf [simp]:
\[ x \cap ((x \cap y) \rightsquigarrow z) = x \cap (y \rightsquigarrow z) \]
by (metis implies-curry implies-mp-eq inf-commute inf.absorb2)

lemma implies-dist-inf:
\[ x \rightsquigarrow (y \cap z) = (x \rightsquigarrow y) \cap (x \rightsquigarrow z) \]
proof
  have \((x \rightsquigarrow y) \cap (x \rightsquigarrow z) \cap x \leq y \cap z\)
  by (simp add: implies-galois)
  hence \((x \rightsquigarrow y) \cap (x \rightsquigarrow z) \leq x \rightsquigarrow (y \cap z)\)
  using implies-galois by blast
  thus \(?thesis\)
  by (simp add: implies-isotone eq-iff)
qed

lemma implies-itself-top:
\[ y \leq x \rightsquigarrow x \]
by (simp add: implies-galois-swap implies-increasing)

lemma inf-implies-top:
\[ z \leq (x \cap y) \rightsquigarrow x \]
using implies-galois-var le-infI1 by blast

lemma inf-inf-implies [simp]:
z \cap ((x \sqcap y) \Rightarrow x) = z \\
by (simp add: inf-implies-top inf-absorb1)

**lemma le-implies-top:**
\[
x \leq y \implies z \leq x \Rightarrow y
\]
using implies-antitone implies-itself-top order.trans by blast

**lemma le-iff-le-implies:**
\[
x \leq y \iff x \leq x \Rightarrow y
\]
using implies-galois inf-idem by force

**lemma implies-inf-isotone:**
\[
x \Rightarrow y \leq (x \cap z) \Rightarrow (y \cap z)
\]
by (metis implies-curry implies-galois-increasing implies-isotone)

**lemma implies-transitive:**
\[
(x \Rightarrow y) \cap (y \Rightarrow z) \leq x \Rightarrow z
\]
using implies-dist-implies implies-galois-var implies-increasing order-lesseq-imp
by blast

**lemma implies-inf-absorb [simp]:**
\[
x \Rightarrow (x \cap y) = x \Rightarrow y
\]
using implies-dist-inf implies-itself-top inf.absorb-iff2 by auto

**lemma implies-implies-absorb [simp]:**
\[
x \Rightarrow (x \Rightarrow y) = x \Rightarrow y
\]
by (simp add: implies-curry)

**lemma implies-inf-identity:**
\[
(x \Rightarrow y) \cap y = y
\]
by (simp add: inf-commute)

**lemma implies-itself-same:**
\[
x \Rightarrow x = y \Rightarrow y
\]
by (simp add: le-implies-top eq-iff)

end

The following class gives equational axioms for the relative pseudocomplement operation (inequalities can be written as equations).

**class** heyting-semilattice-eq = semilattice-inf + implies +
assumes implies-mp-below: \(x \cap (x \Rightarrow y) \leq y\)
and implies-galois-increasing: \(x \leq y \Rightarrow (x \cap y)\)
and implies-isotone-inf: \(x \Rightarrow (y \cap z) \leq x \Rightarrow y\)
begin
subclass heyting-semilattice
apply unfold-locales
apply (rule iffI)
apply (metis implies-galois-increasing implies-isotone-inf inf.absorb2
order-lesseq-imp)
by (metis implies-mp-below inf-commute order-trans inf-mono order-refl)
end

The following class allows us to explicitly give the pseudocomplement of
an element relative to itself.
class bounded-heyting-semilattice = bounded-semilattice-inf-top +
heyting-semilattice
begin

lemma implies-itself [simp]:
  \( x \arto x = \top \) using implies-galois inf-le2 top-le by blast

lemma implies-order:
  \( x \leq y \iff x \arto y = \top \) by (metis implies-galois inf-top
left-neutral top-unique)

lemma inf-implies [simp]:
  \( (x \land y) \arto x = \top \) using implies-order inf-le1 by blast

lemma top-implies [simp]:
  \( \top \arto x = x \) by (metis implies-mp-eq inf-top
left-neutral)
end

3.3.2 Heyting Lattices

We obtain further properties if the underlying structure is a lattice. In
particular, the lattice operations are automatically distributive in this case.
class heyting-lattice = lattice + heyting-semilattice
begin

lemma sup-distrib-inf-le:
  \( (x \lor y) \land (x \lor z) \leq x \lor (y \land z) \)
proof -
  have \( x \lor z \leq y \leadsto (x \lor (y \land z)) \)
    using implies-galois-var implies-increasing sup.bounded-iff sup.cobounded2 by blast
  hence \( x \lor y \leq (x \lor z) \leadsto (x \lor (y \land z)) \)
    using implies-galois-swap implies-increasing le-sup-iff by blast
thus \?thesis
  by (simp add: implies-galois)
qed
subclass distrib-lattice
  apply unfold-locales
  using distrib-sup-le eq-iff sup-distrib-inf-le by auto

lemma implies-isotone-sup:
  \( x \leadsto y \leq x \leadsto (y \sqcup z) \)
  by (simp add: implies-isotone)

lemma implies-antitone-sup:
  \( (x \sqcup y) \leadsto z \leq x \leadsto z \)
  by (simp add: implies-antitone)

lemma implies-sup:
  \( x \leadsto z \leq (y \leadsto z) \Rightarrow ((x \sqcup y) \leadsto z) \)
proof -
  have \( (x \leadsto z) \cap (y \leadsto z) \cap y \leq z \)
    by (simp add: implies-galois)
  hence \( (x \leadsto z) \cap (y \leadsto z) \cap (x \sqcup y) \leq z \)
    using implies-galois-swap implies-galois-var by fastforce
  thus \( \text{thesis} \)
    by (simp add: implies-galois)
qed

lemma implies-dist-sup:
  \( (x \sqcup y) \leadsto z = (x \leadsto z) \cap (y \leadsto z) \)
apply (rule antisym)
apply (simp add: implies-antitone)
by (simp add: implies-sup implies-galois)

lemma implies-antitone-isotone:
  \( (x \sqcup y) \leadsto (x \sqcap y) \leq x \leadsto y \)
by (simp add: implies-antitone-sup implies-dist-inf le-infI2)

lemma implies-antisymmetry:
  \( (x \leadsto y) \cap (y \leadsto x) = (x \sqcup y) \leadsto (x \sqcap y) \)
by (metis implies-dist-sup implies-inf-absorb inf.commute)

lemma sup-inf-implies [simp]:
  \( (x \sqcup y) \cap (x \leadsto y) = y \)
by (simp add: inf-sup-distrib2 sup.absorb2)

lemma implies-subdist-sup:
  \( (x \leadsto y) \sqcup (x \leadsto z) \leq x \leadsto (y \sqcup z) \)
by (simp add: implies-isotone)

lemma implies-subdist-inf:
  \( (x \leadsto z) \sqcup (y \leadsto z) \leq (x \sqcap y) \leadsto z \)
by (simp add: implies-antitone)
lemma implies-sup-absorb:
\[(x \dashv y) \sqcup z \leq (x \sqcup z) \dashv (y \sqcup z)\]
by (metis implies-dist-sup implies-isotone-sup implies-increasing inf-inf-implies le-sup-iff sup-inf-implies)

lemma sup-below-implies-implies:
\[x \sqcup y \leq (x \dashv y) \dashv y\]
by (simp add: implies-dist-sup implies-galois-swap implies-increasing)

end

class bounded-heyting-lattice = bounded-lattice + heyting-lattice
begin

subclass bounded-heyting-semilattice ..

lemma implies-bot [simp]:
\[\bot \dashv x = \top\]
using implies-galois top-unique by fastforce

end

3.3.3 Heyting Algebras

The pseudocomplement operation can be defined in Heyting algebras, but it is typically not part of their signature. We add the definition as an axiom so that we can use the class hierarchy, for example, to inherit results from the class pd-algebra.

class heyting-algebra = bounded-heyting-lattice + uminus +
assumes uminus-eq: \[-x \dashv x \dashv \bot\]
begin

subclass pd-algebra
apply unfold-locales
using bot-unique implies-galois uminus-eq by auto

lemma boolean-implies-below:
\[-x \sqcup y \leq x \dashv y\]
by (simp add: implies-increasing implies-isotone uminus-eq)

lemma negation-implies:
\[-(x \dashv y) = \neg \neg x \sqcap \neg y\]
proof (rule antisym)
show \[-(x \dashv y) \leq \neg \neg x \sqcap \neg y\]
using boolean-implies-below p-antitone by auto

next
have \[x \sqcap \neg y \sqcap (x \dashv y) = \bot\]
by (metis implies-mp-eq inf-p inf-bot-left inf-commute inf-left-commute)

hence \[\neg \neg x \sqcap \neg y \sqcap (x \dashv y) = \bot\]
using pp-inf-bot-iff inf-assoc by auto
thus $-x \sqcap -y \leq -(x \rightsquigarrow y)$
  by (simp add: pseudo-complement)
qed

lemma double-negation-dist-implies:
  $-((x \rightsquigarrow y) = -x \rightsquigarrow -y$
apply (rule antisym)
apply (metis pp-inf-below-iff implies-galois-decreasing implies-galois
negation-implies ppp)
by (simp add: p-antitone-iff negation-implies)
end

The following class gives equational axioms for Heyting algebras.
class heyting-algebra-eq = bounded-lattice + implies + uminus +
assumes implies-mp-eq: $x \sqcap (x \rightsquigarrow y) = x \sqcap y$
  and implies-import-inf: $x \sqcap ((x \sqcap y) \rightsquigarrow (x \rightsquigarrow z)) = x \sqcap (y \rightsquigarrow z)$
  and inf-inf-implies: $z \sqcap ((x \sqcap y) \rightsquigarrow x) = z$
  and uminus-eq-eq: $-x = x \rightsquigarrow \bot$
begin
subclass heyting-algebra
  apply unfold-locales
  apply (rule iffI)
  apply (metis implies-import-inf inf.sup-left-divisibility inf-inf-implies le_iff_inf)
  apply (metis implies-mp-eq inf.commute inf.le-sup iff inf.sup-right-isotone)
  by (simp add: uminus-eq-eq)
end

A relative pseudocomplement is not enough to obtain the Stone equation,
so we add it in the following class.
class heyting-stone-algebra = heyting-algebra +
assumes heyting-stone: $-x \sqcup -x = \top$
begin
subclass stone-algebra
  by unfold-locales (simp add: heyting-stone)
begin
end

3.3.4 Brouwer Algebras
Brouwer algebras are dual to Heyting algebras. The dual pseudocomplement of an element $y$ relative to an element $x$ is the least element whose join with $y$ is above $x$. We can now use the binary operation provided by Boolean algebras in Isabelle/HOL because it is compatible with dual relative pseudocomplements (not relative pseudocomplements).

```plaintext
class brouwer-algebra = bounded-lattice + minus + uminus +
  assumes minus-galois: $x \leq y \sqcup z \iff x - y \leq z$
  and uminus-eq-minus: $-x = \top - x$
begin

sublocale brouwer: heyting-algebra where
  inf = sup and less-eq = greater-eq
and less = greater and sup = inf and bot = top and top = bot and implies =
$\lambda x y. y - x$
apply unfold-locales
apply simp
apply simp
apply simp
apply simp
apply (metis minus-galois sup-commute)
by (simp add: uminus-eq-minus)

lemma curry-minus:
$x - (y \sqcup z) = (x - y) - z$
by (simp add: brouwer.implies-curry sup-commute)

lemma minus-subdist-sup:
$(x - z) \sqcup (y - z) \leq (x \sqcup y) - z$
by (simp add: brouwer.implies-dist-inf)

lemma inf-sup-minus:
$(x \sqcap y) \sqcup (x - y) = x$
by (simp add: inf.absorb1 brouwer.inf-sup-distrib2)

end

3.4 Boolean Algebras

This section integrates Boolean algebras in the above hierarchy. In particular, we strengthen several results shown above.

context boolean-algebra
begin

   Every Boolean algebra is a Stone algebra, a Heyting algebra and a Brouwer algebra.

subclass stone-algebra
apply unfold-locales
apply (rule iffI)
```
apply (metis compl-sup-top inf.order1 inf-bot-right inf-sup-distrib1 inf-top-right sup-inf-absorb)
  using inf.commute inf.sup-right-divisibility apply fastforce
by simp

sublocale heyting: heyting-algebra where implies = \lambda x y . −x ⊔ y
apply unfold-locales
apply (rule iffI)
using shunting-var-p sup-commute apply fastforce
using shunting-var-p sup-commute apply force
by simp

subclass brouwer-algebra
apply unfold-locales
apply (simp add: diff-eq shunting-var-p sup-commute)
by (simp add: diff-eq)

lemma huntington-3 [simp]:
  (−x ⊓ −y) ⊔ (−x ⊔ y) = x
using huntington-3-pp by auto

lemma maddux-3-1:
  x ⊔ −x = y ⊔ −y
by simp

lemma maddux-3-4:
  x ⊔ (y ⊔ −x) = z ⊔ −z
by simp

lemma maddux-3-11 [simp]:
  (x ∩ y) ∪ (x ∩ −y) = x
using brouwer.maddux-3-12 sup-commute by auto

lemma maddux-3-19:
  (−x ∩ y) ∪ (x ∩ z) = (x ∪ y) ∩ (−x ∪ z)
using maddux-3-19-pp by auto

lemma compl-inter-eq:
  x ∩ y = x ∩ z =⇒ −x ∩ y = −x ∩ z =⇒ y = z
by (metis inf-commute maddux-3-11)

lemma maddux-3-21 [simp]:
  x ⊔ (−x ∩ y) = x ⊔ y
by (simp add: sup-inf-distrib1)

lemma shunting-1:
  x ≤ y ↔ x ∩ −y = bot
by (simp add: pseudo-complement)
lemma uminus-involutive:
  \( \uminus \circ \uminus = \text{id} \)
  by auto

lemma uminus-injective:
  \( \uminus \circ f = \uminus \circ g \implies f = g \)
  by (metis comp-assoc id-o minus-comp-minus)

lemma conjugate-unique:
  \( \text{conjugate } f g \implies \text{conjugate } f h \implies g = h \)
  using conjugate-unique-p uminus-injective by blast

lemma dual-additive-additive:
  dual-additive (uminus \circ f) \implies additive f
  by (metis additive-def compl-eq-compl-iff dual-additive-def p-dist-sup o-def)

lemma conjugate-additive:
  \( \text{conjugate } f g \implies \text{additive } f \)
  by (simp add: conjugate-dual-additive dual-additive-additive)

lemma conjugate-isotone:
  \( \text{conjugate } f g \implies \text{isotone } f \)
  by (simp add: conjugate-additive additive-isotone)

lemma conjugate-char-1:
  \( \text{conjugate } f g \longleftrightarrow (\forall x y . f(x \land -(g y)) \leq f x \land -y \land g(y \land -(f x)) \leq g y \land -x) \)
  by (simp add: conjugate-char-1-pp)

lemma conjugate-char-2:
  \( \text{conjugate } f g \longleftrightarrow \text{bot } = \text{bot } \land \text{bot } = \text{bot } \land (\forall x y . f x \land y \leq f(x \land g y) \land g y \land x \leq g(y \land f x)) \)
  by (simp add: conjugate-char-2-pp)

lemma shunting:
  \( x \land y \leq z \longleftrightarrow x \leq z \lor -y \)
  by (simp add: heyting.implies-galois sup.commute)

lemma shunting-var:
  \( x \land -y \leq z \longleftrightarrow x \leq z \lor y \)
  by (simp add: shunting)

end

class non-trivial-stone-algebra = non-trivial-bounded-order + stone-algebra

class non-trivial-boolean-algebra = non-trivial-stone-algebra + boolean-algebra

end
4 Filters

This theory develops filters based on orders, semilattices, lattices and distributive lattices. We prove the ultrafilter lemma for orders with a least element. We show the following structure theorems:

- The set of filters over a directed semilattice forms a lattice with a greatest element.
- The set of filters over a bounded semilattice forms a bounded lattice.
- The set of filters over a distributive lattice with a greatest element forms a bounded distributive lattice.

Another result is that in a distributive lattice ultrafilters are prime filters. We also prove a lemma of Grätzer and Schmidt about principal filters.

We apply these results in proving the construction theorem for Stone algebras (described in a separate theory). See, for example, [4, 5, 6, 9, 17] for further results about filters.

theory Filters

imports Lattice-Basics

begin

4.1 Orders

This section gives the basic definitions related to filters in terms of orders. The main result is the ultrafilter lemma.

context ord

begin

abbreviation down :: 'a ⇒ 'a set (↓· [81] 80)
where ↓x ≡ \{ y . y ≤ x \}

abbreviation down-set :: 'a set ⇒ 'a set (⇓· [81] 80)
where ⇓X ≡ \{ y . \exists x∈X . y ≤ x \}

abbreviation is-down-set :: 'a set ⇒ bool
where is-down-set X ≡ ∀ x∈X . ∀ y . y ≤ x → y∈X

abbreviation is-principal-down :: 'a set ⇒ bool
where is-principal-down X ≡ \exists x . X = ↓x

abbreviation up :: 'a ⇒ 'a set (↑· [81] 80)
where ↑x ≡ \{ y . x ≤ y \}

abbreviation up-set :: 'a set ⇒ 'a set (⇑· [81] 80)

end
where \( \uparrow X \equiv \{ y \mid \exists x \in X . x \leq y \} \)

**abbreviation** `is-up-set :: 'a set ⇒ bool`
where `is-up-set X \equiv \forall x \in X . \forall y . x \leq y \implies y \in X`

**abbreviation** `is-principal-up :: 'a set ⇒ bool`
where `is-principal-up X \equiv \exists x . X = \uparrow x`

A filter is a non-empty, downward directed, up-closed set.

**definition** `filter :: 'a set ⇒ bool`
where `filter F \equiv (F \neq \{\}) \land (\forall x \in F . \forall y \in F . \exists z \in F . z \leq x \land z \leq y) \land is-up-set F`

**abbreviation** `proper-filter :: 'a set ⇒ bool`
where `proper-filter F \equiv filter F \land F \neq \text{UNIV}`

**abbreviation** `ultra-filter :: 'a set ⇒ bool`
where `ultra-filter F \equiv proper-filter F \land (\forall G . proper-filter G \land F \subseteq G \implies F = G)`

**end**

**context** `order`

**begin**

**lemma** `self-in-downset [simp]`:
\[ x \in \downarrow x \]
by simp

**lemma** `self-in-upset [simp]`:
\[ x \in \uparrow x \]
by simp

**lemma** `up-filter [simp]`:
\[ filter (\uparrow x) \]
using `filter-def order-lesseq-imp` by auto

**lemma** `up-set-up-set [simp]`:
\[ is-up-set (\uparrow X) \]
using `order.trans` by fastforce

**lemma** `up-injective`:
\[ \uparrow x = \uparrow y \implies x = y \]
using `antisym` by auto

**lemma** `up-antitone`:
\[ x \leq y \iff \uparrow y \subseteq \uparrow x \]
by auto
context order-bot

begin

lemma bot-in-downset [simp]:
  \( \bot \in \downarrow x \)
  by simp

lemma down-bot [simp]:
  \( \downarrow \bot = \{ \bot \} \)
  by (simp add: bot-unique)

lemma up-bot [simp]:
  \( \uparrow \bot = \text{UNIV} \)
  by simp

The following result is the ultrafilter lemma, generalised from [9, 10.17] to orders with a least element. Its proof uses Isabelle/HOL’s Zorn-Lemma, which requires closure under union of arbitrary (possibly empty) chains. Actually, the proof does not use any of the underlying order properties except bot-least.

lemma ultra-filter:
  assumes proper-filter \( F \)
  shows \( \exists G \cdot \text{ultra-filter } G \land F \subseteq G \)
proof
  let \(?A = \{ G \cdot (\text{proper-filter } G \land F \subseteq G) \lor G = \{\} \}\)
  have \( \forall C \in \text{chains } ?A \cdot \bigcup C \in ?A \)
  proof
    fix \( C :: 'a set \)
    let \(?D = C - \{\}\) 
    assume 1: \( C \in \text{chains } ?A \)
    hence 2: \( \forall x \in \bigcup ?D \cdot \exists H \in ?D . x \in H \land \text{proper-filter } H \)
      using chainsD2 by fastforce
    have 3: \( \bigcup ?D = \bigcup C \)
      by blast
    have \( \bigcup ?D \in ?A \)
    proof (cases \(?D = \{\}\) )
      assume \(?D = \{\}\) 
      thus \(?thesis \)
        by auto
    next
      assume 4: \( ?D \neq \{\} \)
      then obtain \( G \) where \( G \in ?D \)
        by auto
      hence 5: \( F \subseteq \bigcup ?D \)
        using 1 chainsD2 by blast
      have 6: \( is-up-set (\bigcup ?D) \)
        proof
fix \( x \)
assume \( x \in \bigcup \mathcal{D} \)
then obtain \( H \) where \( x \in H \land H \in \mathcal{D} \land \text{filter } H \)
using 2 by auto
thus \( \forall y . \ x \leq y \limp y \in \bigcup \mathcal{D} \)
using filter-def UnionI by fastforce
qed
have 7: \( \bigcup \mathcal{D} \neq \text{UNIV} \)
proof (rule ccontr)
assume \( \neg \bigcup \mathcal{D} \neq \text{UNIV} \)
then obtain \( H \) where \( \text{bot} \in H \land \text{proper-filter } H \)
using 2 by blast
thus \( \text{False} \)
by (meson UNIV-I bot-least filter-def subsetI subset-antisym)
qed

\{ 
fix \( x \ y \)
assume \( x \in \bigcup \mathcal{D} \land y \in \bigcup \mathcal{D} \)
then obtain \( H \ I \) where 8: \( x \in H \land H \in \mathcal{D} \land \text{filter } H \land y \in I \land I \in \mathcal{D} \land \text{filter } I \)
using 2 by metis
have \( \exists z \in \bigcup \mathcal{D} . \ z \leq x \land z \leq y \)
proof (cases \( H \subseteq I \))
assume \( H \subseteq I \)
hence \( \exists z \in I . \ z \leq x \land z \leq y \)
using 8 by (metis subsetCE filter-def)
thus \( \text{thesis} \)
using 8 by (metis UnionI)
next
assume \( \neg (H \subseteq I) \)
hence \( I \subseteq H \)
using 8 by (meson DiffE chainsD)
hence \( \exists z \in H . \ z \leq x \land z \leq y \)
using 8 by (metis subsetCE filter-def)
thus \( \text{thesis} \)
using 8 by (metis UnionI)

qed

thus \( \text{thesis} \)
using 4 5 6 7 filter-def by auto
qed

thus \( \bigcup C \in \mathcal{A} \)
using 3 by simp
qed

hence \( \exists M \in \mathcal{A} . \ \forall X \in \mathcal{A} . \ M \subseteq X \limp X = M \)
by (rule Zorn-Lemma)
then obtain \( M \) where 9: \( M \in \mathcal{A} \land (\forall X \in \mathcal{A} . \ M \subseteq X \limp X = M) \)
by auto
hence 10: \( M \neq \{\} \)

41
using assms filter-def by auto
{
fix G
assume 11: proper-filter G ∧ M ⊆ G
hence F ⊆ G
using 9 10 by blast
hence M = G
using 9 11 by auto
}
thus thesis
using 9 10 by blast
qed
end

context order-top
begin

lemma down-top [simp]:
\downarrow top = UNIV
by simp

lemma top-in-upset [simp]:
top ∈ \uparrow x
by simp

lemma up-top [simp]:
\uparrow top = \{ top \}
by (simp add: top-unique)

lemma filter-top [simp]:
filter \{ top \}
using filter-def top-unique by auto

lemma top-in-filter [simp]:
filter F ⇒ top ∈ F
using filter-def by fastforce
end

The existence of proper filters and ultrafilters requires that the underlying order contains at least two elements.

context non-trivial-order
begin

lemma proper-filter-exists:
\exists F . proper-filter F
proof –
from consistent obtain x y : 'a where x ≠ y
by auto
hence \( \uparrow x \neq \text{UNIV} \lor \uparrow y \neq \text{UNIV} \)
using antisym by blast
hence proper-filter \((\uparrow x) \lor\) proper-filter \((\uparrow y)\)
by simp
thus thesis
by blast
qed

context non-trivial-order-bot
begin

lemma ultra-filter-exists:
\( \exists F . \text{ultra-filter } F \)
using ultra-filter proper-filter-exists by blast

end

context non-trivial-bounded-order
begin

lemma proper-filter-top:
proper-filter \{top\}
using bot-not-top filter-top by blast

lemma ultra-filter-top:
\( \exists G . \text{ultra-filter } G \land \text{top } \in G \)
using ultra-filter proper-filter-top by fastforce

end

4.2 Lattices

This section develops the lattice structure of filters based on a semilattice structure of the underlying order. The main results are that filters over a directed semilattice form a lattice with a greatest element and that filters over a bounded semilattice form a bounded lattice.

context semilattice-sup
begin

abbreviation prime-filter :: 'a set \Rightarrow \text{bool}
where prime-filter \( F \equiv \text{proper-filter } F \land (\forall x y . x \sqcup y \in F \rightarrow x \in F \lor y \in F)\)

end

context semilattice-inf
begin

lemma filter-inf-closed:
filter F \Rightarrow x \in F \Rightarrow y \in F \Rightarrow x \cap y \in F
by (meson filter-def inf.boundedI)

lemma filter-univ:
filter UNIV
by (meson UNIV-I UNIV-not-empty filter-def inf.bounded1 inf.bounded2)

The operation filter-sup is the join operation in the lattice of filters.

abbreviation filter-sup F G \equiv \{ z . \exists x \in F . \exists y \in G . x \cap y \leq z \}

lemma filter-sup:
assumes filter F and filter G
shows filter (filter-sup F G)
proof
have F \neq {} \land G \neq {}
using assms filter-def by blast
hence 1: filter-sup F G \neq {}
by blast
have 2: \forall x \in filter-sup F G . \forall y \in filter-sup F G . \exists z \in filter-sup F G . z \leq x \land z \leq y
proof
fix x
assume x \in filter-sup F G
then obtain t u where 3: t \in F \land u \in G \land t \cap u \leq x
by auto
show \forall y \in filter-sup F G . \exists z \in filter-sup F G . z \leq x \land z \leq y
proof
fix y
assume y \in filter-sup F G
then obtain v w where 4: v \in F \land w \in G \land v \cap w \leq y
by auto
let \exists z = (t \cap v) \cap (u \cap w)
have 5: \exists z \leq x \land \exists z \leq y
using 3 4 by (meson order-trans inf.cobounded1 inf.cobounded2 inf-mono)
have \exists z \in filter-sup F G
using assms 3 4 filter-inf-closed by blast
thus \exists z \in filter-sup F G . z \leq x \land z \leq y
using 5 by blast
qed
qed

have \forall x \in filter-sup F G . \forall y . x \leq y \longrightarrow y \in filter-sup F G
using order-trans by blast
thus \forall y . x \leq y \longrightarrow y \in filter-sup F G
using 1 2 filter-def by presburger
qed
lemma filter-sup-left-upper-bound:
  assumes filter G
  shows F ⊆ filter-sup F G
proof –
  from assms obtain y where y∈G
    using all-not-in-conv filter-def by auto
  thus ?thesis
    using inf.cobounded1 by blast
qed

lemma filter-sup-symmetric:
  filter-sup F G = filter-sup G F
using inf.commute by fastforce

lemma filter-sup-right-upper-bound:
  filter F ⇒ G ⊆ filter-sup F G
using filter-sup-symmetric filter-sup-left-upper-bound by simp

lemma filter-sup-least-upper-bound:
  assumes filter H
    and F ⊆ H
    and G ⊆ H
  shows filter-sup F G ⊆ H
proof
  fix x
  assume x ∈ filter-sup F G
  then obtain y z where 1: y ∈ F ∧ z ∈ G ∧ y ∩ z ⊆ x
    by auto
  hence y ∈ H ∧ z ∈ H
    using assms(2−3) by auto
  hence y ∩ z ∈ H
    by (simp add: assms(1) filter-inf-closed)
  thus x ∈ H
    using 1 assms(1) filter-def by auto
qed

lemma filter-sup-left-isotone:
  G ⊆ H ⇒ filter-sup G F ⊆ filter-sup H F
by blast

lemma filter-sup-right-isotone:
  G ⊆ H ⇒ filter-sup F G ⊆ filter-sup F H
by blast

lemma filter-sup-right-isotone-var:
  filter-sup F (G ∩ H) ⊆ filter-sup F H
by blast
lemma up-dist-inf:
\[ \uparrow(x \sqcap y) = \text{filter-sup} \ (\uparrow x) \ (\uparrow y) \]

proof
  show \( \uparrow(x \sqcap y) \subseteq \text{filter-sup} \ (\uparrow x) \ (\uparrow y) \)
  by blast
next
  show \( \text{filter-sup} \ (\uparrow x) \ (\uparrow y) \subseteq \uparrow(x \sqcap y) \)
  proof
    fix \( z \)
    assume \( z \in \text{filter-sup} \ (\uparrow x) \ (\uparrow y) \)
    then obtain \( u \ v \) where \( u \in \uparrow x \land v \in \uparrow y \land u \sqcap v \leq z \)
    by auto
    hence \( x \sqcap y \leq z \)
    using order.trans inf-mono by blast
    thus \( z \in \uparrow(x \sqcap y) \)
    by blast
  qed
qed

The following result is part of [9, Exercise 2.23].

lemma filter-inf-filter [simp]:
  assumes filter \( F \)
  shows filter \( \uparrow \{ y . \exists z \in \mathcal{F} . x \sqcap z = y \} \)

proof
  let \( ?G = \uparrow \{ y . \exists z \in \mathcal{F} . x \sqcap z = y \} \)
  have \( \mathcal{F} \neq \{ \} \)
    using assms filter-def by simp
  hence \( 1: \ ?G \neq \{ \} \)
  by blast
  have \( 2: \ \text{is-up-set} \ ?G \)
  by auto
  
  fix \( y \ z \)
  assume \( y \in \ ?G \land z \in \ ?G \)
  then obtain \( v \ w \) where \( v \in \mathcal{F} \land w \in \mathcal{F} \land x \sqcap v \leq y \land x \sqcap w \leq z \)
  by auto
  hence \( v \sqcap w \in \mathcal{F} \land x \sqcap (v \sqcap w) \leq y \sqcap z \)
  by (meson assms filter-inf-closed order.trans inf.boundedI inf.cobounded1 inf.cobounded2)
  hence \( \exists u \in \ ?G . u \leq y \land u \leq z \)
  by auto
  
  hence \( \forall x \in \ ?G . \forall y \in \ ?G . \exists z \in \ ?G . z \leq x \land z \leq y \)
  by auto
  thus \( \text{thesis} \)
  using \( 1 \ 2 \ \text{filter-def} \) by presburger
qed

end
context directed-semilattice-inf

begin

Set intersection is the meet operation in the lattice of filters.

lemma filter-inf:
assumes filter F and filter G
shows filter \((F \cap G)\)
proof (unfold filter-def, intro conjI)
from assms obtain x y where 1: \(x \in F \land y \in G\)
using all-not-in-cone filter-def by auto
from ub obtain z where \(x \leq z \land y \leq z\)
by auto
hence \(z \in F \cap G\)
using 1 by (meson assms Int-iff filter-def)
thus \(F \cap G \neq \{\}\)
by blast
next
show is-up-set \((F \cap G)\)
by (meson assms Int-iff filter-def)
next
show \(\forall x \in F \cap G. \forall y \in F \cap G. \exists z \in F \cap G. z \leq x \land z \leq y\)
by (metis assms Int-iff filter-inf-closed inf.cobounded2 inf.commute)
qed

end

We introduce the following type of filters to instantiate the lattice classes
and thereby inherit the results shown about lattices.

typedef (overloaded) \('a filter = \{ F::'a::order set . filter F \}\)
by (meson mem-Collect-eq up-filter)

lemma simp-filter [simp]:
filter (Rep-filter x)
using Rep-filter by simp

setup-lifting type-definition-filter

The set of filters over a directed semilattice forms a lattice with a greatest
element.

instantiation filter :: (directed-semilattice-inf) bounded-lattice-top
begin

lift-definition top-filter :: \('a filter is UNIV\)
by (simp add: filter-univ)

lift-definition sup-filter :: \('a filter ⇒ 'a filter ⇒ 'a filter is filter-sup\)
by (simp add: filter-sup)
lift-definition \textit{inf-filter} :: 'a filter \Rightarrow 'a filter \Rightarrow 'a filter is inter
by (simp add: \textit{filter-inf})

lift-definition \textit{less-eq-filter} :: 'a filter \Rightarrow 'a filter \Rightarrow \text{bool} is subset-eq .

lift-definition \textit{less-filter} :: 'a filter \Rightarrow 'a filter \Rightarrow \text{bool} is subset .

instance
apply intro-classes
apply (simp add: \textit{less-eq-filter}.rep-eq \textit{less-filter}.rep-eq \textit{inf}.less-le-not-le)
apply (simp add: \textit{less-eq-filter}.rep-eq)
apply (simp add: \textit{less-eq-filter}.rep-eq)
apply (simp add: \textit{Rep-filter-inject} \textit{less-eq-filter}.rep-eq)
apply (simp add: \textit{inf-filter}.rep-eq \textit{less-eq-filter}.rep-eq)
apply (simp add: \textit{inf-filter}.rep-eq \textit{less-eq-filter}.rep-eq)
apply (simp add: \textit{inf-filter}.rep-eq \textit{less-eq-filter}.rep-eq \textit{sup-filter}.rep-eq)
apply (simp add: \textit{less-eq-filter}.rep-eq \textit{filter-sup-left-upper-bound} \textit{sup-filter}.rep-eq)
apply (simp add: \textit{less-eq-filter}.rep-eq \textit{filter-sup-right-upper-bound} \textit{sup-filter}.rep-eq)
apply (simp add: \textit{less-eq-filter}.rep-eq \textit{filter-sup-least-upper-bound} \textit{sup-filter}.rep-eq)
by (simp add: \textit{less-eq-filter}.rep-eq \textit{top-filter}.rep-eq)

end

context \textit{bounded-semilattice-inf-top}
begin
abbreviation \textit{filter-complements} \( F \ G \equiv \text{filter} \ F \wedge \text{filter} \ G \land \text{filter-sup} \ F \ G = \text{UNIV} \land F \cap G = \{ \text{top} \} \)

end

\textbf{The set of filters over a bounded semilattice forms a bounded lattice.}

instantiation \textit{filter} :: \textit{(bounded-semilattice-inf-top)} \textit{bounded-lattice}
begin

lift-definition \textit{bot-filter} :: 'a filter is \{ \text{top} \}
by simp

instance
by intro-classes (simp add: \textit{less-eq-filter}.rep-eq \textit{bot-filter}.rep-eq)

end

context \textit{lattice}
begin
For convenience, the following function injects principal filters into the filter type. We cannot define it in the order class since the type filter requires the sort constraint order that is not available in the class. The result of the function is a filter by lemma up-filter.

**abbreviation** up-filter :: `'a::order ⇒ 'a filter

**where** up-filter x ≡ Abs-filter (∧x)

**lemma** up-filter-dist-inf:

up-filter ((x::'a::lattice) ⊓ y) = up-filter x ⊓ up-filter y

**by** (simp add: eq-onp-def sup-filter.abs-eq up-dist-inf)

**lemma** up-filter-dist-sup:

up-filter ((x::'a::lattice) ⊔ y) = up-filter x ⊔ up-filter y

**by** (metis eq-onp-def inf-filter.abs-eq up-dist-sup up-filter)

**lemma** up-filter-injective:

up-filter x = up-filter y =⇒ x = y

**by** (metis Abs-filter-inject mem-Collect-eq up-filter up-injective)

**lemma** up-filter-antitone:

x ≤ y =⇒ up-filter y ≤ up-filter x

**by** (metis eq-onp-same-args less-eq-filter.abs-eq up-antitone up-filter)

The following definition applies a function to each element of a filter. The subsequent lemma gives conditions under which the result of this application is a filter.

**abbreviation** filter-map :: ('a::order ⇒ 'b::order) ⇒ 'a filter ⇒ 'b filter

**where** filter-map f F ≡ Abs-filter (f ' Rep-filter F)

**lemma** filter-map-filter:

**assumes** mono f

and ∀ x y. f x ≤ y → (∃ z. x ≤ z ∧ y = f z)

**shows** filter (f ' Rep-filter F)

**proof** (unfold filter-def, intro conjI)

**show** f ' Rep-filter F ≠ {}

**by** (metis empty-is-image filter-def simp-filter)

**next**

**show** ∀ x∈f ' Rep-filter F . ∀ y∈f ' Rep-filter F . ∃ z∈f ' Rep-filter F . z ≤ x ∧ z ≤ y

**proof** (intro ballI)

**fix** x y

**assume** x ∈ f ' Rep-filter F and y ∈ f ' Rep-filter F
then obtain $u \; v$ where $1 \colon x = f \; u \land u \in \text{Rep-filter} \; F \land y = f \; v \land v \in \text{Rep-filter} \; F$ 
by auto

then obtain $w$ where $w \leq u \land w \leq v \land w \in \text{Rep-filter} \; F$
by (meson filter-def simp-filter)

thus $\exists z \in f \; \text{Rep-filter} \; F \cdot z \leq x \land z \leq y$
using 1 assms(1) mono-def rev-image-eqI by blast

qed

next

show is-up-set ($f \; \text{Rep-filter} \; F$)

proof

fix $x$

assume $x \in f \; \text{Rep-filter} \; F$

then obtain $u$ where $1 \colon x = f \; u \land u \in \text{Rep-filter} \; F$
by auto

show $\forall y \cdot x \leq y \Longrightarrow y \in f \; \text{Rep-filter} \; F$

proof (rule allI, rule impI)

fix $y$

assume $x \leq y$

hence $f \; u \leq y$

using 1 by simp

then obtain $z$ where $u \leq z \land y = f \; z$

using assms(2) by auto

thus $y \in f \; \text{Rep-filter} \; F$

using 1 by (meson image_iff filter-def simp-filter)

qed

qed

qed

4.3 Distributive Lattices

In this section we additionally assume that the underlying order forms a distributive lattice. Then filters form a bounded distributive lattice if the underlying order has a greatest element. Moreover ultrafilters are prime filters. We also prove a lemma of Grätzer and Schmidt about principal filters.

context distrib-lattice
begin

lemma filter-sup-left-dist-inf:

assumes $\text{filter} \; F$

and $\text{filter} \; G$

and $\text{filter} \; H$

shows $\text{filter-sup} \; F \; (G \cap H) = \text{filter-sup} \; F \; G \cap \text{filter-sup} \; F \; H$

proof

show $\text{filter-sup} \; F \; (G \cap H) \subseteq \text{filter-sup} \; F \; G \cap \text{filter-sup} \; F \; H$

using $\text{filter-sup-right-isotone-var}$ by blast

next
show \( \text{filter-sup } F \cap \text{filter-sup } F \subseteq \text{filter-sup } F \) \((G \cap H)\)

proof

fix \(x\)

assume \(x \in \text{filter-sup } F \cap \text{filter-sup } F \)

then obtain \(t u v w\) where \(I: t \in F \land u \in G \land v \in F \land w \in H \land t \cap u \leq x \land v \cap w \leq x\)

by auto

let \(?y = t \cap v\)

let \(?z = u \sqcup w\)

have \(?y \in G \cap H\)

using 1 by (simp add: assms(1) filter-inf-closed)

have \(?z \in F\)

also have \(\ldots \leq (t \cap u) \cup (v \cap w)\)

by (metis inf.cobounded1 inf.cobounded2 inf.left-idem inf.mono sup.mono)

also have \(\ldots \leq x\)

using 1 by (simp add: le-supI)

finally show \(x \in \text{filter-sup } F \) \((G \cap H)\)

using 2 3 by blast

qed

lemma filter-inf-principal-rep:

\(F \cap G = \uparrow z \Rightarrow (\exists x \in F . \exists y \in G . z = x \sqcup y)\)

by force

lemma filter-sup-principal-rep:

assumes \(\text{filter } F\)

and \(\text{filter } G\)

and \(\text{filter-sup } F G = \uparrow z\)

shows \(\exists x \in F . \exists y \in G . z = x \cap y\)

proof –

from assms(3) obtain \(x y\) where \(I: x \in F \land y \in G \land x \cap y \leq z\)

using order-refl by blast

hence \(2: x \sqcup z \in F \land y \sqcup z \in G\)

by (meson assms(1–2) sup-ge1 filter-def)

have \((x \sqcup z) \cap (y \sqcup z) = z\)

using 1 sup-absorb2 sup-inf-distrib2 by fastforce

thus \(?\text{thesis}\)

using 2 by force

qed

lemma inf-sup-principal-aux:

assumes \(\text{filter } F\)

and \(\text{filter } G\)

and \(\text{is-principal-up } (\text{filter-sup } F G)\)

and \(\text{is-principal-up } (F \cap G)\)

\[51\]
shows is-principal-up $F$

proof –

from assms(3–4) obtain $x$ $y$ where 1: \( \text{filter-sup } F \cap G = \uparrow y \)

by blast

from filter-inf-principal-rep obtain $t$ $u$ where 2: \( t \in F \land u \in G \land y = t \sqcup u \)

using 1 by meson

from filter-sup-principal-rep obtain $v$ $w$ where 3: \( v \in F \land w \in G \land \exists x \in v \sqcap w \)

using 1 by (meson assms(1–2))

have \( t \in \text{filter-sup } F \cap G \land u \in \text{filter-sup } F \cap G \)

using 2 inf.cobounded1 inf.cobounded2 by blast

hence \( x \leq t \land x \leq u \)

using 1 by blast

hence 4: \( (t \sqcap v) \sqcup (u \sqcap w) = x \)

using 3 by (simp add: inf.absorb2 inf.assoc inf.left-commute)

have \( (t \sqcap v) \sqcup (u \sqcap w) \in F \land (t \sqcap v) \sqcap (u \sqcap w) \in G \)

using 2 3 by (metis (no-types, lifting) assms(1–2) filter-inf-closed)

hence \( y \leq (t \sqcap v) \sqcup (u \sqcap w) \)

using 1 Int-iff by blast

hence 5: \( (t \sqcap v) \sqcup (u \sqcap w) = y \)

using 2 by (simp add: antisym inf.coboundedII)

have \( F = \uparrow (t \sqcap v) \)

proof

show \( F \subseteq \uparrow (t \sqcap v) \)

proof

fix $z$

assume 6: \( z \in F \)

hence \( z \in \text{filter-sup } F \cap G \)

using 2 inf.cobounded1 by blast

hence \( x \leq z \)

using 1 by simp

hence 7: \( (t \sqcap v \sqcap z) \sqcup (u \sqcap w) = x \)

using 4 by (metis inf.absorb1 inf.assoc inf.commute)

have \( z \sqcup u \in F \land z \sqcup u \in G \land z \sqcup w \in F \land z \sqcup w \in G \)

using 2 3 6 by (meson assms(1–2) filter-def sup-ge1 sup-ge2)

hence \( y \leq (z \sqcup u) \sqcap (z \sqcup w) \)

using 1 Int-iff filter-inf-closed by auto

hence 8: \( (t \sqcap v \sqcap z) \sqcup (u \sqcap w) = y \)

using 5 by (metis inf.absorb1 sup.commute sup-inf-distrib2)

have \( t \sqcap v \sqcap z = t \sqcap v \)

using 4 5 7 8 relative-equality by blast

thus \( z \in \uparrow (t \sqcap v) \)

by (simp add: inf.orderI)

qed

next

show \( \uparrow (t \sqcap v) \subseteq F \)

proof

fix $z$

have 9: \( t \sqcap v \in F \)
using 2 3 by (simp add: assms(1) filter-inf-closed)
assume z ∈ ↑(t ∩ v)
hence t ∩ v ≤ z by simp
thus z ∈ F
using assms(1) 9 filter-def by auto
qed
qed
thus ?thesis
by blast
qed

The following result is [18, Lemma II]. If both join and meet of two filters are principal filters, both filters are principal filters.

lemma inf-sup-principal:
  assumes filter F
  and filter G
  and is-principal-up (filter-sup F G)
  and is-principal-up (F ∩ G)
  shows is-principal-up F ∧ is-principal-up G
proof –
  have filter G ∧ filter F ∧ is-principal-up (filter-sup G F) ∧ is-principal-up (G ∩ F)
    by (simp add: assms Int-commute filter-sup-symmetric)
    thus ?thesis
    using assms(3) inf-sup-principal-aux by blast
qed

lemma filter-sup-absorb-inf: filter F ⇒ filter G ⇒ filter-sup (F ∩ G) G = G
  by (simp add: filter-inf filter-sup-least-upper-bound filter-sup-left-upper-bound filter-sup-symmetric subset-antisym)

lemma filter-inf-absorb-sup: filter F ⇒ filter G ⇒ filter-sup F G ∩ G = G
  apply (rule subset-antisym)
  apply simp
  by (simp add: filter-sup-right-upper-bound)

lemma filter-inf-right-dist-sup:
  assumes filter F
  and filter G
  and filter H
  shows filter-sup F G ∩ H = filter-sup (F ∩ H) (G ∩ H)
proof –
  have filter-sup (F ∩ H) (G ∩ H) = filter-sup (F ∩ H) G ∩ filter-sup (F ∩ H) H
    by (simp add: assms filter-sup-left-dist-inf filter-inf)
  also have ... = filter-sup (F ∩ H) G ∩ H
    using assms(1,3) filter-sup-absorb-inf by simp
  also have ... = filter-sup F G ∩ filter-sup G H ∩ H
    using assms filter-sup-left-dist-inf filter-sup-symmetric by simp

53
also have \( \ldots = \text{filter-sup} \ F \ G \cap H \)
by \((\text{simp add: assms(2-3) filter-inf-absorb-sup semilattice-inf-class.inf-assoc})\)
finally show \(?\text{thesis}\)
by \(\text{simp}\)
qed

The following result generalises [9, 10.11] to distributive lattices as re-marked after that section.

lemma \(\text{ultra-filter-prime}\):
assumes \(\text{ultra-filter} \ F\)
shows \(\text{prime-filter} \ F\)
proof –
{ 
  fix \(x\ y\)
  assume 1: \(x \sqcup y \in F \land x \notin F\)
  let \(?G = \{ z . \exists w \in F . x \sqcap w = z \}\)
  have 2: \(\text{filter} \ ?G\)
    using \(\text{assms filter-inf-filter}\) by \(\text{simp}\)
  have \(x \in ?G\)
    using \(1\) by \(\text{auto}\)
  hence 3: \(F \neq ?G\)
    using \(1\) by \(\text{auto}\)
  have \(F \subseteq ?G\)
    using \(\text{inf-le\text{-}2 order-trans}\) by \(\text{blast}\)
  hence \(?G = \text{UNIV}\)
    using \(2\ 3\ \text{assms}\) by \(\text{blast}\)
  then obtain \(z\) where 4: \(z \in F \land x \sqcap z \leq y\)
    by \(\text{blast}\)
  hence \(y \sqcap z = (x \sqcup y) \sqcap z\)
    by \((\text{simp add: inf-sup-distrib2 sup-absorb2})\)
  also have \(\ldots \in F\)
    using \(1\ 4\ \text{assms filter-inf-closed}\) by \(\text{auto}\)
  finally have \(y \in F\)
    using \(\text{assms}\) by \((\text{simp add: filter-def})\)
}
thus \(\text{?thesis}\)
using \(\text{assms}\) by \(\text{blast}\)
qed

context \(\text{distrib-lattice-bot}\)
begin

lemma \(\text{prime-filter}\):
\(\text{proper-filter} \ F \implies \exists G . \text{prime-filter} \ G \land F \subseteq G\)
by \(\text{(metis ultra-filter ultra-filter-prime)}\)

end
context distrib-lattice-top
begin

lemma complemented-filter-inf-principal:
  assumes filter-complements F G
  shows is-principal-up (F ∩ ↑x)
proof
  have 1: filter F ∧ filter G
    by (simp add: assms)
  hence 2: filter (F ∩ ↑x) ∧ filter (G ∩ ↑x)
    by (simp add: filter-inf)
  have (F ∩ ↑x) ∩ (G ∩ ↑x) = {top}
    using assms Int-assoc Int-insert-left-if1 inf-bot-left inf-sup-aci(3) top-in-upset
    inf.idem by auto
  hence 3: is-principal-up ((F ∩ ↑x) ∩ (G ∩ ↑x))
    using up-top by blast
  have filter-sup (F ∩ ↑x) (G ∩ ↑x) = filter-sup F G ∩ ↑x
    using 1 filter-inf-right-dist-sup up-filter by auto
  also have ... = ↑x
    by (simp add: assms)
  finally have is-principal-up (filter-sup (F ∩ ↑x) (G ∩ ↑x))
    by auto
  thus ?thesis
    using 1 2 3 inf-sup-principal-aux by blast
qed

end

The set of filters over a distributive lattice with a greatest element forms a bounded distributive lattice.

instantiation filter :: (distrib-lattice-top) bounded-distrib-lattice
begin

instance
proof
  fix x y z :: 'a filter
  have Rep-filter (x ⊓ (y ∩ z)) = filter-sup (Rep-filter x) (Rep-filter (y ∩ z))
    by (simp add: sup-filter.rep-eq)
  also have ... = filter-sup (Rep-filter x) (Rep-filter y ∩ Rep-filter z)
    by (simp add: inf-filter.rep-eq)
  also have ... = filter-sup (Rep-filter x) (Rep-filter y) ∩ filter-sup (Rep-filter x)
    (Rep-filter z)
    by (simp add: filter-sup-left-dist-inf)
  also have ... = Rep-filter ((x ⊓ y) ∩ (x ⊓ z))
    by (simp add: sup-filter.rep-eq)
  also have ... = Rep-filter ((x ⊓ y) ∩ (x ∪ z))
    by (simp add: inf-filter.rep-eq)
  finally show x ⊓ (y ∩ z) = (x ⊓ y) ∩ (x ∪ z)

55
by (simp add: Rep-filter-inject)
qed
end
end

5 Stone Construction

This theory proves the uniqueness theorem for the triple representation of Stone algebras and the construction theorem of Stone algebras [7, 21]. Every Stone algebra \( S \) has an associated triple consisting of

* the set of regular elements \( B(S) \) of \( S \),
* the set of dense elements \( D(S) \) of \( S \), and
* the structure map \( \varphi(S) : B(S) \to F(D(S)) \) defined by \( \varphi(x) = \uparrow x \cap D(S) \).

Here \( F(X) \) is the set of filters of a partially ordered set \( X \). We first show that

* \( B(S) \) is a Boolean algebra,
* \( D(S) \) is a distributive lattice with a greatest element, whence \( F(D(S)) \) is a bounded distributive lattice, and
* \( \varphi(S) \) is a bounded lattice homomorphism.

Next, from a triple \( T = (B, D, \varphi) \) such that \( B \) is a Boolean algebra, \( D \) is a distributive lattice with a greatest element and \( \varphi : B \to F(D) \) is a bounded lattice homomorphism, we construct a Stone algebra \( S(T) \). The elements of \( S(T) \) are pairs taken from \( B \times F(D) \) following the construction of [21]. We need to represent \( S(T) \) as a type to be able to instantiate the Stone algebra class. Because the pairs must satisfy a condition depending on \( \varphi \), this would require dependent types. Since Isabelle/HOL does not have dependent types, we use a function lifting instead. The lifted pairs form a Stone algebra.

Next, we specialise the construction to start with the triple associated with a Stone algebra \( S \), that is, we construct \( S(B(S), D(S), \varphi(S)) \). In this case, we can instantiate the lifted pairs to obtain a type of pairs (that no longer implements a dependent type). To achieve this, we construct an embedding of the type of pairs into the lifted pairs, so that we inherit the Stone algebra axioms (using a technique of universal algebra that works for universally quantified equations and equational implications).
Next, we show that the Stone algebras $S(B(S), D(S), \varphi(S))$ and $S$ are isomorphic. We give explicit mappings in both directions. This implies the uniqueness theorem for the triple representation of Stone algebras.

Finally, we show that the triples $(B(S(T)), D(S(T)), \varphi(S(T)))$ and $T$ are isomorphic. This requires an isomorphism of the Boolean algebras $B$ and $B(S(T))$, an isomorphism of the distributive lattices $D$ and $D(S(T))$, and a proof that they preserve the structure maps. We give explicit mappings of the Boolean algebra isomorphism and the distributive lattice isomorphism in both directions. This implies the construction theorem of Stone algebras. Because $S(T)$ is implemented by lifted pairs, so are $B(S(T))$ and $D(S(T))$; we therefore also lift $B$ and $D$ to establish the isomorphisms.

theory Stone-Construction

imports P-Algebras Filters

begin

5.1 Triples

This section gives definitions of lattice homomorphisms and isomorphisms and basic properties. It concludes with a locale that represents triples as discussed above.

class sup-inf-top-bot-uminus = sup + inf + top + bot + uminus

class sup-inf-top-bot-uminus-ord = sup-inf-top-bot-uminus + ord

context p-algebra

begin

subclass sup-inf-top-bot-uminus-ord .

end

abbreviation sup-homomorphism :: (′a::sup ⇒ ′b::sup) ⇒ bool
where sup-homomorphism f ≡ ∀ x y . f (x ⊔ y) = f x ⊔ f y

abbreviation inf-homomorphism :: (′a::inf ⇒ ′b::inf) ⇒ bool
where inf-homomorphism f ≡ ∀ x y . f (x ⊓ y) = f x ⊓ f y

abbreviation sup-inf-homomorphism :: (′a::{sup,inf} ⇒ ′b::{sup,inf}) ⇒ bool
where sup-inf-homomorphism f ≡ sup-homomorphism f ∧ inf-homomorphism f

abbreviation sup-inf-top-homomorphism :: (′a::{sup,inf,top} ⇒ ′b::{sup,inf,top}) ⇒ bool
where sup-inf-top-homomorphism f ≡ sup-inf-homomorphism f ∧ f top = top

abbreviation sup-inf-top-bot-homomorphism :: (′a::{sup,inf,top,bot} ⇒ ′b::{sup,inf,top,bot}) ⇒ bool
where sup-inf-top-bot-homomorphism \( f \equiv \text{sup-inf-top-homomorphism } f \land f \text{ bot} \)

abbreviation bounded-lattice-homomorphism :: ('a::bounded-lattice \Rightarrow 'b::bounded-lattice) \Rightarrow bool
  where bounded-lattice-homomorphism \( f \equiv \text{sup-inf-top-bot-homomorphism } f \)

abbreviation sup-inf-top-bot-uminus-homomorphism ::
  ('a::sup-inf-top-bot-uminus \Rightarrow 'b::sup-inf-top-bot-uminus) \Rightarrow bool
  where sup-inf-top-bot-uminus-homomorphism \( f \equiv \text{sup-inf-top-bot-homomorphism } f \land (\forall x. f (-x) = -f x) \)

abbreviation sup-inf-top-bot-uminus-ord-homomorphism ::
  ('a::sup-inf-top-bot-uminus-ord \Rightarrow 'b::sup-inf-top-bot-uminus-ord) \Rightarrow bool
  where sup-inf-top-bot-uminus-ord-homomorphism \( f \equiv \text{sup-inf-top-bot-homomorphism } f \land (\forall x y. x \leq y \rightarrow f x \leq f y) \)

case lemma sup-homomorphism-mono:
  sup-homomorphism \( (f::'a::semilattice-sup \Rightarrow 'b::semilattice-sup) \Rightarrow \text{mono } f \)
  by (metis le_iff_sup monoI)

lemma sup-isomorphism-ord-isomorphism:
  assumes sup-homomorphism \( (f::'a::semilattice-sup \Rightarrow 'b::semilattice-sup) \)
  and bij \( f \)
  shows \( x \leq y \leftrightarrow f x \leq f y \)
  proof
    assume \( x \leq y \)
    thus \( f x \leq f y \)
A triple consists of a Boolean algebra, a distributive lattice with a greatest element, and a structure map. The Boolean algebra and the distributive lattice are represented as HOL types. Because both occur in the type of the structure map, the triple is determined simply by the structure map and its HOL type. The structure map needs to be a bounded lattice homomorphism.

locale triple = 
  fixes phi :: 'a::boolean-algebra ⇒ 'b::distrib-lattice-top filter
  assumes hom: bounded-lattice-homomorphism phi

5.2 The Triple of a Stone Algebra
In this section we construct the triple associated to a Stone algebra.

5.2.1 Regular Elements
The regular elements of a Stone algebra form a Boolean subalgebra.

typedef (overloaded) 'a regular = regular-elements::'a::stone-algebra set
  by auto

lemma simp-regular [simp]:
  ∃ y . Rep-regular x = − y
using Rep-regular by simp

setup-lifting type-definition-regular

instantiation regular :: (stone-algebra) boolean-algebra
begin

lift-definition sup-regular :: 'a regular ⇒ 'a regular ⇒ 'a regular is sup
  by (meson regular-in-p-image-iff regular-closed-sup)

lift-definition inf-regular :: 'a regular ⇒ 'a regular ⇒ 'a regular is inf
  by (meson regular-in-p-image-iff regular-closed-inf)

lift-definition minus-regular :: 'a regular ⇒ 'a regular ⇒ 'a regular is λx y . x \ Intersection − y

59
by (meson regular-in-p-image-iff regular-closed-inf)

lift-definition uminus-regular :: 'a regular ⇒ 'a regular is uminus
    by auto

lift-definition bot-regular :: 'a regular is bot
    by (meson regular-in-p-image-iff regular-closed-bot)

lift-definition top-regular :: 'a regular is top
    by (meson regular-in-p-image-iff regular-closed-top)

lift-definition less-eq-regular :: 'a regular ⇒ 'a regular ⇒ bool is less-eq.

lift-definition less-regular :: 'a regular ⇒ 'a regular ⇒ bool is less.

instance
    apply intro-classes
    apply (simp add: less-eq-regular.rep-eq less-regular.rep-eq inf.le-not-le)
    apply (simp add: less-eq-regular.rep-eq)
    apply (simp add: inf-eq-less-regular.rep-eq)
    apply (simp add: inf-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: inf-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: sup-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: sup-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: sup-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: sup-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: sup-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: bot-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: bot-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: bot-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: bot-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: bot-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: top-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: top-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: top-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: top-regular.rep-eq less-eq-regular.rep-eq)
    apply (simp add: top-regular.rep-eq less-eq-regular.rep-eq)
    apply (metis (mono-tags) Rep-regular-inj inf-regular.rep-eq sup-inf-distrib1
        sup-regular.rep-eq)
    apply (metis (mono-tags) Rep-regular-inj inf-regular.rep-eq inf-regular.rep-eq
        inf-p uminus-regular.rep-eq)
    apply (metis (mono-tags) top-regular.abs-eq Rep-regular-inverse simp-regular
        stone sup-regular.rep-eq uminus-regular.rep-eq)
    by (metis (mono-tags) Rep-regular-inj inf-regular.rep-eq minus-regular.rep-eq
        uminus-regular.rep-eq)

end

instantiation regular :: (non-trivial-stone-algebra) non-trivial-boolean-algebra
begin

instance
proof (intro-classes, rule econtr)
    assume ¬(∃x y::'a regular . x ≠ y)
    hence (bot::'a regular) = top
        by simp
    hence (bot::'a) = top
5.2.2 Dense Elements

The dense elements of a Stone algebra form a distributive lattice with a greatest element.

typedef (overloaded) ’a dense = dense-elements::’a::stone-algebra set
using dense-closed-top by blast

lemma simp-dense [simp]:
−Rep-dense x = bot
using Rep-dense by simp

setup-lifting type-definition-dense

instantiation dense :: (stone-algebra) distrib-lattice-top
begin

lift-definition sup-dense :: ’a dense ⇒ ’a dense ⇒ ’a dense is sup
by simp

lift-definition inf-dense :: ’a dense ⇒ ’a dense ⇒ ’a dense is inf
by simp

lift-definition top-dense :: ’a dense is top
by simp

lift-definition less-eq-dense :: ’a dense ⇒ ’a dense ⇒ bool is less-eq .

lift-definition less-dense :: ’a dense ⇒ ’a dense ⇒ bool is less .

instance
apply intro-classes
apply (simp add: less-eq-dense.rep-eq less-dense.rep-eq inf.less-le-not-le)
apply (simp add: less-eq-dense.rep-eq)
apply (simp add: less-eq-dense.rep-eq)
apply (simp add: Rep-dense-inject less-eq-dense.rep-eq)
apply (simp add: inf-dense.rep-eq less-eq-dense.rep-eq)
apply (simp add: inf-dense.rep-eq less-eq-dense.rep-eq)
apply (simp add: inf-dense.rep-eq less-eq-dense.rep-eq)
apply (simp add: sup-dense.rep-eq less-eq-dense.rep-eq)
apply (simp add: sup-dense.rep-eq less-eq-dense.rep-eq)
apply (simp add: sup-dense.rep-eq less-eq-dense.rep-eq)
apply (simp add: top-dense.rep-eq less-eq-dense.rep-eq)
by (metis (mono-tags, lifting) Rep-dense-inject sup-inf-distrib1 inf-dense.rep-eq sup-dense.rep-eq)

end

lemma up-filter-dense-antitone-dense:
  dense \((x \sqcup -x \sqcup y)\) \(\land\) dense \((x \sqcup -x \sqcup y \sqcup z)\)
by simp

lemma up-filter-dense-antitone:
  up-filter \((Abs\text{-}dense \((x \sqcup -x \sqcup y \sqcup z)\))\) \(\leq\) up-filter \((Abs\text{-}dense \((x \sqcup -x \sqcup y)\))\)
by (unfold up-filter-antitone[THEN sym]) (simp add: Abs-dense-inverse less-eq-dense.rep-eq)

The filters of dense elements of a Stone algebra form a bounded distributive lattice.

type-synonym 'a dense-filter = 'a dense filter
typedef (overloaded) 'a dense-filter-type = \{ x::'a dense-filter . True \}
  using filter-top by blast

setup-lifting type-definition-dense-filter-type

instantiation dense-filter-type :: (stone-algebra) bounded-distrib-lattice
begin

lift-definition sup-dense-filter-type :: 'a dense-filter-type \Rightarrow 'a dense-filter-type \Rightarrow 'a dense-filter-type is sup .

lift-definition inf-dense-filter-type :: 'a dense-filter-type \Rightarrow 'a dense-filter-type \Rightarrow 'a dense-filter-type is inf .

lift-definition bot-dense-filter-type :: 'a dense-filter-type is bot ..

lift-definition top-dense-filter-type :: 'a dense-filter-type is top ..

lift-definition less-eq-dense-filter-type :: 'a dense-filter-type \Rightarrow 'a dense-filter-type \Rightarrow bool is less-eq .

lift-definition less-dense-filter-type :: 'a dense-filter-type \Rightarrow 'a dense-filter-type \Rightarrow bool is less .

instance
  apply intro-classes
  apply (simp add: less-eq-dense-filter-type.rep-eq less-dense-filter-type.rep-eq inf.less-le-not-le)
  apply (simp add: less-eq-dense-filter-type.rep-eq)
  apply (simp add: less-eq-dense-filter-type.rep-eq inf.orderless-eq-imp)
  apply (simp add: Rep-dense-filter-type-inject less-eq-dense-filter-type.rep-eq)
apply (simp add: inf-dense-filter-type.rep-eq less-eq-dense-filter-type.rep-eq)
apply (simp add: inf-dense-filter-type.rep-eq less-eq-dense-filter-type.rep-eq)
apply (simp add: inf-dense-filter-type.rep-eq less-eq-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
apply (simp add: less-eq-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)
by (metis (mono-tags, lifting) Rep-dense-filter-type-inject sup-inf-distrib1 inf-dense-filter-type.rep-eq sup-dense-filter-type.rep-eq)

end

5.2.3 The Structure Map

The structure map of a Stone algebra is a bounded lattice homomorphism. It maps a regular element \( x \) to the set of all dense elements above \(-x\). This set is a filter.

abbreviation stone-phi-set :: 'a::stone-algebra regular ⇒ 'a dense set
where stone-phi-set x ≡ \{ y . Rep-regular x ≤ Rep-dense y \}

lemma stone-phi-set-filter:
filter (stone-phi-set x)
apply (unfold filter-def, intro conjI)
apply (metis Collect-empty-eq top-dense.rep-eq top-greatest)
apply (metis inf-dense.rep-eq inf-le2 le-inf-iff mem-Collect-eq)
using order-trans less-eq-dense.rep-eq by blast

definition stone-phi :: 'a::stone-algebra regular ⇒ 'a dense-filter
where stone-phi x = Abs-filter (stone-phi-set x)

To show that we obtain a triple, we only need to prove that stone-phi is a bounded lattice homomorphism. The Boolean algebra and the distributive lattice requirements are taken care of by the type system.

interpretation stone-phi: triple stone-phi
proof (unfold-locales, intro conjI)
  have 1: Rep-regular (Abs-regular bot) = bot
    by (metis bot-regular.rep-eq bot-regular-def)
  show stone-phi bot = bot
    apply (unfold stone-phi-def bot-regular-def 1 p-bot bot-filter-def)
    by (metis (mono-tags, lifting) Collect-cong Rep-dense-inject order-refl singleton-conv top.extremum-uniqueI top-dense.rep-eq)
next
  show stone-phi top = top
next
  show \( \forall x y::'a \text{ regular} . \text{stone-phi} (x \sqcup y) = \text{stone-phi} x \sqcup \text{stone-phi} y \)
proof (intro allI)
fix x y :: 'a regular
have stone-phi-set (x ⊔ y) = filter-sup (stone-phi-set x) (stone-phi-set y)
proof (rule set-eqI, rule iffI)
fix z
assume 2: z ∈ stone-phi-set (x ⊔ y)
let ℓt = ¬Rep-regular x ⊔ Rep-dense z
let ℓu = ¬Rep-regular y ⊔ Rep-dense z
let ℓv = Abs-dense ℓt
let ℓw = Abs-dense ℓu
have 3: ℓv ∈ stone-phi-set x ∧ ℓw ∈ stone-phi-set y
  by (simp add: Abs-dense-inverse)
have ℓv ⊓ ℓw = Abs-dense (ℓt ⊓ ℓu)
  by (simp add: eq-onp-def inf-dense.abs-eq)
also have ... = Abs-dense (¬Rep-regular (x ⊔ y) ⊔ Rep-dense z)
  by (simp add: distrib(1) sup-commute sup-regular.rep-eq)
also have ... = Abs-dense (Rep-dense z)
  using 2 by (simp add: le_iff-sup)
also have ... = z
  by (simp add: Abs-dense-inverse)
finally show z ∈ filter-sup (stone-phi-set x) (stone-phi-set y)
  using 3 mem-Collect-eq order-refl by fastforce
next
fix z
assume z ∈ filter-sup (stone-phi-set x) (stone-phi-set y)
then obtain v w where 4: v ∈ stone-phi-set x ∧ w ∈ stone-phi-set y ∧ v ⊓ w ≤ z
  by auto
have ¬Rep-regular (x ⊔ y) = Rep-regular (¬(x ⊔ y))
  by (metis uminus-regular.rep-eq)
also have ... = ¬Rep-regular x ⊓ ¬Rep-regular y
  by (simp add: inf-regular.rep-eq uminus-regular.rep-eq)
also have ... ≤ Rep-dense v ⊓ Rep-dense w
  using 4 inf-mono mem-Collect-eq by blast
also have ... = Rep-dense (v ⊓ w)
  by (simp add: inf-dense.rep-eq)
also have ... ≤ Rep-dense z
  using 4 by (simp add: less-eq-dense.rep-eq)
finally show z ∈ stone-phi-set (x ⊔ y)
  by simp
qed
thus stone-phi (x ⊔ y) = stone-phi x ⊔ stone-phi y
  by (simp add: stone-phi-def eq-onp-same-args stone-phi-set-filter sup-filter.abs-eq)
qed
next
show ∀ x y :: 'a regular . stone-phi (x ⊓ y) = stone-phi x ⊓ stone-phi y
proof (intro allI)
fix x y :: 'a regular
have \( \forall z \cdot \neg \text{Rep-regular} (x \land y) \leq \text{Rep-dense} z \iff \neg \text{Rep-regular} x \leq \text{Rep-dense} z \land \neg \text{Rep-regular} y \leq \text{Rep-dense} z \)

by (simp add: inf-regular, rep-eq)

hence \( \text{stone-phi-set} (x \land y) = (\text{stone-phi-set} x) \cap (\text{stone-phi-set} y) \)

by auto

thus \( \text{stone-phi} (x \land y) = \text{stone-phi} x \cap \text{stone-phi} y \)

by (simp add: stone-phi-def eq-omp-same-args stone-phi-set-filter inf-filter.abs-eq)

qed

5.3 Properties of Triples

In this section we construct a certain set of pairs from a triple, introduce operations on these pairs and develop their properties. The given set and operations will form a Stone algebra.

codecontext triple
begin

lemma phi-bot:
\( \phi \bot = \text{Abs-filter} \{\top\} \)
by (metis hom bot-filter-def)

lemma phi-top:
\( \phi \top = \text{Abs-filter} \text{UNIV} \)
by (metis hom top-filter-def)

The occurrence of \( \phi \) in the following definition of the pairs creates a need for dependent types.

definition pairs :: \((a \times b \text{filter}) \text{set}\)
where pairs = \(\{ (x,y) \cdot \exists z \cdot y = \phi (\neg x) \sqcup \text{up-filter} z \}\)

Operations on pairs are defined in the following. They will be used to establish that the pairs form a Stone algebra.

fun pairs-less-eq :: \((a \times b \text{filter}) \Rightarrow (a \times b \text{filter}) \Rightarrow \text{bool}\)
where pairs-less-eq \((x,y) (z,w) = (x \leq z \land w \leq y)\)

fun pairs-less :: \((a \times b \text{filter}) \Rightarrow (a \times b \text{filter}) \Rightarrow \text{bool}\)
where pairs-less \((x,y) (z,w) = \{\text{pairs-less-eq} (x,y) (z,w) \land \neg \text{pairs-less-eq} (z,w) (x,y)\}\)

fun pairs-sup :: \((a \times b \text{filter}) \Rightarrow (a \times b \text{filter}) \Rightarrow (a \times b \text{filter})\)
where pairs-sup \((x,y) (z,w) = (x \sqcup z,y \sqcap w)\)

fun pairs-inf :: \((a \times b \text{filter}) \Rightarrow (a \times b \text{filter}) \Rightarrow (a \times b \text{filter})\)
where pairs-inf \((x,y) (z,w) = (x \sqcap z,y \sqcup w)\)

fun pairs-uminus :: \((a \times b \text{filter}) \Rightarrow (a \times b \text{filter})\)

where \( \text{pairs-uminus} \( (x, y) = (\neg x, \phi x) \)

abbreviation \( \text{pairs-bot} :: (\alpha \times \beta \text{ filter}) \)
where \( \text{pairs-bot} \equiv (\bot, \text{Abs-filter UNIV}) \)

abbreviation \( \text{pairs-top} :: (\alpha \times \beta \text{ filter}) \)
where \( \text{pairs-top} \equiv (\top, \text{Abs-filter \{top\}}) \)

lemma \( \text{pairs-top-in-set:} \)
\((x, y) \in \text{pairs} \implies \top \in \text{Rep-filter y} \)
by simp

lemma \( \phi\text{-complemented:} \)
complement \((\phi x (\phi (\neg x))) \)
by (metis hom inf-compl-bot sup-compl-top)

lemma \( \phi\text{-inf-principal:} \)
\( \exists z . \up-filter z = \phi x \cap \up-filter y \)
proof –
let \( ?F = \text{Rep-filter} (\phi x) \)
let \( ?G = \text{Rep-filter} (\phi (\neg x)) \)
have 1: \( \text{eq-onp filter} ?F ?F \land \text{eq-onp filter} (\up y) (\up y) \)
by (simp add: eq-onp-def)
have \( \text{filter-complements} ?F ?G \)
apply (intro conjI)
apply simp
apply simp
apply (metis (no-types) phi-complemented sup-filter.rep-eq top-filter.rep-eq)
by (metis (no-types) phi-complemented inf-filter.rep-eq bot-filter.rep-eq)
hence \( \text{is-principal-up} (?F \cap \up y) \)
using complemented-filter-inf-principal by blast
then obtain \( z \) where \( \up z = ?F \cap \up y \)
by auto
hence \( \up-filter z = \text{Abs-filter} (?F \cap \up y) \)
by simp
also have \( ... = \text{Abs-filter} ?F \cap \up-filter y \)
using 1 inf-filter.abs-eq by force
also have \( ... = \phi x \cap \up-filter y \)
by (simp add: Rep-filter-inverse)
finally show \( \text{thesis} \)
by auto
qed

Quite a bit of filter theory is involved in showing that the intersection of \( \phi x \) with a principal filter is a principal filter, so the following function can extract its least element.

fun \( \rho :: \alpha = \beta \Rightarrow \beta \)
where \( \rho x y = (\text{SOME} z . \up-filter z = \phi x \cap \up-filter y) \)
lemma rho-char:
  \text{up-filter} (\rho x y) = \phi x \cap \text{up-filter} y
by \text{(metis (mono-tags) someI-ex rho.simps phi-inf-principal)}

The following results show that the pairs are closed under the given operations.

lemma pairs-sup-closed:
assumes \((x,y) \in \text{pairs}\)
and \((z,w) \in \text{pairs}\)
shows \(\text{pairs-sup} (x,y) (z,w) \in \text{pairs}\)
proof
  from \text{assms} obtain \(u v\) where \(y = \phi (\neg x) \cup \text{up-filter} u \land w = \phi (\neg z) \cup \text{up-filter} v\)
using \text{pairs-def} by auto
hence \(\text{pairs-sup} (x,y) (z,w) = (x \cup z,(\phi (\neg x) \cup \text{up-filter} u) \cap (\phi (\neg z) \cup \text{up-filter} v))\)
by simp
also have \... = \(x \cup z,(\phi (\neg x) \cap \phi (\neg z)) \cup (\phi (\neg x) \cap \text{up-filter} v) \cup (\text{up-filter} u \cap \phi (\neg z)) \cup (\text{up-filter} u \cap \text{up-filter} v)\)
by \text{(simp add: sup-commute inf-sup-distrib1 sup-commute sup-left-commute)}
also have \... = \(x \cup z,\phi (\neg (x \cup z)) \cup (\phi (\neg x) \cap \text{up-filter} v) \cup (\text{up-filter} u \cap \phi (\neg z)) \cup (\text{up-filter} u \cap \text{up-filter} v)\)
using \text{hom} by simp
also have \... = \(x \cup z,\phi (\neg (x \cup z)) \cup \text{up-filter} (\rho (\neg x) v) \cup \text{up-filter} (\rho (\neg z) u) \cup (\text{up-filter} u \cap \text{up-filter} v)\)
by \text{(metis inf-sup-commute rho-char)}
also have \... = \(x \cup z,\phi (\neg (x \cup z)) \cup \text{up-filter} (\rho (\neg x) v) \cup \text{up-filter} (\rho (\neg z) u) \cup \text{up-filter} (u \cup v)\)
by \text{(metis up-filter-dist-sup)}
also have \... = \(x \cup z,\phi (\neg (x \cup z)) \cup \text{up-filter} (\rho (\neg x) v \cap \rho (\neg z) u \cap (u \cup v))\)
by \text{(simp add: sup-commute sup-left-commute up-filter-dist-inf)}
finally show \(?\text{thesis}\)
using \text{pairs-def} by auto
qed

lemma pairs-inf-closed:
assumes \((x,y) \in \text{pairs}\)
and \((z,w) \in \text{pairs}\)
shows \(\text{pairs-inf} (x,y) (z,w) \in \text{pairs}\)
proof
  from \text{assms} obtain \(u v\) where \(y = \phi (\neg x) \cup \text{up-filter} u \land w = \phi (\neg z) \cup \text{up-filter} v\)
using \text{pairs-def} by auto
hence \(\text{pairs-inf} (x,y) (z,w) = (x \cap z,(\phi (\neg x) \cup \text{up-filter} u) \cap (\phi (\neg z) \cup \text{up-filter} v))\)
by simp
also have \... = \(x \cap z,(\phi (\neg x) \cup \phi (\neg z)) \cup (\text{up-filter} u \cup \text{up-filter} v)\)
by (simp add: sup-commute sup-left-commute)
also have \( \ldots = (x \cap z, \phi (- (x \cap z)) \sqcup (\text{up-filter } u \sqcup \text{up-filter } v)) \)
using hom by simp
also have \( \ldots = (x \cap z, \phi (- (x \cap z)) \sqcup \text{up-filter } (u \cap v)) \)
by (simp add: up-filter-dist-inf)
finally show \(?thesis
using pairs-def by auto
qed

**lemma** pairs-uminus-closed:
\( \text{pairs-uminus } (x,y) \in \text{pairs} \)
**proof** –
\( \text{have } \text{pairs-uminus } (x,y) = (-x, \phi (-x) \sqcup \text{bot}) \)
by simp
also have \( \ldots = (-x, \phi (-x) \sqcup \text{up-filter top}) \)
by (simp add: bot-filter.abs-eq)
finally show \(?thesis
by (metis (mono-tags, lifting) mem-Collect-eq old.prod.case pairs-def)
qed

**lemma** pairs-bot-closed:
\( \text{pairs-bot} \in \text{pairs} \)
using pairs-def phi-top triple.hom triple-axioms by fastforce

**lemma** pairs-top-closed:
\( \text{pairs-top} \in \text{pairs} \)
by (metis p-bot pairs-uminus.simps pairs-uminus-closed phi-bot)

We prove enough properties of the pair operations so that we can later show they form a Stone algebra.

**lemma** pairs-sup-dist-inf:
\( (x,y) \in \text{pairs} \Longrightarrow (z,w) \in \text{pairs} \Longrightarrow (u,v) \in \text{pairs} \Longrightarrow \text{pairs-sup } (x,y) \text{ (pairs-inf } (z,w) \text{ (u,v) } ) = \text{pairs-inf } (\text{pairs-sup } (x,y) \text{ (z,w) } ) \text{ (pairs-sup } (x,y) \text{ (u,v) } ) \)
using sup-inf-distrib1 inf-sup-distrib1 by auto

**lemma** pairs-phi-less-eq:
\( (x,y) \in \text{pairs} \Longrightarrow \phi (-x) \leq y \)
using pairs-def by auto

**lemma** pairs-uminus-galois:
\( \text{assumes } (x,y) \in \text{pairs} \quad \text{and } (z,w) \in \text{pairs} \)
\( \text{shows } \text{pairs-inf } (x,y) (z,w) = \text{pairs-bot } \Longleftrightarrow \text{pairs-less-eq } (x,y) \text{ (pairs-uminus } (z,w) ) \)
**proof** –
\( \text{have } 1: x \cap z = \text{bot } \wedge y \sqcup w = \text{Abs-filter } \text{UNIV } \Longrightarrow \phi z \leq y \)
by (metis (no-types, lifting) assms(1) heyting.implies-inf-absorb hom le-supE pairs-phi-less-eq sup-bot-right)
\( \text{have } 2: x \leq -z \land \phi z \leq y \Longrightarrow y \sqcup w = \text{Abs-filter } \text{UNIV} \)
proof
  assume 3: $x \leq -z \land \phi z \leq y$
  have Abs-filter $\text{UNIV} = \phi z \sqcup \phi (-z)$
    using hom phi-complemented phi-top by auto
  also have ... $\leq y \sqcup w$
    using 3 assms(2) sup-mono pairs-phi-less-eq by auto
  finally show $y \sqcup w = \text{Abs-filter UNIV}$
    using hom phi-top top.extremum-uniquel by auto

qed

have $x \sqcap z = \text{bot} \iff x \leq -z$
  by (simp add: shunting-1)
thus ?thesis
  using 1 2 Pair-inject pairs-inf.simps pairs-less-eq.simps pairs-uminus.simps
    by auto

qed

lemma pairs-stone:
  $(x,y) \in \text{pairs} \implies \text{pairs-sup (pairs-uminus (x,y)) (pairs-uminus (pairs-uminus (x,y))) = pairs-top}$
  by (metis hom pairs-sup.simps pairs-uminus.simps phi-bot phi-complemented stone)

  The following results show how the regular elements and the dense elements
  among the pairs look like.

abbreviation dense-pairs $\equiv \{ (x,y) . (x,y) \in \text{pairs} \land \text{pairs-uminus (x,y)} = \text{pairs-bot} \}$
abbreviation regular-pairs $\equiv \{ (x,y) . (x,y) \in \text{pairs} \land \text{pairs-uminus (x,y)} = (x,y) \}$
abbreviation is-principal-up-filter $x \equiv \exists y . x = \text{up-filter y}$

lemma dense-pairs:
  dense-pairs $= \{ (x,y) . x = \text{top} \land \text{is-principal-up-filter y} \}$
proof
  have dense-pairs $= \{ (x,y) . (x,y) \in \text{pairs} \land x = \text{top} \}$
    by (metis Pair-inject compl-bot-eq double-compl pairs-uminus.simps phi-top)
  also have ... $= \{ (x,y) . (\exists z . y = \text{up-filter z}) \land x = \text{top} \}$
    using hom pairs-def by auto
  finally show ?thesis
    by auto
qed

lemma regular-pairs:
  regular-pairs $= \{ (x,y) . y = \phi (-x) \}$
  using pairs-def pairs-uminus-closed by fastforce

  The following extraction function will be used in defining one direction
  of the Stone algebra isomorphism.

fun rho-pair :: 'a + 'b filter $\Rightarrow$ 'b
where rho-pair $(x,y) = (\text{SOME z . up-filter z = phi x} \sqcap y)$
lemma get-rho-pair-char:
assumes \((x,y) \in \text{pairs}\)
shows \(\text{up-filter} (\rho\text{-pair} (x,y)) = \phi x \cap y\)

proof
from assms obtain \(w\) where \(y = \phi (-x) \sqcup \text{up-filter} w\)
using pairs-def by auto
hence \(\phi x \cap y = \phi x \cap \text{up-filter} w\)
by (simp add: inf-sup-distrib1 phi-complemented)
thus ?thesis
using rho-char by auto
qed

lemma sa-iso-pair:
\((-x,\phi (-x) \sqcup \text{up-filter} y) \in \text{pairs}\)
using pairs-def by auto

end

5.4 The Stone Algebra of a Triple

In this section we prove that the set of pairs constructed in a triple forms a Stone Algebra. The following type captures the parameter \(\phi\) on which the type of triples depends. This parameter is the structure map that occurs in the definition of the set of pairs. The set of all structure maps is the set of all bounded lattice homomorphisms (of appropriate type). In order to make it a HOL type, we need to show that at least one such structure map exists. To this end we use the ultrafilter lemma: the required bounded lattice homomorphism is essentially the characteristic map of an ultrafilter, but the latter must exist. In particular, the underlying Boolean algebra must contain at least two elements.

typedef (overloaded) \((\cdot a,\cdot b)\) \(\phi = \{ f : \cdot a::\text{non-trivial-boolean-algebra} \Rightarrow \cdot b::\text{distrib-lattice-top filter} . \text{bounded-lattice-homomorphism} f \} \)

proof
from ultra-filter-exists obtain \(F : \cdot a \text{ set} \) where \(1: \text{ ultra-filter} F\)
by auto
hence \(2: \text{ prime-filter} F\)
using ultra-filter-prime by auto
let \(\exists f = \lambda x . \text{ if } x \in F \text{ then top else bot} : \cdot b \text{ filter}\)
have bounded-lattice-homomorphism ?f
proof (intro conjI)
show \(?f \text{ bot} = \text{ bot}\)
using \(I\) by (meson bot.extremum filter-def subset-eq top.extremum-unique)
next
show \(?f \text{ top} = \text{ top}\)
using \(I\) by simp
next
show \(\forall x y . \exists f \ (x \sqcup y) = \?f \sqcup \?f y\)
proof (intro allI)
  fix x y
  show \( ?f (x \sqcup y) = ?f x \sqcup ?f y \)
  apply (cases \( x \in F \); cases \( y \in F \))
  using 1 filter-def apply fastforce
  using 1 filter-def apply fastforce
  using 1 filter-def apply fastforce
  using 2 sup-bot-left by auto
qed

next
  show \( \forall x y. ?f (x \sqcap y) = ?f x \sqcap ?f y \)
proof (intro allI)
  fix x y
  show \( ?f (x \sqcap y) = ?f x \sqcap ?f y \)
  apply (cases \( x \in F \); cases \( y \in F \))
  using 1 apply (simp add: filter-inf-closed)
  using 1 apply (metis (mono-tags, lifting) brouwer.inf-sup-ord(4) inf-top-left filter-def)
  using 1 apply (metis (mono-tags, lifting) brouwer.inf-sup-ord(3) inf-top-right filter-def)
  using 1 filter-def by force
qed

hence \( ?f \in \{f . \text{bounded-lattice-homomorphism } f\} \)
  by simp
thus \( \vdash \text{thesis} \)
  by meson
qed

lemma simp-phi [simp]:
  bounded-lattice-homomorphism (Rep-phi x)
  using Rep-phi by simp

setup-lifting type-definition-phi

The following implements the dependent type of pairs depending on structure maps. It uses functions from structure maps to pairs with the requirement that, for each structure map, the corresponding pair is contained in the set of pairs constructed for a triple with that structure map.

If this type could be defined in the locale \textit{triple} and instantiated to Stone algebras there, there would be no need for the lifting and we could work with triples directly.

typedef (overloaded) ('a,'b) lifted-pair = {
  pf::('a::non-trivial-boolean-algebra,'b::distrib-lattice-top) phi ⇒ 'a × 'b filter . \forall f . pf f ∈ triple.pairs (Rep-phi f) }
proof –
  have \( \forall f::('a,'b) phi . \text{triple.pairs-bot} \in \text{triple.pairs} (Rep-phi f) \)
  proof
    fix f :: ('a,'b) phi
have triple (Rep-\(\phi f\))
by (simp add: triple-def)
thus triple.pairs-bot \(\in\) triple.pairs (Rep-\(\phi f\))
using triple.regular-pairs triple.phi-top by fastforce
qed
thus \(?\)thesis
by auto
qed

lemma simp-lifted-pair [simp]:
\(\forall f\) . Rep-lifted-pair \(pf f\) \(\in\) triple.pairs (Rep-\(\phi f\))
using Rep-lifted-pair by simp

setup-lifting type-definition-lifted-pair

The lifted pairs form a Stone algebra.

instantiation lifted-pair :: (non-trivial-boolean-algebra, distrib-lattice-top)
stone-algebra
begin

All operations are lifted point-wise.

lift-definition sup-lifted-pair :: ('a', 'b) lifted-pair \(\Rightarrow\) ('a', 'b) lifted-pair \(\Rightarrow\) ('a', 'b) lifted-pair is \(\lambda f yf f\) . triple.pairs-sup (xf f) (yf f)
by (metis (no-types, hide-lams) simp-phi triple-def triple.pairs-sup-closed prod.collapse)

lift-definition inf-lifted-pair :: ('a', 'b) lifted-pair \(\Rightarrow\) ('a', 'b) lifted-pair \(\Rightarrow\) ('a', 'b) lifted-pair is \(\lambda f yf f\) . triple.pairs-inf (xf f) (yf f)
by (metis (no-types, hide-lams) simp-phi triple-def triple.pairs-inf-closed prod.collapse)

lift-definition uminus-lifted-pair :: ('a', 'b) lifted-pair \(\Rightarrow\) ('a', 'b) lifted-pair \(\Rightarrow\) ('a', 'b) lifted-pair is \(\lambda f yf f\) . triple.pairs-uminus (Rep-\(\phi f\)) (xf f)
by (metis (no-types, hide-lams) simp-phi triple-def triple.pairs-uminus-closed prod.collapse)

lift-definition bot-lifted-pair :: ('a', 'b) lifted-pair \(\Rightarrow\) ('a', 'b) lifted-pair is \(\lambda f\) . triple.pairs-bot
by (metis (no-types, hide-lams) simp-phi triple-def triple.pairs-bot-closed)

lift-definition top-lifted-pair :: ('a', 'b) lifted-pair \(\Rightarrow\) ('a', 'b) lifted-pair is \(\lambda f\) . triple.pairs-top
by (metis (no-types, hide-lams) simp-phi triple-def triple.pairs-top-closed)

lift-definition less-eq-lifted-pair :: ('a', 'b) lifted-pair \(\Rightarrow\) ('a', 'b) lifted-pair \(\Rightarrow\) bool
is \(\lambda f yf f\) . \(\forall f\) . triple.pairs-less-eq (xf f) (yf f).

lift-definition less-lifted-pair :: ('a', 'b) lifted-pair \(\Rightarrow\) ('a', 'b) lifted-pair \(\Rightarrow\) bool is \(\lambda f yf f\) . \(\forall f\) . triple.pairs-less (xf f) (yf f) \&(\forall f\) . triple.pairs-less-eq (yf f) (xf f)).
instance
proof intro-classes
  fix \( xf, yf :: ('a, 'b) \) lifted-pair
  show \( xf < yf \iff xf \leq yf \land \neg yf \leq xf \)
    by (simp add: less-lifted-pair.rep-eq less-eq-lifted-pair.rep-eq)
next
  fix \( xf :: ('a, 'b) \) lifted-pair
  { fix \( f :: ('a, 'b) \) phi 
    have 1: triple (Rep-phi \( f \)) 
      by (simp add: triple-def) 
    let \( ?x = \) Rep-lifted-pair \( xf \) \( f \) 
    obtain \( x1, x2 \) where \( (x1, x2) = ?x \) 
      using prod.collapse by blast 
    hence triple.pairs-less-eq \( ?x \) \( ?x \) 
      using 1 by (metis triple.pairs-less-eq.simps order-refl) 
    } hence \( xf \leq xf \) 
      by (simp add: less-eq-lifted-pair.rep-eq)
next
  fix \( xf, yf, zf :: ('a, 'b) \) lifted-pair
  assume 1: \( xf \leq yf \) and 2: \( yf \leq zf \)
  { fix \( f :: (a, 'b) \) phi 
    have 3: triple (Rep-phi \( f \)) 
      by (simp add: triple-def) 
    let \( ?x = \) Rep-lifted-pair \( xf \) \( f \) 
    let \( ?y = \) Rep-lifted-pair \( yf \) \( f \) 
    let \( ?z = \) Rep-lifted-pair \( zf \) \( f \) 
    obtain \( x1, x2, y1, y2, z1, z2 \) where 4: \( (x1, x2) = ?x \land (y1, y2) = ?y \land (z1, z2) = ?z \) 
      using prod.collapse by blast 
    have triple.pairs-less-eq \( ?x \) \( ?y \land \) triple.pairs-less-eq \( ?y \) \( ?z \) 
      using 1 2 3 less-eq-lifted-pair.rep-eq by simp 
    hence triple.pairs-less-eq \( ?x \) \( ?z \) 
      using 3 4 by (metis (mono-tags, lifting) triple.pairs-less-eq.simps order-trans) 
    } hence \( xf \leq zf \) 
      by (simp add: less-eq-lifted-pair.rep-eq)
next
  fix \( xf, yf :: (a, 'b) \) lifted-pair
  assume 1: \( xf \leq yf \) and 2: \( yf \leq xf \)
  { fix \( f :: (a, 'b) \) phi 
    have 3: triple (Rep-phi \( f \)) 
      by (simp add: triple-def) 
    let \( ?x = \) Rep-lifted-pair \( xf \) \( f \) 
    let \( ?y = \) Rep-lifted-pair \( yf \) \( f \)
obtain \(x_1\ x_2\ y_1\ y_2\) where \(x_1 x_2 = \exists x\ y_1 y_2 = \exists y\) using \text{prod.collapse} by blast

have \text{triple.pairs-less-eq} ?x ?y \land \text{triple.pairs-less-eq} ?y ?x using 1 2 3 \text{less-eq-lifted-pair.rep-eq} by simp

hence \(?x = ?y\) using 3 4 by (metis (mono-tags, lifting) \text{triple.pairs-less-eq} \text{simps} antisym)

thus \(xf = yf\) by (metis \text{Rep-lifted-pair-inverse} ext)

next
fix \(xf\ yf\) :: \('a', 'b\) lifted-pair

\{ fix \(f\) :: \('a', 'b\) \text{phi} have 1: \text{triple} \text{(Rep-phi} f) by (simp add: \text{triple-def})

let \(?x = \text{Rep-lifted-pair} xf\ f\) let \(?y = \text{Rep-lifted-pair} yf\ f\)

obtain \(x_1\ x_2\ y_1\ y_2\) where \((x_1,x_2) = \exists x\ (y_1,y_2) = \exists y\)

using \text{prod.collapse} by blast

hence \text{triple.pairs-less-eq} \text{(triple.pairs-inf} ?x ?y) ?y using 1 by (metis (mono-tags, lifting) \text{inf-sup-ord} \text{sup.cobounded2} \text{triple.pairs-inf} \text{simps} \text{triple.pairs-less-eq} \text{inf-lifted-pair.rep-eq})

\} thus \(xf \sqcap yf \leq yf\) by (simp add: \text{less-eq-lifted-pair.rep-eq} \text{inf-lifted-pair.rep-eq})

next
fix \(xf\ yf\) :: \('a', 'b\) lifted-pair

\{ fix \(f\) :: \('a', 'b\) \text{phi} have 1: \text{triple} \text{(Rep-phi} f) by (simp add: \text{triple-def})

let \(?x = \text{Rep-lifted-pair} xf\ f\) let \(?y = \text{Rep-lifted-pair} yf\ f\)

obtain \(x_1\ x_2\ y_1\ y_2\) where \((x_1,x_2) = \exists x\ (y_1,y_2) = \exists y\)

using \text{prod.collapse} by blast

hence \text{triple.pairs-less-eq} \text{(triple.pairs-inf} ?x ?y) ?x using 1 by (metis (mono-tags, lifting) \text{inf-sup-ord} \text{sup.cobounded1} \text{triple.pairs-inf} \text{simps} \text{triple.pairs-less-eq} \text{inf-lifted-pair.rep-eq})

\} thus \(xf \sqcap yf \leq xf\) by (simp add: \text{less-eq-lifted-pair.rep-eq} \text{inf-lifted-pair.rep-eq})

next
fix \(xf\ yf\ zf\) :: \('a', 'b\) lifted-pair

assume 1: \(xf \leq yf\) and 2: \(xf \leq zf\)

\{ fix \(f\) :: \('a', 'b\) \text{phi} have 3: \text{triple} \text{(Rep-phi} f) by (simp add: \text{triple-def})

let \(?x = \text{Rep-lifted-pair} xf\ f\)
let \( \text{?y} = \text{Rep-lifted-pair yf f} \)
let \( \text{?z} = \text{Rep-lifted-pair zf f} \)

obtain \( x1 \ x2 \ y1 \ y2 \ z1 \ z2 \) where 4: \((x1, x2) = \text{?x} \land (y1, y2) = \text{?y} \land (z1, z2) = \text{?z}\)

using prod-collapse by blast

have triple.pairs-less-eq \?x \ ?y \land triple.pairs-less-eq \?x \ ?z
using 1 2 3 less-eq-lifted-pair.rep-eq by simp

hence triple.pairs-less-eq \?x (triple.pairs-inf \text{?y} \ ?z)
using 3 4 by (metis (mono-tags, lifting) le-inf-iff sup.bounded-iff triple.pairs-inf.simps triple.pairs-less-eq.simps)

thus \( xf \leq yf \sqcup zf \)
by (simp add: less-eq-lifted-pair.rep-eq inf-lifted-pair.rep-eq)

next
fix \( xf \ yf :: (\text{'a}', \text{'b}) \text{ lifted-pair} \)

fix \( f :: (\text{'a}', \text{'b}) \phi f \)
have 1: triple (Rep-phi f)
by (simp add: triple-def)

let \( \text{?x} = \text{Rep-lifted-pair xf f} \)
let \( \text{?y} = \text{Rep-lifted-pair yf f} \)

obtain \( x1 \ x2 \ y1 \ y2 \) where \((x1, x2) = \text{?x} \land (y1, y2) = \text{?y} \)

using prod-collapse by blast

hence triple.pairs-less-eq ?y (triple.pairs-sup ?x ?y)
using 1 by (metis (no-types, lifting) inf-commute sup.cobounded1 inf.cobounded2 triple.pairs-sup.simps triple.pairs-less-eq.simps sup-lifted-pair.rep-eq)

thus \( xf \leq xf \sqcup yf \)
by (simp add: less-eq-lifted-pair.rep-eq sup-lifted-pair.rep-eq)

next
fix \( xf \ yf :: (\text{'a}', \text{'b}) \text{ lifted-pair} \)

fix \( f :: (\text{'a}', \text{'b}) \phi f \)
have 1: triple (Rep-phi f)
by (simp add: triple-def)

let \( \text{?x} = \text{Rep-lifted-pair xf f} \)
let \( \text{?y} = \text{Rep-lifted-pair yf f} \)

obtain \( x1 \ x2 \ y1 \ y2 \) where \((x1, x2) = \text{?x} \land (y1, y2) = \text{?y} \)

using prod-collapse by blast

hence triple.pairs-less-eq ?y (triple.pairs-sup ?x ?y)
using 1 by (metis (no-types, lifting) sup.cobounded2 inf.cobounded2 triple.pairs-sup.simps triple.pairs-less-eq.simps sup-lifted-pair.rep-eq)

thus \( yf \leq xf \sqcup yf \)
by (simp add: less-eq-lifted-pair.rep-eq sup-lifted-pair.rep-eq)

next
fix \( xf \ yf :: (\text{'a}', \text{'b}) \text{ lifted-pair} \)
assume 1: \( yf \leq xf \) and 2: \( zf \leq xf \)
{ 
  fix \( f :: ('a,'b) \text{phi} \)
  have 3: triple \((\text{Rep-phi } f)\) 
    by (simp add: triple-def)
  let \( ?x = \text{Rep-lifted-pair } xf \)
  let \( ?y = \text{Rep-lifted-pair } yf \)
  let \( ?z = \text{Rep-lifted-pair } zf \)
  obtain \( x1 \ x2 \ y1 \ y2 \ z1 \ z2 \) \text{where} 4: \( (x1,x2) = ?x \land (y1,y2) = ?y \land (z1,z2) = ?z \)
  using prod-collapse by blast
  have triple.pairs-less-eq \(?y \land \text{triple.pairs-less-eq } ?x \)
    using 1 2 3 less-eq-lifted-pair.rep-eq by simp
  hence triple.pairs-less-eq \((\text{triple.pairs-sup } ?y \land ?z) \)
    using 3 4 by (metis (mono-tags, lifting) le-inf-iff sup.bounded-iff 
      triple.pairs-sup.simps triple.pairs-less-eq.simps)
  thus \( yf \sqcup zf \le xf \)
    by (simp add: less-eq-lifted-pair.rep-eq sup-lifted-pair.rep-eq)
}

next
fix \( xf :: ('a,'b) \text{lifted-pair} \)
{ 
  fix \( f :: ('a,'b) \text{phi} \)
  have 1: triple \((\text{Rep-phi } f)\) 
    by (simp add: triple-def)
  let \( ?x = \text{Rep-lifted-pair } xf \)
  obtain \( x1 \ x2 \) \text{where} 4: \( (x1,x2) = ?x \)
    using prod-collapse by blast
  hence triple.pairs-less-eq \((\text{triple.pairs-bot } ?x) \)
    using 1 by (metis bot.extremum top-greatest top-filter.abs-eq 
      triple.pairs-less-eq.simps)
  thus \( \text{bot} \le xf \)
    by (simp add: less-eq-lifted-pair.rep-eq bot-lifted-pair.rep-eq)
}

next
fix \( xf :: ('a,'b) \text{lifted-pair} \)
{ 
  fix \( f :: ('a,'b) \text{phi} \)
  have 1: triple \((\text{Rep-phi } f)\) 
    by (simp add: triple-def)
  let \( ?x = \text{Rep-lifted-pair } xf \)
  obtain \( x1 \ x2 \) \text{where} 4: \( (x1,x2) = ?x \)
    using prod-collapse by blast
  hence triple.pairs-less-eq \(?x \text{triple.pairs-top} \)
    using 1 by (metis top.extremum bot-least bot-filter.abs-eq 
      triple.pairs-less-eq.simps)
  thus \( xf \le \text{top} \)
    by (simp add: less-eq-lifted-pair.rep-eq top-lifted-pair.rep-eq)
}

next

76
fix xf yf zf :: ('a,'b) lifted-pair
{
  fix f :: ('a,'b) phi
  have 1: triple (Rep-phi f)
      by (simp add: triple-def)
  let ?x = Rep-lifted-pair xf f
  let ?y = Rep-lifted-pair yf f
  let ?z = Rep-lifted-pair zf f
  obtain x1 x2 y1 y2 z1 z2 where (x1,x2) = ?x ∧ (y1,y2) = ?y ∧ (z1,z2) = ?z
    using prod.collapse by blast
  hence triple.pairs-sup ?x (triple.pairs-inf ?y ?z) = triple.pairs-inf
      (triple.pairs-sup ?x ?y) (triple.pairs-sup ?x ?z)
    using 1 by (metis (no-types) sup-inf-distrib1 inf-sup-distrib1 triple.pairs-sup.simps triple.pairs-inf.simps)
}
thus xf ⊔ (yf ⊓ zf) = (xf ⊔ yf) ⊓ (xf ⊔ zf)
  by (metis Rep-lifted-pair-inverse ext sup-lifted-pair rep-eq inf-lifted-pair rep-eq)
next
fix xf yf :: ('a,'b) lifted-pair
{
  fix f :: ('a,'b) phi
  have 1: triple (Rep-phi f)
      by (simp add: triple-def)
  let ?x = Rep-lifted-pair xf f
  obtain x1 x2 y1 y2 where 2: (x1,x2) = ?x ∧ (y1,y2) = ?y
    using prod.collapse by blast
  have ?x ∈ triple.pairs (Rep-phi f) ∧ ?y ∈ triple.pairs (Rep-phi f)
    by simp
  hence (triple.pairs-inf ?x ?y = triple.pairs-bot) ←→ triple.pairs-less-eq ?x
      (triple.pairs-uminus (Rep-phi f) ?y)
    using 1 2 by (metis triple.pairs-uminus-galois)
}
hence ∀ f . (Rep-lifted-pair (xf ∩ yf) f = Rep-lifted-pair bot f) ←→
  triple.pairs-less-eq (Rep-lifted-pair xf f) (Rep-lifted-pair (−yf) f)
hence (Rep-lifted-pair (xf ∩ yf) = Rep-lifted-pair bot) ←→ xf ≤ −yf
  using less-eq-lifted-pair.rep-eq by auto
thus (xf ∩ yf = bot) ←→ (xf ≤ −yf)
  by (simp add: Rep-lifted-pair-inject)
next
fix xf :: ('a,'b) lifted-pair
{
  fix f :: ('a,'b) phi
  have 1: triple (Rep-phi f)
      by (simp add: triple-def)
  let ?x = Rep-lifted-pair xf f
  obtain x1 x2 where (x1,x2) = ?x

using prod-collapse by blast

hence triple.pairs-sup (triple.pairs-uminus (Rep-phi f) ?x) (triple.pairs-uminus (Rep-phi f) (triple.pairs-uminus (Rep-phi f) ?x)) = triple.pairs-top

using I by (metis simp-lifted-pair triple.pairs-stone)
}
hence Rep-lifted-pair (−xf ⊔ −−xf) = Rep-lifted-pair top

using sup-lifted-pair.rep-eq uminus-lifted-pair.rep-eq top-lifted-pair.rep-eq by simp

thus −xf ⊔ −−xf = top

by (simp add: Rep-lifted-pair-inject)
qed

5.5 The Stone Algebra of the Triple of a Stone Algebra

In this section we specialise the above construction to a particular structure map, namely the one obtained in the triple of a Stone algebra. For this particular structure map (as well as for any other particular structure map) the resulting type is no longer a dependent type. It is just the set of pairs obtained for the given structure map.

typedef (overloaded) 'a stone-phi-pair = triple.pairs

(stone-phi::'a::stone-algebra regular ⇒ 'a dense-filter)

using stone-phi.pairs-bot-closed by auto

setup-lifting type-definition-stone-phi-pair

instantiation stone-phi-pair :: (stone-algebra) sup-inf-top-bot-uminus-ord

begin

lift-definition sup-stone-phi-pair :: 'a stone-phi-pair ⇒ 'a stone-phi-pair ⇒ 'a stone-phi-pair is triple.pairs-sup

using stone-phi.pairs-sup-closed by auto

lift-definition inf-stone-phi-pair :: 'a stone-phi-pair ⇒ 'a stone-phi-pair ⇒ 'a stone-phi-pair is triple.pairs-inf

using stone-phi.pairs-inf-closed by auto

lift-definition uminus-stone-phi-pair :: 'a stone-phi-pair ⇒ 'a stone-phi-pair is triple.pairs-uminus

using stone-phi.pairs-uminus-closed by auto

lift-definition bot-stone-phi-pair :: 'a stone-phi-pair ⇒ 'a stone-phi-pair is triple.pairs-bot

by (rule stone-phi.pairs-bot-closed)

lift-definition top-stone-phi-pair :: 'a stone-phi-pair ⇒ 'a stone-phi-pair is triple.pairs-top

by (rule stone-phi.pairs-top-closed)

78
The result is a Stone algebra and could be proved so by repeating and specialising the above proof for lifted pairs. We choose a different approach, namely by embedding the type of pairs into the lifted type. The embedding injects a pair $x$ into a function as the value at the given structure map; this makes the embedding injective. The value of the function at any other structure map needs to be carefully chosen so that the resulting function is a Stone algebra homomorphism. We use $\lambda x$, which is essentially a projection to the regular element component of $x$, whence the image has the structure of a Boolean algebra.

```plaintext
fun stone-phi-embed :: 'a::non-trivial-stone-algebra stone-phi-pair ⇒ ('a regular, 'a dense) lifted-pair
  where stone-phi-embed x = Abs-lifted-pair (λf . if Rep-phi f = stone-phi then Rep-stone-phi-pair x else triple.pairs-uminus (Rep-phi f) (triple.pairs-uminus (Rep-stone-phi-pair x)))
```

The following lemma shows that in both cases the value of the function is a valid pair for the given structure map.

```plaintext
lemma stone-phi-embed-triple-pair:
  (if Rep-phi f = stone-phi then Rep-stone-phi-pair x else triple.pairs-uminus (Rep-phi f) (triple.pairs-uminus (Rep-stone-phi-pair x))) ∈ triple.pairs (Rep-phi f)
  by (metis (no-types, hide-lams) Rep-stone-phi-pair simp-phi surj-pair triple.pairs-uminus-closed triple-def)
```

The following result shows that the embedding preserves the operations of Stone algebras. Of course, it is not (yet) a Stone algebra homomorphism as we do not know (yet) that the domain of the embedding is a Stone algebra. To establish the latter is the purpose of the embedding.

```plaintext
lemma stone-phi-embed-homomorphism:
  sup-inf-top-bot-uminus-ord-homomorphism stone-phi-embed
proof (intro conjI)
  let ?p = λf . triple.pairs-uminus (Rep-phi f)
  let ?pp = λf x . ?p f (?p f x)
  let ?q = λf x . ?pp f (Rep-stone-phi-pair x)
  show ∀ x y::'a stone-phi-pair . stone-phi-embed (x ⊔ y) = stone-phi-embed x ⊔ stone-phi-embed y
  proof (intro allI)
```
\textbf{fix \(x, y::\) a stone-phi-pair}

\textbf{have 1: \(\forall f.\) triple.pairs-sup \((?q f x) (?q f y) = ?q f (x \sqcup y)\)}

\textbf{proof}

\textbf{fix \(f::\) (a regular, a dense) phi}

\textbf{let \(?r = \text{Rep-phi} f\)}

\textbf{obtain \(x1, x2, y1, y2\) where 2: \((x1, x2) = \text{Rep-stone-phi-pair} x \land (y1, y2) = \text{Rep-stone-phi-pair} y\)}

\textbf{using \text{prod.collapse} by blast}

\textbf{hence triple.pairs-sup \((?q f x) (?q f y) = \text{triple.pairs-sup} (?pp f (x1, x2))\)}

\((?pp f (y1, y2))\)

\textbf{by simp}

\textbf{also have \(\ldots = \text{triple.pairs-sup} (- - x1, ?r (- x1)) (- - y1, ?r (- y1))\)}

\textbf{by \(\text{(simp add: triple.pairs-uminus.simps triple-def)}\)}

\textbf{also have \(\ldots = (- - x1 \sqcup - - y1, ?r (- x1) \sqcap ?r (- y1))\)}

\textbf{by simp}

\textbf{also have \(\ldots = (- - (x1 \sqcup y1), ?r (- (x1 \sqcup y1)))\)}

\textbf{by simp}

\textbf{also have \(\ldots = ?pp f (x1 \sqcup y1, x2 \sqcap y2)\)}

\textbf{by \(\text{(simp add: triple.pairs-uminus.simps triple-def)}\)}

\textbf{also have \(\ldots = ?pp f \text{ (triple.pairs-sup} (x1, x2) (y1, y2))\)}

\textbf{by simp}

\textbf{also have \(\ldots = ?q f (x \sqcup y)\)}

\textbf{using 2 by \(\text{(simp add: sup-stone-phi-pair.rep-eq)}\)}

\textbf{finally show \text{triple.pairs-sup} (?q f x) (?q f y) = ?q f (x \sqcup y)}

\textbf{.}

\textbf{qed}

\textbf{have stone-phi-embed \(x \sqcup\) stone-phi-embed \(y\) = Abs-lifted-pair \((\lambda f.\) if Rep-phi \(f = \text{stone-phi}\) then Rep-stone-phi-pair \(x\) else \(?q f x\) \(\sqcup\) Abs-lifted-pair \((\lambda f.\) if Rep-phi \(f = \text{stone-phi}\) then Rep-stone-phi-pair \(y\) else \(?q f y\)\)}

\textbf{by simp}

\textbf{also have \(\ldots = \text{Abs-lifted-pair} (\lambda f.\) triple.pairs-sup \((\text{if Rep-phi} f = \text{stone-phi}\) then \text{Rep-stone-phi-pair} \(x\) else \(?q f x\)) \(\sqcup\) \text{Abs-lifted-pair} \((\lambda f.\) if Rep-phi \(f = \text{stone-phi}\) then \text{Rep-stone-phi-pair} \(y\) else \(?q f y\))\)}

\textbf{by \(\text{(rule sup-lifted-pair.abs-eq) (simp-all add: eq-onp-same-args stone-phi-embed-triple-pair)}\)}

\textbf{also have \(\ldots = \text{Abs-lifted-pair} (\lambda f.\) triple.pairs-sup \((\text{Rep-stone-phi-pair} \(x\)) \text{Rep-stone-phi-pair} \(y\) else triple.pairs-sup \(?q f x\) \(?q f y\))\)}

\textbf{by \(\text{(simp add: if-distrib-2)}\)}

\textbf{also have \(\ldots = \text{Abs-lifted-pair} (\lambda f.\) triple.pairs-sup \((\text{Rep-stone-phi-pair} \(x\)) \text{Rep-stone-phi-pair} \(y\) else ?q f (x \sqcup y))\)}

\textbf{using \(f\) by meson}

\textbf{also have \(\ldots = \text{Abs-lifted-pair} (\lambda f.\) \text{Rep-stone-phi-pair} \(x \sqcup y\) else ?q f (x \sqcup y))\)}

\textbf{by \(\text{(metis sup-stone-phi-pair.rep-eq)}\)}

\textbf{also have \(\ldots = \text{stone-phi-embed} \(x \sqcup y\)\)}

\textbf{by simp}

\textbf{finally show \text{stone-phi-embed} \(x \sqcup y\) = \text{stone-phi-embed} \(x \sqcup\) \text{stone-phi-embed} \(y\)}
by simp

qed

next

let \( ?p = \lambda f . \text{triple.pairs-uminus } (\text{Rep-phi } f) \)
let \( ?pp = \lambda f x . ?p f (?p f x) \)
let \( ?q = \lambda f x . ?pp f (\text{Rep-stone-phi-pair } x) \)

show \( \forall x y . \text{'a stone-phi-pair . stone-phi-embed } (x \sqcap y) = \text{stone-phi-embed } x \sqcap \text{stone-phi-embed } y \)

proof (intro allI)

fix \( x y :: \text{'a stone-phi-pair} \)

have \( 1 . \forall f . \text{triple.pairs-inf } (?q f x) (?q f y) = ?q f (x \sqcap y) \)

proof

fix \( f :: (\text{'}a regular,\text{'a dense}) \text{phi} \)
let \( ?r = \text{Rep-phi } f \)

obtain \( x1 x2 y1 y2 \) where \( 2 . (x1,x2) = \text{Rep-stone-phi-pair } x \land (y1,y2) = \text{Rep-stone-phi-pair } y \)

using \( \text{prod.collapse by blast} \)

hence \( \text{triple.pairs-inf } (?q f x) (?q f y) = \text{triple.pairs-inf } (?pp f (x1,x2)) \)

by simp

also have \( ... = \text{triple.pairs-inf } (\text{--}x1,?r (\text{--}x1)) (\text{--}y1,?r (\text{--}y1)) \)

by (simp add: \( \text{triple.pairs-uminus,simps triple-def} \))

also have \( ... = (\text{--}x1 \sqcap \text{--}y1,?r (\text{--}x1) \sqcup ?r (\text{--}y1)) \)

by simp

also have \( ... = (\text{--}(x1 \sqcap y1),?r (\text{--}(x1 \sqcap y1))) \)

by simp

also have \( ... = ?pp f (x1 \sqcap y1,x2 \sqcup y2) \)

by (simp add: \( \text{triple.pairs-uminus,simps triple-def} \))

also have \( ... = ?pp f (\text{triple.pairs-inf } (x1,x2) (y1,y2)) \)

by simp

also have \( ... = ?q f (x \sqcap y) \)

using \( 2 \) by (simp add: \( \text{inf-stone-phi-pair.rep-eq} \))

finally show \( \text{triple.pairs-inf } (?q f x) (?q f y) = ?q f (x \sqcap y) \)

.

qed

have \( \text{stone-phi-embed } x \sqcap \text{stone-phi-embed } y = \text{Abs-lifted-pair } (\lambda f . \text{if Rep-phi } f = \text{stone-phi then Rep-stone-phi-pair } x \text{ else } ?q f x) \sqcap \text{Abs-lifted-pair } (\lambda f . \text{if Rep-phi } f = \text{stone-phi then Rep-stone-phi-pair } y \text{ else } ?q f y) \)

by simp

also have \( ... = \text{Abs-lifted-pair } (\lambda f . \text{triple.pairs-inf } (\text{if Rep-phi } f = \text{stone-phi then Rep-stone-phi-pair } x \text{ else } ?q f x) \sqcap \text{Abs-lifted-pair } (\lambda f . \text{if Rep-phi } f = \text{stone-phi then Rep-stone-phi-pair } y \text{ else } ?q f y)) \)

by (rule \( \text{inf-lifted-pair.abs-eq} \) (simp-all add: \( \text{eq-onp-same-args stone-phi-embed-triple-pair} \))

also have \( ... = \text{Abs-lifted-pair } (\lambda f . \text{if Rep-phi } f = \text{stone-phi then triple.pairs-inf } (\text{Rep-stone-phi-pair } x) (\text{Rep-stone-phi-pair } y) \text{ else triple.pairs-inf } (?q f x) (?q f y)) \)

by (simp add: \( \text{if-distrib-2} \))

also have \( ... = \text{Abs-lifted-pair } (\lambda f . \text{if Rep-phi } f = \text{stone-phi then} \)
triple.pairs-inf (Rep-stone-phi-pair x) (Rep-stone-phi-pair y) else ?q f (x ∩ y))

using f by meson
also have ... = Abs-lifted-pair (λf . if Rep-phi f = stone-phi then Rep-stone-phi-pair (x ∩ y) else ?q f (x ∩ y))
  by (metis inf-stone-phi-pair.rep-eq)
also have ... = stone-phi-embed (x ∩ y)
  by simp

finally show stone-phi-embed (x ∩ y) = stone-phi-embed x ∩ stone-phi-embed y
  by simp
qed

next
have stone-phi-embed (top:'a stone-phi-pair) = Abs-lifted-pair (λf . if Rep-phi f = stone-phi then Rep-stone-phi-pair top else triple.pairs-uminus (Rep-phi f))
  (triple.pairs-uminus (Rep-phi f) (Rep-stone-phi-pair top)))
  by simp
also have ... = Abs-lifted-pair (λf . if Rep-phi f = stone-phi then (top,bot) else triple.pairs-uminus (Rep-phi f) (triple.pairs-uminus (Rep-phi f) (top,bot)))
  by (metis (no-types, hide-lams) bot-filter.abs-eq top-stone-phi-pair.rep-eq)
also have ... = Abs-lifted-pair (λf . if Rep-phi f = stone-phi then (top,bot) else triple.pairs-uminus (Rep-phi f) (bot,top))
  by (metis (no-types, hide-lams) dense-closed-top simp-phi triple.pairs-uminus.simps triple-def)
also have ... = Abs-lifted-pair (λf . if Rep-phi f = stone-phi then (top,bot) else (top,bot))
  by (metis (no-types, hide-lams) p-bot simp-phi triple.pairs-uminus.simps triple-def)
also have ... = Abs-lifted-pair (λf . (top,Abs-filter {top}))
  by (simp add: bot-filter.abs-eq)
also have ... = top
  by (rule top-lifted-pair.abs-eq[THEN sym])
finally show stone-phi-embed (top:'a stone-phi-pair) = top

next
have stone-phi-embed (bot:'a stone-phi-pair) = Abs-lifted-pair (λf . if Rep-phi f = stone-phi then Rep-stone-phi-pair bot else triple.pairs-uminus (Rep-phi f))
  (triple.pairs-uminus (Rep-phi f) (Rep-stone-phi-pair bot)))
  by simp
also have ... = Abs-lifted-pair (λf . if Rep-phi f = stone-phi then (bot,top) else triple.pairs-uminus (Rep-phi f) (triple.pairs-uminus (Rep-phi f) (bot,top)))
  by (metis (no-types, hide-lams) top-filter.abs-eq bot-stone-phi-pair.rep-eq)
also have ... = Abs-lifted-pair (λf . if Rep-phi f = stone-phi then (bot,top) else triple.pairs-uminus (Rep-phi f) (top,bot))
  by (metis (no-types, hide-lams) p-bot simp-phi triple.pairs-uminus.simps triple-def)
also have ... = Abs-lifted-pair (λf . if Rep-phi f = stone-phi then (bot,top) else (bot,top))
  by (metis (no-types, hide-lams) p-top simp-phi triple.pairs-uminus.simps triple-def)

82
also have \( \ldots = \text{Abs-lifted-pair} (\lambda f . (\bot, \text{Abs-filter} \ \text{UNIV})) \)
  by (simp add: top-filter.abs-eq)
also have \( \ldots = \bot \)
  by (rule bot-lifted-pair.abs-eq[THEN sym])
finally show \( \text{stone-phi-embed} (\bot.\cdot\text{a stone-phi-pair}) = \bot \)

next
let \( ?p = \lambda f \cdot \text{triple.pairs-uminus} (\text{Rep-phi} \ f) \)
let \( ?pp = \lambda f x . ?p f (?p f x) \)
let \( ?q = \lambda f x . ?pp f (\text{Rep-stone-phi-pair} \ x) \)
show \( \forall x::'a\ \text{stone-phi-pair}. \ \text{stone-phi-embed} (-x) = -\text{stone-phi-embed} x \)
proof (intro allI)
  fix \( x :: 'a \ \text{stone-phi-pair} \)
  have 1: \( \forall f. \ \text{triple.pairs-uminus} (\text{Rep-phi} \ f) (\ ?q f x) = ?q f (-x) \)
  proof
    fix \( f :: ('a \ \text{regular},'a \ \text{dense}) \ \text{phi} \)
    let \( \ ?r = \text{Rep-phi} \ f \)
    obtain \( x1 x2 \) where 2: \( (x1,x2) = \text{Rep-stone-phi-pair} \ x \)
    using prod.collapse by blast
    hence \( \text{triple.pairs-uminus} (\text{Rep-phi} \ f) (\ ?q f x) = \text{triple.pairs-uminus} (\ ?pp f (x1,x2)) \)
  by simp
  also have \( \ldots = \text{triple.pairs-uminus} (\text{Rep-phi} \ f) (-x1,?r (-x1)) \)
    by (simp add: triple.pairs-uminus.simps triple-def)
  also have \( \ldots = (-x1,?r (-x1)) \)
    by (simp add: triple.pairs-uminus.simps triple-def)
  also have \( \ldots = ?pp f (-x1,\text{stone-phi} x1) \)
    by (simp add: triple.pairs-uminus.simps triple-def)
  also have \( \ldots = ?pp f (\text{triple.pairs-uminus} \ \text{stone-phi} \ (x1,x2)) \)
  by simp
  also have \( \ldots = ?q f (-x) \)
  using 2 by (simp add: uminus-stone-phi-pair.rep-eq)
finally show \( \text{triple.pairs-uminus} (\text{Rep-phi} \ f) (\ ?q f x) = ?q f (-x) \)
qed

have \( -\text{stone-phi-embed} x = -\text{Abs-lifted-pair} (\lambda \ f . \ \text{if Rep-phi} \ f = \text{stone-phi} \ \text{then Rep-stone-phi-pair} \ x \ \text{else} \ ?q f x) \)
then Rep-stone-phi-pair \ x \ \text{else} \ ?q f x \)
by simp
also have \( \ldots = \text{Abs-lifted-pair} (\lambda \ f . \ \text{triple.pairs-uminus} (\text{Rep-phi} \ f) (\text{if Rep-phi} \ f = \text{stone-phi} \ \text{then Rep-stone-phi-pair} \ x \ \text{else} \ ?q f x)) \)
by (rule uminus-lifted-pair.abs-eq) (simp-all add: eq-onp-same-args stone-phi-embed-triple-pair)
also have \( \ldots = \text{Abs-lifted-pair} (\lambda \ f . \ \text{if Rep-phi} \ f = \text{stone-phi} \ \text{then triple.pairs-uminus} (\text{Rep-phi} \ f) (\text{Rep-stone-phi-pair} \ x) \ \text{else} \ \text{triple.pairs-uminus} (\text{Rep-phi} \ f) (\ ?q f x)) \)
by (simp add: if-distrib)
also have \( \ldots = \text{Abs-lifted-pair} (\lambda \ f . \ \text{if Rep-phi} \ f = \text{stone-phi} \ \text{then triple.pairs-uminus} (\text{Rep-phi} \ f) (\text{Rep-stone-phi-pair} \ x) \ \text{else} \ ?q f (-x)) \)
using \( f \) by meson

83
also have \( \ldots = \text{Abs-lifted-pair} (\lambda f . \text{if } \text{Rep-phi } f = \text{stone-phi} \text{ then } \text{Rep-stone-phi-pair} \ (x) \text{ else } ?q f \ (\neg x)) \)
by (metis uminus-stone-phi-pair.rep-eq)
also have \( \ldots = \text{stone-phi-embed} \ (x) \)
by simp
finally show \( \text{stone-phi-embed} \ (x) = -\text{stone-phi-embed} x \)
by simp
qed

next
let \( ?p = \lambda f . \text{triple.pairs-uminus} (\text{Rep-phi } f) \)
let \( ?pp = \lambda f x . ?p f \ (\neg x) \)
let \( ?q = \lambda f x . ?pp f \ (\text{Rep-stone-phi-pair} x) \)
show \( \forall x y :: \text{stone-phi-pair} . \ x \leq y \to \text{stone-phi-embed} x \leq \text{stone-phi-embed} y \)
proof (intro allI, rule impI)
  fix \( x y :: \text{stone-phi-pair} \)
  assume \( \mathbf{1} : x \leq y \)
  have \( \forall f :: \text{triple.pairs-less-eq} \ (\text{if } \text{Rep-phi } f = \text{stone-phi} \text{ then } \text{Rep-stone-phi-pair} x \text{ else } ?q f x) \ (\text{if } \text{Rep-phi } f = \text{stone-phi} \text{ then } \text{Rep-stone-phi-pair} y \text{ else } ?q f y) \)
proof
  fix \( f :: (\text{a regular}, \text{a dense}) \phi \)
  let \( ?r = \text{Rep-phi } f \)
  obtain \( x_1 x_2 y_1 y_2 \) where \( \mathbf{2} : (x_1 x_2) = \text{Rep-stone-phi-pair} x \wedge (y_1 y_2) = \text{Rep-stone-phi-pair} y \)
using prod.collapse by blast
have \( x_1 \leq y_1 \)
using \( \mathbf{1} \ \mathbf{2} \) by (metis less-eq-stone-phi-pair.rep-eq stone-phi.pairs-less-eq.simps)
hence \( \neg x_1 \leq \neg y_1 \wedge ?r \ (\neg y_1) \leq ?r \ (\neg x_1) \)
by (metis compl-le-compl-iff le-iff-sup simp-phi)
hence triple.pairs-less-eq \( (-x_1, ?r \ (-x_1)) \ (-y_1, ?r \ (-y_1)) \)
by simp
hence triple.pairs-less-eq \( (?pp f \ (x_1 x_2)) \ (?pp f \ (y_1 y_2)) \)
by (simp add: triple.pairs-uminus.simps triple-def)
hence triple.pairs-less-eq \( (?q f x) \ (?q f y) \)
using \( \mathbf{2} \) by simp
hence if \( ?r = \text{stone-phi} \text{ then } \text{triple.pairs-less-eq} \ (\text{Rep-stone-phi-pair} x) \ (\text{Rep-stone-phi-pair} y) \) else triple.pairs-less-eq \( (?q f x) \ (?q f y) \)
using \( \mathbf{1} \) by (simp add: less-eq-stone-phi-pair.rep-eq)
thus triple.pairs-less-eq \( (?r = \text{stone-phi} \text{ then } \text{Rep-stone-phi-pair} x \text{ else } ?q f x) \ (\text{if } ?r = \text{stone-phi} \text{ then } \text{Rep-stone-phi-pair} y \text{ else } ?q f y) \)
by (simp add: if-distrib-2)
Qed
hence \( \text{Abs-lifted-pair} (\lambda f . \text{if } \text{Rep-phi } f = \text{stone-phi} \text{ then } \text{Rep-stone-phi-pair} x \text{ else } ?q f x) \leq \text{Abs-lifted-pair} (\lambda f . \text{if } \text{Rep-phi } f = \text{stone-phi} \text{ then } \text{Rep-stone-phi-pair} y \text{ else } ?q f y) \)
by (subst less-eq-lifted-pair.abs-eq) (simp-all add: eq-onp-same-args stone-phi-embed-triple-pair)
thus \( \text{stone-phi-embed} x \leq \text{stone-phi-embed} y \)
The following lemmas show that the embedding is injective and reflects the order. The latter allows us to easily inherit properties involving inequalities from the target of the embedding, without transforming them to equations.

**lemma** stone-phi-embed-injective:

\[ \text{inj \ stone-phi-embed} \]

**proof** (rule injI)

- **fix** \( x \ y :: \text{'a stone-phi-pair} \)
- **have** 1: \( \text{Rep-phi} (\text{Abs-phi \ stone-phi}) = \text{stone-phi} \)
  - **by** (simp add: Abs-phi-inverse stone-phi.hom)
- **assume** 2: \( \text{stone-phi-embed} \ x = \text{stone-phi-embed} \ y \)
  - **have** \( \forall \ x :: \text{'a stone-phi-pair} \dot. \\text{Rep-lifted-pair} (\text{stone-phi-embed} \ x) = (\lambda \ f \dot. \text{if} \ \text{Rep-phi} \ f = \text{stone-phi} \ \text{then} \ \text{Rep-stone-phi-pair} \ x \ \text{else} \ \text{triple.pairs-uminus} (\text{Rep-phi} \ f) (\text{Rep-stone-phi-pair} \ x)) \)
  - **by** (simp add: Abs-lifted-pair-inverse stone-phi-embed-triple-pair)
  - **hence** \( (\lambda \ f \dot. \text{if} \ \text{Rep-phi} \ f = \text{stone-phi} \ \text{then} \ \text{Rep-stone-phi-pair} \ x \ \text{else} \ \text{triple.pairs-uminus} (\text{Rep-phi} \ f) (\text{Rep-stone-phi-pair} \ y)) \)
  - **using** 2 by metis
  - **hence** \( \text{Rep-stone-phi-pair} \ x = \text{Rep-stone-phi-pair} \ y \)
  - **using** 1 by metis
  - **thus** \( x = y \)
  - **by** (simp add: Rep-stone-phi-pair-inject)

**qed**

**lemma** stone-phi-embed-order-injective:

\[ \text{assumes stone-phi-embed} \ x \leq \text{stone-phi-embed} \ y \]

\[ \text{shows} \ x \leq y \]

**proof**

- **let** \( ?f = \text{Abs-phi \ stone-phi} \)
- **have** \( \forall \ f :: \text{triple.pairs-less-eq} (\text{if} \ \text{Rep-phi} \ ?f = \text{stone-phi} \ \text{then} \ \text{Rep-stone-phi-pair} \ x \ \text{else} \ \text{triple.pairs-uminus} (\text{Rep-phi} \ f) (\text{Rep-stone-phi-pair} \ x)) \)
  - **using** \( \text{assms} \) by (subst less-eq-lifted-pair.abs-eq[THEN sym]) (simp-all add: eq-onp-same-args stone-phi-embed-triple-pair)
  - **hence** \( \text{triple.pairs-less-eq} (\text{if} \ \text{Rep-phi} \ ?f = (\text{stone-phi} :: \text{'a regular} \Rightarrow \text{'a dense-filter}) \ \text{then} \ \text{Rep-stone-phi-pair} \ x \ \text{else} \ \text{triple.pairs-uminus} (\text{Rep-phi} \ ?f) (\text{Rep-stone-phi-pair} \ x)) \)
  - **using** \( \text{assms} \) by (subst less-eq-lifted-pair.abs-eq[THEN sym]) (simp-all add: eq-onp-same-args stone-phi-embed-triple-pair)

85
by simp

hence \( \text{triple.pairs-less-eq} (\text{Rep-stone-phi-pair } x) (\text{Rep-stone-phi-pair } y) \)
by (simp add: Abs-phi-inverse stone-phi.hom)
thus \( x \leq y \)
by (simp add: less-eq-stone-phi-pair.rep-eq)

qed

Now all Stone algebra axioms can be inherited using the embedding. This is due to the fact that the axioms are universally quantified equations or conditional equations (or inequalities); this is called a quasivariety in universal algebra. It would be useful to have this construction available for arbitrary quasivarieties.

instantiation stone-phi-pair :: (non-trivial-stone-algebra) stone-algebra
begin

instance
apply intro-classes
apply (simp add: less-stone-phi-pair.rep-eq less-eq-stone-phi-pair.rep-eq)
apply (simp add: stone-phi-embed-order-injective)
apply (meson order.trans stone-phi-embed-homomorphism
stone-phi-embed-order-injective)
apply (meson stone-phi-embed-homomorphism antisym
stone-phi-embed-order-injective injD)
apply (meson inf.ge1 stone-phi-embed-homomorphism
stone-phi-embed-order-injective)
apply (meson inf.ge2 stone-phi-embed-homomorphism
stone-phi-embed-order-injective)
apply (meson infgreatest stone-phi-embed-homomorphism
stone-phi-embed-order-injective)
apply (metis sup.ge1)
apply (metis sup.least)
apply (metis bot.extremum)
apply (metis sup-inf-distrib1 stone-phi-embed-injective injD)
apply (metis sup-inf-distrib2 stone-phi-embed-injective injD)
apply (metis pseudo-complement)
by (metis injD stone-phi-embed-homomorphism stone-phi-embed-injective stone)

end

5.6 Stone Algebra Isomorphism
In this section we prove that the Stone algebra of the triple of a Stone algebra is isomorphic to the original Stone algebra. The following two definitions give the isomorphism.

**abbreviation** sa-iso-inv :: 'a::non-trivial-stone-algebra stone-phi-pair ⇒ 'a

**abbreviation** sa-iso :: 'a::non-trivial-stone-algebra ⇒ 'a stone-phi-pair
where sa-iso ≡ λx. Abs-stone-phi-pair (Abs-regular (−−x),stone-phi (Abs-regular (−x)) ⊔ up-filter (Abs-dense (x ⊔ −x)))

**lemma** sa-iso-triple-pair: (Abs-regular (−−x),stone-phi (Abs-regular (−x)) ⊔ up-filter (Abs-dense (y ⊔ −y))) ∈ triple.pairs stone-phi by (metis (mono-tags, lifting) double-compl eq-onp-same-args stone-phi-def sa-iso-inv-uminus-regular.abs-eq)

**lemma** stone-phi-inf-dense: stone-phi (Abs-regular (−x)) ⊓ up-filter (Abs-dense (y ⊔ −y)) ≤ up-filter (Abs-dense (y ⊔ −y))

**proof**
- have Rep-filter (stone-phi (Abs-regular (−x)) ⊓ up-filter (Abs-dense (y ⊔ −y))) ≤ ↑(Abs-dense (y ⊔ −y))
  - by (simp add: Abs-dense-inverse Int-Collect)

**finally have** −−x ≤ ?r ∧ Abs-dense (y ⊔ −y) ≤ z
  - by (metis (mono-tags, lifting) Abs-regular-inverse Int-Collect)

**hence** −−x ≤ ?r ∧ −y ≤ ?r
  - by (simp add: Abs-dense-inverse less-eq-dense.rep-eq)

**hence** y ⊔ −y ⊔ x ≤ ?r
  - using order-trans pp-increasing by auto

**hence** Abs-dense (y ⊔ −y ⊔ x) ≤ Abs-dense ?r
  - by (subst less-eq-dense.abs-eq) (simp-all add: eq-onp-same-args)

**thus** z ∈ ↑(Abs-dense (y ⊔ −y ⊔ x))
  - by (simp add: Rep-dense-inverse)

**qed**

**hence** Abs-filter (Rep-filter (stone-phi (Abs-regular (−x)) ⊓ up-filter (Abs-dense (y ⊔ −y)))) ≤ up-filter (Abs-dense (y ⊔ −y ⊔ x))
  - by (simp add: eq-onp-same-args less-eq-filter.abs-eq)
thus thesis
  by (simp add: Rep-filter-inverse)
qed

lemma stone-phi-complement:
  complement (stone-phi (Abs-regular (−x))) (stone-phi (Abs-regular (−x)))
  by (metis (mono-tags, lifting) eq-onp-same-args stone-phi phi-complemented uminus-regular abs-eq)

lemma up-dense-stone-phi:
  up-filter (Abs-dense (x ⊔ −x)) ≤ stone-phi (Abs-regular (−x))
proof
  have ↑ (Abs-dense (x ⊔ −x)) ≤ stone-phi-set (Abs-regular (−x))
    proof
      fix z :: 'a dense
      let ?r = Rep-dense z
      assume z ∈ ↑ (Abs-dense (x ⊔ −x))
      hence −−− x ≤ ?r
        by (simp add: Abs-dense-inverse less-eq-dense rep-eq)
      hence − Rep-regular (Abs-regular (−x)) ≤ ?r
        by (metis (mono-tags, lifting) Abs-regular-inverse mem-Collect-eq)
      thus z ∈ stone-phi-set (Abs-regular (−x))
        by simp
    qed
  also have ...
    = stone-phi (Abs-regular (−x)) ∩ up-filter (Abs-dense (x ⊔ −x))
    by (simp add: inf sup-commute inf-sup-distrib1 stone-phi-complement)
  also have ...
    = up-filter (Abs-dense (x ⊔ −x))
    using up-dense-stone-phi inf.absorb2 by auto
  finally have 1: triple.rho-pair stone-phi (Abs-regular (−x), stone-phi (Abs-regular (−x)) ∪ up-filter (Abs-dense (x ⊔ −x))) = Abs-dense (x ⊔ −x)
    using up-filter-injective by auto
  have sa-iso-inv (sa-iso x) = (λp . Rep-regular (fst p) ∩ Rep-dense (triple.rho-pair stone-phi p)) (Abs-regular (−x), stone-phi (Abs-regular (−x)) ∪ up-filter (Abs-dense (x ⊔ −x)))
    by (simp add: Abs-stone-phi-pair-inverse sa-iso-triple-pair)
also have ... = Rep-regular (Abs-regular \((-x)\)) ∩ Rep-dense (triple.rho-pair stone-phi (Abs-regular \((-x)\)),\(\text{stone-phi} (Abs-regular \((-x)\)) \cup \text{up-filter} (Abs-dense (x \sqcup -x)))\)
   by simp
also have ... = \(-x \sqcap \text{Rep-dense} (Abs-dense (x \sqcup -x))\)
   using 1 by (subst Abs-regular-inverse) auto
also have ... = \(-x \sqcap (x \sqcup -x)\)
   by (subst Abs-dense-inverse) simp-all
also have ... = \(x\)
   by simp
finally show ?thesis
   by auto
qed

lemma sa-iso-right-invertible:
  sa-iso (sa-iso-inv p) = p:
proof
  obtain x y where 1: \((x,y) = \text{Rep-stone-phi-pair} p\)
    using prod.collapse by blast
  hence 2: \((x,y) \in \text{triple.pairs} \text{stone-phi}\)
    by (simp add: Rep-stone-phi-pair)
  hence 3: \text{stone-phi} \((-x) \leq y\)
    by (simp add: stone-phi.pairs-phi-less-eq)
  have 4: \(\forall z . \ z \in \text{Rep-filter} (\text{stone-phi} x \sqcap y) \longrightarrow -\text{Rep-regular} x \leq \text{Rep-dense} z\)
proof (rule allI, rule impl)
  fix z :: 'a dense
  let ?r = \text{Rep-dense} z
  assume z \in Rep-filter (stone-phi x \sqcap y)
  hence z \in Rep-filter (stone-phi x)
    by (simp add: inf-filter.rep-eq)
  also have ... = \text{stone-phi-set} \(x\)
    by (simp add: stone-phi-def Abs-filter-inverse stone-phi-set-filter)
  finally show -\text{Rep-regular} x \leq ?r
    by simp
qed

have triple.rho-pair stone-phi \((x,y) \in \uparrow\text{triple.rho-pair} \text{stone-phi} (x,y)\)\)
   by simp
also have ... = Rep-filter (Abs-filter (∧\text{triple.rho-pair} \text{stone-phi} (x,y)))
   by (simp add: Abs-filter-inverse)
also have ... = Rep-filter (\text{stone-phi} x \sqcap y)
   using 2 \text{stone-phi.get-rho-pair-char by fastforce}
finally have triple.rho-pair stone-phi \((x,y) \in \text{Rep-filter} (\text{stone-phi} x \sqcap y)\)
   by simp
hence 5: -\text{Rep-regular} x \leq \text{Rep-dense} (\text{triple.rho-pair} \text{stone-phi} (x,y))
   using 4 by simp
have 6: \text{sa-iso-inv p} = \text{Rep-regular} x \sqcap \text{Rep-dense} (\text{triple.rho-pair} \text{stone-phi} (x,y))
   using 1 by (metis fstI)
  hence -\text{sa-iso-inv p} = -\text{Rep-regular} x
by simp  

hence sa-iso (sa-iso-inv p) = Abs-stone-phi-pair (Abs-regular (¬ Rep-regular x),stone-phi (Abs-regular (¬ Rep-regular x)) ⊔ up-filter (Abs-dense ((Rep-regular x ∩ Rep-dense (triple.rho-pair stone-phi (x,y)))) ⊔ ¬ Rep-regular x))

using 6 by simp  

also have ... = Abs-stone-phi-pair (x,stone-phi (¬ x) ⊔ up-filter (Abs-dense ((Rep-regular x ∩ Rep-dense (triple.rho-pair stone-phi (x,y)))) ⊔ ¬ Rep-regular x))

  by (metis (mono-tags, lifting) Rep-regular-inverse double-compl uminus-regular.rep-eq)

  also have ... = Abs-stone-phi-pair (x,stone-phi (¬ x) ⊔ up-filter (Abs-dense (Rep-dense (triple.rho-pair stone-phi (x,y)))) ⊔ ¬ Rep-regular x))

  6 by simp

  also have ... = Abs-stone-phi-pair (x,stone-phi (¬ x) ⊔ up-filter (triple.rho-pair stone-phi (x,y))))

  6 by (simp add: sup.absorb1)

  also have ... = Abs-stone-phi-pair (x,stone-phi (¬ x) ⊔ up-filter (triple.rho-pair stone-phi (x,y))))

  6 by (simp add: Rep-dense-inverse)

  also have ... = Abs-stone-phi-pair (x,stone-phi (¬ x) ⊔ (stone-phi x ∩ y))

  using 2 stone-phi.get-rho-pair-char by fastforce

  also have ... = Abs-stone-phi-pair (x,stone-phi (¬ x) ⊔ y)

  3 by (simp add: stone-phi.complemented sup.commute sup-inf-distrib1)

  also have ... = Abs-stone-phi-pair (x,y)

  3 by (simp add: le-iff-sup)

  also have ... = p

  using 1 by (simp add: Rep-stone-phi-pair-inverse)

  finally show thesis

  qed

  It remains to show the homomorphism properties, which is done in the following result.

lemma sa-iso:  

  stone-algebra-isomorphism sa-iso

proof (intro conjI)

have Abs-stone-phi-pair (Abs-regular (¬ bot),stone-phi (Abs-regular (¬ bot)) ⊔ up-filter (Abs-dense (bot ∪ ¬ bot)) = Abs-stone-phi-pair (bot,stone-phi top ⊔ up-filter top)

  by (simp add: bot-regular.abs-eq top-regular.abs-eq top-dense.abs-eq)

  also have ... = Abs-stone-phi-pair (bot,stone-phi top)

  6 by (simp add: stone-phi.hom)

  also have ... = bot

  6 by (simp add: bot-stone-phi-pair-def stone-phi.top)

  finally show sa-iso bot = bot

  next

have Abs-stone-phi-pair (Abs-regular (¬ top),stone-phi (Abs-regular (¬ top)) ⊔ up-filter (Abs-dense (top ∪ ¬ top)) = Abs-stone-phi-pair (top,stone-phi bot ⊔
up-filter top
by (simp add: bot-regular.abs-eq top-regular.abs-eq top-dense.abs-eq)
also have ... = top
by (simp add: stone-phi.phi-bot top-stone-phi-pair-def)
finally show sa-iso top = top
.
next
have 1: \( \forall x y :: 'a . \text{dense } (x \sqcup -x \sqcup y) \)
by simp
have 2: \( \forall x y :: 'a . \text{up-filter } (\text{Abs-dense } (x \sqcup -x \sqcup y)) \leq (\text{stone-phi} (\text{Abs-regular } (-x)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x))) \cap (\text{stone-phi} (\text{Abs-regular } (-y)) \sqcup \text{up-filter } (\text{Abs-dense } (y \sqcup -y))) \)
proof (intro allI)
fix x y :: 'a
let ?u = \text{Abs-dense } (x \sqcup -x \sqcup -y)
let ?v = \text{Abs-dense } (y \sqcup -y)
have \( \uparrow (\text{Abs-dense } (x \sqcup -x \sqcup y)) \leq \text{Rep-filter } (\text{stone-phi} (\text{Abs-regular } (-y)) \sqcup \text{up-filter } ?v) \)
proof
fix z
assume z \( \in \uparrow (\text{Abs-dense } (x \sqcup -x \sqcup y)) \)
hence \( \text{Abs-dense } (x \sqcup -x \sqcup y) \leq z \)
by simp
hence 3: \( x \sqcup -x \sqcup y \leq \text{Rep-dense } z \)
by (simp add: Abs-dense-inverse less-eq-dense.rep-eq)
have \( y \leq x \sqcup -x \sqcup -y \)
by (simp add: le-supI2 pp-increasing)
hence \( (x \sqcup -x \sqcup -y) \cap (y \sqcup -y) = y \sqcup ((x \sqcup -x \sqcup -y) \cap -y) \)
by (simp add: le-iff-sup sup-inf-distrib1)
also have ... = \( y \sqcup ((x \sqcup -x) \cap -y) \)
by (simp add: inf-commute inf-sup-distrib1)
also have ... \( \leq \text{Rep-dense } z \)
using 3 by (meson le-inflI sup.bounded-iff)
finally have \( \text{Abs-dense } ((x \sqcup -x \sqcup -y) \cap (y \sqcup -y)) \leq z \)
by (simp add: Abs-dense-inverse less-eq-dense.rep-eq)
hence 4: \( ?u \cap ?v \leq z \)
by (simp add: eq-onp-same-args inf-dense.abs-eq)
have \( \neg \text{Rep-regular } (\text{Abs-regular } (-y)) = -y \)
by (metis (mono-tags, lifting) mem-Collect-eq Abs-regular-inverse)
also have ... \( \leq \text{Rep-dense } ?u \)
by (simp add: Abs-dense-inverse)
finally have \( ?u \in \text{stone-phi-set } (\text{Abs-regular } (-y)) \)
by simp
hence 5: \( ?u \in \text{Rep-filter } (\text{stone-phi } (\text{Abs-regular } (-y))) \)
by (metis mem-Collect-eq stone-phi-def stone-phi-set-filter Abs-filter-inverse)
have \( ?v \in \uparrow ?v \)
by simp
hence \( ?v \in \text{Rep-filter } (\text{up-filter } ?v) \)

91
by (metis Abs-filter-inverse mem-Collect-eq up-filter)
thus \( z \in \text{Rep-filter} (\text{stone-phi} (\text{Abs-regular} (-y)) \cup \text{up-filter} ?v) \)
using 4 5 sup-filter.rep-eq by blast
qed

hence \( \text{up-filter} (\text{Abs-dense} (x \sqcup -x \sqcup y)) \leq \text{Abs-filter} (\text{stone-phi} (\text{Abs-regular} (-y)) \cup \text{up-filter} ?v) \)
by (simp add: eq-onp-same-args less-eq-filter.abs-eq)
also have \( \cdots = \text{stone-phi} (\text{Abs-regular} (-y)) \sqcup \text{up-filter} ?v \)
by (simp add: Rep-filter-inverse)

finally show \( \text{up-filter} (\text{Abs-dense} (x \sqcup -x \sqcup y)) \leq (\text{stone-phi} (\text{Abs-regular} (-x)) \cup \text{up-filter} (\text{Abs-dense} (x \sqcup -x))) \cap (\text{stone-phi} (\text{Abs-regular} (-y)) \cup \text{up-filter} (\text{Abs-dense} (y \sqcup -y))) \)
by (metis le-infl le-supI2 sup-bot.right-neutral up-filter-dense-antitone)

qed

have 6: \( \forall x::a . \in-p-image (-x) \)
by auto
show \( \forall x y::a . \text{sa-iso} (x \sqcup y) = \text{sa-iso} x \sqcup \text{sa-iso} y \)
proof (intro allI)
fix \( x y :: a \)
have 7: \( \text{up-filter} (\text{Abs-dense} (x \sqcup -x)) \cap \text{up-filter} (\text{Abs-dense} (y \sqcup -y)) \leq \text{up-filter} (\text{Abs-dense} (y \sqcup -y)) \)
proof

have \( \text{up-filter} (\text{Abs-dense} (x \sqcup -x)) \cap \text{up-filter} (\text{Abs-dense} (y \sqcup -y)) = \text{up-filter} (\text{Abs-dense} (x \sqcup -x) \sqcup \text{Abs-dense} (y \sqcup -y)) \)
by (metis up-filter-dist-sup)
also have \( \cdots = \text{up-filter} (\text{Abs-dense} (x \sqcup -x \sqcup (y \sqcup -y))) \)
by (subst sup-dense.abs-eq) (simp-all add: eq-onp-same-args)
also have \( \cdots = \text{up-filter} (\text{Abs-dense} (y \sqcup -y \sqcup x \sqcup -x)) \)
by (simp add: sup-commute sup-left-commute)
also have \( \cdots \leq \text{up-filter} (\text{Abs-dense} (y \sqcup -y \sqcup x)) \)
using up-filter-dense-antitone by auto

finally show \( ?thesis \)

qed

have \( \text{Abs-dense} (x \sqcup y \sqcup -(x \sqcup y)) = \text{Abs-dense} ((x \sqcup -x \sqcup y) \sqcap (y \sqcup -y \sqcup x)) \)
by (simp add: sup-commute sup-inf-distrib1 sup-left-commute)
also have \( \cdots = \text{Abs-dense} (x \sqcup -x \sqcup y) \sqcap \text{Abs-dense} (y \sqcup -y \sqcup x) \)
using 1 by (metis (mono-tags, lifting) Abs-dense-inverse Rep-dense-inverse inf-dense_rep-eq mem-Collect-eq)

finally have 8: \( \text{up-filter} (\text{Abs-dense} (x \sqcup y \sqcup -(x \sqcup y))) = \text{up-filter} (\text{Abs-dense} (x \sqcup -x \sqcup y)) \cup \text{up-filter} (\text{Abs-dense} (y \sqcup -y \sqcup x)) \)
by (simp add: up-filter-dist-inf)
also have \( \cdots \leq (\text{stone-phi} (\text{Abs-regular} (-x)) \cup \text{up-filter} (\text{Abs-dense} (x \sqcup -x))) \cap (\text{stone-phi} (\text{Abs-regular} (-y)) \cup \text{up-filter} (\text{Abs-dense} (y \sqcup -y))) \)
using 2 by (simp add: inf.sup-commute le-sup-iff)

finally have 9: \( (\text{stone-phi} (\text{Abs-regular} (-x)) \cap \text{stone-phi} (\text{Abs-regular} (-y))) \sqcup \text{up-filter} (\text{Abs-dense} (x \sqcup y \sqcup -(x \sqcup y))) \leq \cdots \)
by (simp add: le-supI1)

92
have \(\ldots = (\text{stone-phi} (\text{Abs-regular} (-x)) \cap \text{stone-phi} (\text{Abs-regular} (-y))) \cup \\
(\text{stone-phi} (\text{Abs-regular} (-x)) \cap \text{up-filter} (\text{Abs-dense} (y \sqcup -y)) \cup ((\text{up-filter} (\text{Abs-dense} (x \sqcup -x)) \cap \text{stone-phi} (\text{Abs-regular} (-y))) \cup (\text{up-filter} (\text{Abs-dense} (x \sqcup -y)))))\)

by (metis (no-types) inf-sup-distrib1 inf-sup-distrib2)
also have \(\ldots \leq (\text{stone-phi} (\text{Abs-regular} (-x)) \cap \text{stone-phi} (\text{Abs-regular} (-y)))\)
\(\cup \text{up-filter} (\text{Abs-dense} (y \sqcup -y \sqcup x)) \cup ((\text{up-filter} (\text{Abs-dense} (x \sqcup -x)) \cap \text{stone-phi} (\text{Abs-regular} (-y))) \cup (\text{up-filter} (\text{Abs-dense} (x \sqcup -y)) \cap \text{up-filter} (\text{Abs-dense} (y \sqcup -y))))\)

by (meson sup-left-isotone sup-right-isotone stone-phi-inf-dense)
also have \(\ldots \leq (\text{stone-phi} (\text{Abs-regular} (-x)) \cap \text{stone-phi} (\text{Abs-regular} (-y)))\)
\(\cup \text{up-filter} (\text{Abs-dense} (y \sqcup -y \sqcup x)) \cup \text{up-filter} (\text{Abs-dense} (x \sqcup -x \sqcup y)) \cup \\
(\text{up-filter} (\text{Abs-dense} (x \sqcup -x)) \cap \text{up-filter} (\text{Abs-dense} (y \sqcup -y))))\)

by (metis inf.commute sup-left-isotone sup-right-isotone stone-phi-inf-dense)
also have \(\ldots \leq (\text{stone-phi} (\text{Abs-regular} (-x)) \cap \text{stone-phi} (\text{Abs-regular} (-y)))\)
\(\cup \text{up-filter} (\text{Abs-dense} (y \sqcup -y \sqcup x)) \cup \text{up-filter} (\text{Abs-dense} (x \sqcup -x \sqcup y))\)

using 7 by (simp add: sup.absorb1 sup-commute sup-left-commute)
also have \(\ldots = (\text{stone-phi} (\text{Abs-regular} (-x)) \cap \text{stone-phi} (\text{Abs-regular} (-y)))\)
\(\cup \text{up-filter} (\text{Abs-dense} (x \sqcup y \sqcup -(x \sqcup y)))\)

using 8 by ( simp add: sup.commute sup.left-commute)
finally have \(\text{stone-phi} (\text{Abs-regular} (-x)) \cup \text{up-filter} (\text{Abs-dense} (x \sqcup -x))\)
\(\cap (\text{stone-phi} (\text{Abs-regular} (-y)) \cup \text{up-filter} (\text{Abs-dense} (y \sqcup -y))) = \ldots\)

using 9 using antisym by blast
also have \(\ldots = \text{stone-phi} (\text{Abs-regular} (-x) \cap \text{Abs-regular} (-y)) \cup \text{up-filter} (\text{Abs-dense} (x \sqcup y \sqcup -(x \sqcup y)))\)

by (simp add: stone-phi.hom)
also have \(\ldots = \text{stone-phi} (\text{Abs-regular} (-x \sqcup y)) \cup \text{up-filter} (\text{Abs-dense} (x \sqcup y \sqcup -(x \sqcup y)))\)

using 6 by (subst inf-regular.abs-eq) (simp-all add: eq-omp-same-arys)
finally have 10: \(\text{stone-phi} (\text{Abs-regular} (-x \sqcup y)) \cup \text{up-filter} (\text{Abs-dense} (x \sqcup y \sqcup -(x \sqcup y))) = (\text{stone-phi} (\text{Abs-regular} (-x)) \cup \text{up-filter} (\text{Abs-dense} (x \sqcup -x))) \cap (\text{stone-phi} (\text{Abs-regular} (-y)) \cup \text{up-filter} (\text{Abs-dense} (y \sqcup -y)))\)

by simp
have \(\text{Abs-regular} (-x \sqcup y) = \text{Abs-regular} (-x) \cup \text{Abs-regular} (-y)\)
using 6 by (subst sup-regular.abs-eq) (simp-all add: eq-omp-same-arys)

hence \(\text{Abs-stone-phi-pair} (\text{Abs-regular} (-x \sqcup y)), \text{stone-phi} (\text{Abs-regular} (-x \sqcup y)) \cup \text{up-filter} (\text{Abs-dense} (x \sqcup y \sqcup -(x \sqcup y))) = \text{Abs-stone-phi-pair} (\text{Abs-regular} (-x), \text{stone-phi} (\text{Abs-regular} (-x)) \cup \text{up-filter} (\text{Abs-dense} (x \sqcup -x))) \cap \text{Abs-regular} (-y), \text{stone-phi} (\text{Abs-regular} (-y)) \cup \text{up-filter} (\text{Abs-dense} (y \sqcup -y)))\)

using 10 by auto
also have \(\ldots = \text{Abs-stone-phi-pair} (\text{Abs-regular} (-x), \text{stone-phi} (\text{Abs-regular} (-x)) \cup \text{up-filter} (\text{Abs-dense} (x \sqcup -y))) \cup \text{Abs-stone-phi-pair} (\text{Abs-regular} (-y), \text{stone-phi} (\text{Abs-regular} (-y)) \cup \text{up-filter} (\text{Abs-dense} (y \sqcup -y)))\)

by (rule sup-stone-phi-pair.abs-eq[THEN sym]) (simp-all add: eq-omp-same-arys sa-iso-triple-pair)
finally show sa-iso \((x \sqcup y) = sa-iso x \sqcup sa-iso y\)

qed
next

have 1: \( \forall x y . 'a . \text{dense } (x \sqcup -x \sqcup y) \)
  by simp

have 2: \( \forall x . 'a . \text{in-p-image } (-x) \)
  by auto

have 3: \( \forall x y . 'a . \text{stone-phi } (\text{Abs-regular } (-y)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x)) = \text{stone-phi } (\text{Abs-regular } (-y)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x \sqcup -y)) \)
  proof (intro allI)
    fix \( x y :: 'a \)
    have \( \overline{\text{up-filter } (\text{Abs-dense } (x \sqcup -x)) \leq \text{stone-phi } (\text{Abs-regular } (-y)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x \sqcup -y))} \)
      by (metis (no-types, lifting) complement-shunting stone-phi-inf-dense stone-phi-complement complement-symmetric)
    have \( \overline{\text{up-filter } (\text{Abs-dense } (x \sqcup -x)) \leq \text{up-filter } (\text{Abs-dense } (x \sqcup -x))} \)
      by (metis sup-idem up-filter-dense-antitone)
    thus \( \text{stone-phi } (\text{Abs-regular } (-y)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x)) = \text{stone-phi } (\text{Abs-regular } (-y)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x \sqcup -y)) \)
      using 4 by (simp add: le-iff-sup sup-commute sup-left-commute)

qed

show \( \forall x y . 'a . \text{sa-iso } (x \sqcap y) = \text{sa-iso } x \sqcap \text{sa-iso } y \)
  proof (intro allI)
    fix \( x y :: 'a \)
    have \( \overline{\text{Abs-dense } ((x \sqcap y) \sqcup -(x \sqcap y)) = \text{Abs-dense } ((x \sqcup -x \sqcup -y) \sqcap (y \sqcup -y \sqcup -x))} \)
      by (simp add: sup-commute sup-inf-distrib1 sup-left-commute)
    also have \( \ldots = \text{Abs-dense } (x \sqcup -x \sqcup -y) \sqcap \text{Abs-dense } (y \sqcup -y \sqcup -x) \)
      using 7 by (metis (mono-tags, lifting) Abs-dense-inverse Rep-dense-inverse inf-dense,rep-eq mem-Collect-eq)
    finally have 5: \( \text{up-filter } (\text{Abs-dense } ((x \sqcap y) \sqcup -(x \sqcap y))) = \text{up-filter } (\text{Abs-dense } (x \sqcup -x \sqcup -y) \sqcup \text{up-filter } (\text{Abs-dense } (y \sqcup -y \sqcup -x))) \)
      by (simp add: up-filter-dist-inf)
    have \( \overline{\text{stone-phi } (\text{Abs-regular } (-x)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x)) \sqcup \text{stone-phi } (\text{Abs-regular } (-y)) \sqcup \text{up-filter } (\text{Abs-dense } (y \sqcup -y)) = (\text{stone-phi } (\text{Abs-regular } (-y)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x))) \sqcup (\text{stone-phi } (\text{Abs-regular } (-x)) \sqcup \text{up-filter } (\text{Abs-dense } (y \sqcup -y)))} \)
      by (simp add: inf-sup-aci(6) sup-left-commute)
    also have \( \ldots = (\text{stone-phi } (\text{Abs-regular } (-y)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x \sqcup -y))) \sqcup (\text{stone-phi } (\text{Abs-regular } (-x)) \sqcup \text{up-filter } (\text{Abs-dense } (y \sqcup -y \sqcup -x))) \)
      using 3 by simp
    also have \( \ldots = (\text{stone-phi } (\text{Abs-regular } (-x)) \sqcup \text{stone-phi } (\text{Abs-regular } (-y))) \sqcup \text{up-filter } (\text{Abs-dense } ((x \sqcap y) \sqcup -(x \sqcap y))) \)
      by (simp add: up-sup-aci(6) sup-left-commute)
    also have \( \ldots = (\text{stone-phi } (\text{Abs-regular } (-x)) \sqcup \text{stone-phi } (\text{Abs-regular } (-y))) \sqcup \text{up-filter } (\text{Abs-dense } ((x \sqcap y) \sqcup -(x \sqcap y))) \)
      by (simp add: sup-commute sup-left-commute)
    finally have \( \overline{\text{stone-phi } (\text{Abs-regular } (-x)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x)) \sqcup \text{stone-phi } (\text{Abs-regular } (-y)) \sqcup \text{up-filter } (\text{Abs-dense } (y \sqcup -y)) = \ldots} \)
      by simp
    also have \( \ldots = \text{stone-phi } (\text{Abs-regular } (-x) \sqcup \text{Abs-regular } (-y)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x \sqcup -y)) \)

\( (\text{Abs-dense } ((x \sqcap y) \sqcup -(x \sqcap y))) \)

by (simp add: stone-phi.hom)

also have \( \ldots = \text{stone-phi } (\text{Abs-regular }(-(x \sqcap y))) \sqcup \text{up-filter } (\text{Abs-dense } ((x \sqcap y) \sqcup -(x \sqcap y))) \)

using 2 by (subst sup-regular.abs-eq) (simp-all add: eq-onp-same-args)

finally have 6: \( \text{stone-phi } (\text{Abs-regular }(-(x \sqcap y))) \sqcup \text{up-filter } (\text{Abs-dense } ((x \sqcap y) \sqcup -(x \sqcap y))) = (\text{stone-phi } (\text{Abs-regular }(-x)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x))) \sqcup (\text{stone-phi } (\text{Abs-regular }(-y)) \sqcup \text{up-filter } (\text{Abs-dense } (y \sqcup -y))) \)

by simp

have \( \text{Abs-regular }(-(x \sqcap y)) = \text{Abs-regular }(-x) \sqcap \text{Abs-regular }(-y) \)

using 2 by (subst inf-regular.abs-eq) (simp-all add: eq-onp-same-args)

\( \text{hence } \text{Abs-stone-phi-pair } (\text{Abs-regular }-(x \sqcap y)), \text{stone-phi } (\text{Abs-regular }-(x \sqcap y)) \sqcup \text{up-filter } (\text{Abs-dense } ((x \sqcap y) \sqcup -(x \sqcap y))) = \text{Abs-stone-phi-pair } (\text{Abs-regular }-x) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x)) \sqcup (\text{Abs-regular }(-y)) \sqcup \text{up-filter } (\text{Abs-dense } (y \sqcup -y))) \)

using 6 by auto

also have \( \ldots = \text{Abs-stone-phi-pair } (\text{Abs-regular }-x) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x)) \sqcup \text{Abs-stone-phi-pair } (\text{Abs-regular }(-y)) \sqcup \text{up-filter } (\text{Abs-dense } (y \sqcup -y)) \)

by (rule inf-stone-phi-pair.abs-eq[THEN sym]) (simp-all add: eq-onp-same-args sa-iso-triple-pair)

finally show \( \text{sa-iso } (x \sqcap y) = \text{sa-iso } x \sqcap \text{sa-iso } y \)

qed

next

show \( \forall x::'a . \text{sa-iso } (-x) = -\text{sa-iso } x \)

proof

fix \( x :: 'a \)

have \( \text{sa-iso } (-x) = \text{Abs-stone-phi-pair } (\text{Abs-regular }-x) \sqcup \text{up-filter } \text{top} \)

by (simp add: top-dense-def)

also have \( \ldots = \text{Abs-stone-phi-pair } (\text{Abs-regular }-x) \sqcup \text{up-filter } \text{top} \)

by (metis bot-filter.abs-eq sup-bot.right-neutral up-top)

also have \( \ldots = \text{Abs-stone-phi-pair } (\text{Abs-regular }-x) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup -x)) \)

by (subst minus-regular.abs-eq[THEN sym], unfold eq-onp-same-args) auto

also have \( \ldots = -\text{sa-iso } x \)

by (simp add: eq-onp-def sa-iso-triple-pair minus-stone-phi-pair.abs-eq)

finally show \( \text{sa-iso } (-x) = -\text{sa-iso } x \)

by simp

qed

next

show \( \text{bij } \text{sa-iso} \)

by (metis (mono-tags, lifting) sa-iso-left-invertible sa-iso-right-invertible invertible-bij[where g=sa-iso-inv])

qed
5.7 Triple Isomorphism

In this section we prove that the triple of the Stone algebra of a triple is isomorphic to the original triple. The notion of isomorphism for triples is described in [7]. It amounts to an isomorphism of Boolean algebras, an isomorphism of distributive lattices with a greatest element, and a commuting diagram involving the structure maps.

5.7.1 Boolean Algebra Isomorphism

We first define and prove the isomorphism of Boolean algebras. Because the Stone algebra of a triple is implemented as a lifted pair, we also lift the Boolean algebra.

typedef (overloaded) ('a,'b) lifted-boolean-algebra = {
  xf::('a::non-trivial-boolean-algebra,'b::distrib-lattice-top) phi ⇒ 'a . True 
  by simp
}

setup-lifting type-definition-lifted-boolean-algebra

instantiation lifted-boolean-algebra ::
  (non-trivial-boolean-algebra,distrib-lattice-top) boolean-algebra
begin

lift-definition sup-lifted-boolean-algebra :: ('a,'b) lifted-boolean-algebra ⇒ ('a,'b) lifted-boolean-algebra is λxf yf f . sup (xf f) (yf f) .

lift-definition inf-lifted-boolean-algebra :: ('a,'b) lifted-boolean-algebra ⇒ ('a,'b) lifted-boolean-algebra is λxf yf f . inf (xf f) (yf f) .

lift-definition minus-lifted-boolean-algebra :: ('a,'b) lifted-boolean-algebra ⇒ ('a,'b) lifted-boolean-algebra is λxf yf f . minus (xf f) (yf f) .

lift-definition uminus-lifted-boolean-algebra :: ('a,'b) lifted-boolean-algebra ⇒ ('a,'b) lifted-boolean-algebra is λxf yf f . uminus (xf f) .

lift-definition bot-lifted-boolean-algebra :: ('a,'b) lifted-boolean-algebra is λf . bot ..

lift-definition top-lifted-boolean-algebra :: ('a,'b) lifted-boolean-algebra is λf . top ..

lift-definition less-eq-lifted-boolean-algebra :: ('a,'b) lifted-boolean-algebra ⇒ ('a,'b) lifted-boolean-algebra ⇒ bool is λxf yf . ∀ f . less-eq (xf f) (yf f) .

lift-definition less-lifted-boolean-algebra :: ('a,'b) lifted-boolean-algebra ⇒ ('a,'b) lifted-boolean-algebra ⇒ bool is λxf yf . (∀ f . less-eq (xf f) (yf f)) ∧ ¬ (∀ f .

96
let less-eq (yf f) (xf f).

instance
  apply intro-classes
  apply (simp add: less-eq-lifted-boolean-algebra.rep-eq
  less-lifted-boolean-algebra.rep-eq)
  apply (simp add: less-eq-lifted-boolean-algebra.rep-eq)
  using less-eq-lifted-boolean-algebra.rep-eq order-trans apply fastforce
  apply (metis less-eq-lifted-boolean-algebra.rep-eq antisym ext
  Rep-lifted-boolean-algebra-inject)
  apply (simp add: inf-lifted-boolean-algebra.rep-eq
  less-eq-lifted-boolean-algebra.rep-eq)
  apply (simp add: inf-lifted-boolean-algebra.rep-eq
  less-eq-lifted-boolean-algebra.rep-eq)
  apply (simp add: sup-lifted-boolean-algebra.rep-eq
  less-eq-lifted-boolean-algebra.rep-eq)
  apply (simp add: sup-lifted-boolean-algebra.rep-eq
  less-eq-lifted-boolean-algebra.rep-eq)
  apply (simp add: sup-lifted-boolean-algebra.rep-eq
  sup-lifted-boolean-algebra.rep-eq)
  apply (simp add: sup-lifted-boolean-algebra.rep-eq
  sup-lifted-boolean-algebra.rep-eq)
  apply (simp add: sup-lifted-boolean-algebra.rep-eq
  sup-lifted-boolean-algebra.rep-eq)
  by (unfold Rep-lifted-boolean-algebra-inject[THEN sym]
  inf-lifted-boolean-algebra.rep-eq inf-lifted-boolean-algebra.rep-eq, simp add: 
  unfold Rep-lifted-boolean-algebra-inject[THEN sym]
  inf-lifted-boolean-algebra.rep-eq inf-lifted-boolean-algebra.rep-eq, simp add: 
  unfold Rep-lifted-boolean-algebra-inject[THEN sym]
  inf-lifted-boolean-algebra.rep-eq inf-lifted-boolean-algebra.rep-eq, simp add: 
  unfold Rep-lifted-boolean-algebra-inject[THEN sym]
  inf-lifted-boolean-algebra.rep-eq inf-lifted-boolean-algebra.rep-eq, simp add: 
  diff-eq)

end

The following two definitions give the Boolean algebra isomorphism.

abbreviation ba-iso-inv :: ('a::non-trivial-boolean-algebra,'b::distrib-lattice-top) 
lifted-boolean-algebra ⇒ ('a,'b) lifted-pair regular
  where ba-iso-inv ≡ λxf . Abs-regular (Abs-lifted-pair (λf . 
  (Rep-lifted-boolean-algebra xf f,Rep-phi f (¬ Rep-lifted-boolean-algebra xf f))))

abbreviation ba-iso :: ('a::non-trivial-boolean-algebra,'b::distrib-lattice-top) 
lifted-pair regular ⇒ ('a,'b) lifted-boolean-algebra

97
where \( ba\text{-iso} \equiv \lambda pf \cdot \text{Abs-lifted-boolean-algebra} (\lambda f . \text{fst} (\text{Rep-lifted-pair} (\text{Rep-regular pf} f))) \)

**Lemma** \( ba\text{-iso-inv-lifted-pair} \):

\[
(\text{Rep-lifted-boolean-algebra} xf f, \text{Rep-phi f} (\lambda f . \text{Rep-lifted-boolean-algebra} xf f)) \in \text{triple.pairs} (\text{Rep-phi f})
\]

by (metis (no-types, hide-lams) double-compl simp-phi triple.pairs-uminus simp triple.pairs-uminus-closed)

**Lemma** \( ba\text{-iso-inv-regular} \):

\[
\text{regular} (\lambda f . (\text{Rep-lifted-boolean-algebra} xf f, \text{Rep-phi f} (\lambda f . \text{Rep-lifted-boolean-algebra} xf f)))
\]

**Proof** –

have \( \forall f . (\text{Rep-lifted-boolean-algebra} xf f, \text{Rep-phi f} (\lambda f . \text{Rep-lifted-boolean-algebra} xf f)) = \text{triple.pairs-uminus} (\text{Rep-phi f}) (\text{triple.pairs-uminus} (\text{Rep-phi f})) (\lambda f . (\text{Rep-lifted-boolean-algebra} xf f, \text{Rep-phi f} (\lambda f . \text{Rep-lifted-boolean-algebra} xf f))) \)

by (simp add: triple.pairs-uminus.simps triple-def)

hence \( \text{abs-lifted-pair} (\lambda f . (\text{Rep-lifted-boolean-algebra} xf f, \text{Rep-phi f} (\lambda f . \text{Rep-lifted-boolean-algebra} xf f))) = \lambda f . (\text{Rep-lifted-boolean-algebra} xf f, \text{Rep-phi f} (\lambda f . \text{Rep-lifted-boolean-algebra} xf f)) \)

by (simp add: triple.pairs-uminus-closed triple-def eq-onp-def uminus-lifted-pair.abs-eq ba-iso-inv-lifted-pair)

thus \(?\text{thesis}\) by simp

**Qed**

The following two results prove that the isomorphisms are mutually inverse.

**Lemma** \( ba\text{-iso-left-invertible} \):

\[
ba\text{-iso-inv} (ba\text{-iso pf}) = pf
\]

**Proof** –

have \( \forall f . \text{snd} (\text{Rep-lifted-pair} (\text{Rep-regular pf} f)) = \text{Rep-phi f} (\lambda f . \text{Rep-lifted-pair} (\text{Rep-regular pf} f)) \)

**Proof**

fix \( f :: (\text{'}a,\text{'}b) \text{phi} \)

let \( ?r = \text{Rep-phi f} \)

have \( \text{triple} ?r \)

by (simp add: triple-def)

hence \( \forall p . \text{triple.pairs-uminus} ?r p = (\text{fst} p, ?r (\text{fst} p)) \)

by (metis prod.collapse triple.pairs-uminus.simps)

have \( \text{Rep-regular pf} = \lambda f . \text{Rep-regular pf} \)

by (simp add: regular-in-p-image-iff)

show \( \text{snd} (\text{Rep-lifted-pair} (\text{Rep-regular pf} f)) = ?r (\lambda f . \text{Rep-lifted-pair} (\text{Rep-regular pf} f)) \)

using 2 3 by (metis fstI sndI uminus-lifted-pair.rep-eq)

**Qed**

have \( ba\text{-iso-inv} (ba\text{-iso pf}) = \text{Abs-regular} (\lambda f . (\text{fst} (\text{Rep-lifted-pair} (\text{Rep-regular pf} f)), \text{Rep-phi f} (\lambda f . \text{Rep-lifted-pair} (\text{Rep-regular pf} f)))) \)
by (simp add: Abs-lifted-boolean-algebra-inverse)
also have ... = Abs-regular (Abs-lifted-pair (Rep-lifted-pair (Rep-regular pf)))
  using 1 by (metis prod.collapse)
also have ... = pf
  by (simp add: Rep-regular-inverse Rep-lifted-pair-inverse)
finally show ?thesis
.
qed

lemma ba-iso-right-invertible:
  ba-iso (ba-iso-inv xf) = xf
proof -
  let \(?rf\) = Rep-lifted-boolean-algebra xf
  have 1: \(\forall f\ .\ (-\ ?rf\ f,Rep-\phi\ f (\ ?rf\ f)) \in\ triple.pairs\ (Rep-\phi\ f) \land (\ ?rf\ f,Rep-\phi\ f (-\ ?rf\ f)) \in\ triple.pairs\ (Rep-\phi\ f)\)
  proof
    fix f
    have up-filter top = bot
      by (simp add: bot-filter.abs-eq)
    hence (\(\exists z\ .\ Rep-\phi\ f (\ ?rf\ f) = Rep-\phi\ f (\ ?rf\ f) \sqcup up-filter\ z\)) \land (\(\exists z\ .\ Rep-\phi\ f (-\ ?rf\ f) = Rep-\phi\ f (-\ ?rf\ f) \sqcup up-filter\ z\))
      by (metis sup-bot-right)
    thus (\(-\ ?rf\ f,Rep-\phi\ f (\ ?rf\ f)) \in\ triple.pairs\ (Rep-\phi\ f) \land (\ ?rf\ f,Rep-\phi\ f (-\ ?rf\ f)) \in\ triple.pairs\ (Rep-\phi\ f)\)
      by (simp add: triple-def triple.pairs-def)
  qed
  have regular (Abs-lifted-pair (\(\lambda f\ .\ (?rf\ f,Rep-\phi\ f (-\ ?rf\ f))\)))
  proof -
    have \(-\ Abs-lifted-pair (\(\lambda f\ .\ (?rf\ f,Rep-\phi\ f (-\ ?rf\ f))\)) = \(-\ Abs-lifted-pair (\(\lambda f\ .\ triple.pairs-uminus\) (Rep-\phi\ f) (\ ?rf\ f,Rep-\phi\ f (-\ ?rf\ f))\))
      using 1 by (simp add: eq-onp-same-args uminus-lifted-pair.abs-eq)
    also have ... = \(-\ Abs-lifted-pair (\(\lambda f\ .\ (?rf\ f,Rep-\phi\ f (-\ ?rf\ f))\))
      by (metis (no-types, lifting) simp-phi triple-def triple.pairs-uminus.simps)
    also have ... = Abs-lifted-pair (\(\lambda f\ .\ triple.pairs-uminus\) (Rep-\phi\ f) (-\ ?rf\ f,Rep-\phi\ f (\ ?rf\ f)))
      using 1 by (simp add: eq-onp-same-args uminus-lifted-pair.abs-eq)
    also have ... = Abs-lifted-pair (\(\lambda f\ .\ (?rf\ f,Rep-\phi\ f (-\ ?rf\ f))\))
      by (metis (no-types, lifting) simp-phi triple-def triple.pairs-uminus.simps double-compl)
  finally show ?thesis
    by simp
  qed
  hence in-p-image (Abs-lifted-pair (\(\lambda f\ .\ (?rf\ f,Rep-\phi\ f (-\ ?rf\ f))\)))
    by blast
  thus ?thesis
    using 1 by (simp add: Rep-lifted-boolean-algebra-inverse Abs-lifted-pair-inverse Abs-regular-inverse)
  qed

The isomorphism is established by proving the remaining Boolean alge-
bra homomorphism properties.

**lemma** ba-iso:

boolean-algebra-isomorphism ba-iso

**proof** (intro conjI)

**show** Abs-lifted-boolean-algebra \((\lambda f\ .\ fst\ (Rep\text{-lifted-pair}\ (Rep\text{-regular}\ bot\ f))) =\) bot

by (simp add: bot-lifted-boolean-algebra-def bot-regular.rep-eq
bot-lifted-pair.rep-eq)

next

**show** Abs-lifted-boolean-algebra \((\lambda f\ .\ fst\ (Rep\text{-lifted-pair}\ (Rep\text{-regular}\ top\ f))) =\) top

by (simp add: top-lifted-boolean-algebra-def top-regular.rep-eq
top-lifted-pair.rep-eq)

next

**show** \(\forall\ pf\ qf\ .\ Abs\text{-lifted-boolean-algebra}\ (\lambda f::(a,b)\ phi\ :\ fst\ (Rep\text{-lifted-pair}\ (Rep\text{-regular}\ (pf\ \sqcup\ qf)\ f))) = Abs\text{-lifted-boolean-algebra}\ (\lambda f\ :\ fst\ (Rep\text{-lifted-pair}\ (Rep\text{-regular}\ qf)\ f))\) \sqcap Abs\text{-lifted-boolean-algebra}\ (\lambda f\ :\ fst\ (Rep\text{-lifted-pair}\ (Rep\text{-regular}\ qf)\ f))

**proof** (intro allI)

fix pf qf :: (a,b) lifted-pair regular

{ 
fix f

obtain x y z w where 1: \((x,y) = Rep\text{-lifted-pair}\ (Rep\text{-regular}\ pf)\ f\ \land\ (z,w) = Rep\text{-lifted-pair}\ (Rep\text{-regular}\ qf)\ f\)

using prod.collapse by blast

have triple \((Rep\text{-phi}\ f)\)

by (simp add: triple-def)

hence \(fst\ (\text{triple.pairs-sup}\ (x,y)\ (z,w)) = \text{fst}\ (x,y)\ \sqcup\ \text{fst}\ (z,w)\)

using triple.pairs-sup.simps by force

hence \(fst\ (\text{triple.pairs-sup}\ (Rep\text{-lifted-pair}\ (Rep\text{-regular}\ pf)\ f))\)

\((Rep\text{-lifted-pair}\ (Rep\text{-regular}\ qf)\ f)) = \text{fst}\ (\text{Rep\text{-lifted-pair}\ (Rep\text{-regular}\ pf)\ f})\ \sqcup\ \text{fst}\ (\text{Rep\text{-lifted-pair}\ (Rep\text{-regular}\ qf)\ f})\)

using 1 by simp

hence \(\text{fst}\ (\text{Rep\text{-lifted-pair}\ (Rep\text{-regular}\ (pf\ \sqcup\ qf))\ f}) = \text{fst}\ (\text{Rep\text{-lifted-pair}\ (Rep\text{-regular}\ pf)\ f})\ \sqcup\ \text{fst}\ (\text{Rep\text{-lifted-pair}\ (Rep\text{-regular}\ qf)\ f})\)

by (unfold sup-regular.rep-eq sup-lifted-pair.rep-eq simp)

}

thus \(Abs\text{-lifted-boolean-algebra}\ (\lambda f\ :\ fst\ (Rep\text{-lifted-pair}\ (Rep\text{-regular}\ (pf\ \sqcup\ qf))\ f))) = Abs\text{-lifted-boolean-algebra}\ (\lambda f\ :\ fst\ (Rep\text{-lifted-pair}\ (Rep\text{-regular}\ pf)\ f))\)

\(\sqcap Abs\text{-lifted-boolean-algebra}\ (\lambda f\ :\ fst\ (Rep\text{-lifted-pair}\ (Rep\text{-regular}\ qf)\ f))\)

by (simp add: eq-onp-same-args sup-lifted-boolean-algebra.abs-eq sup-regular.rep-eq sup-lifted-boolean-algebra.rep-eq)

qed

next

**show** \(\forall\ pf\ qf\ .\ Abs\text{-lifted-boolean-algebra}\ (\lambda f::(a,b)\ phi\ :\ fst\ (Rep\text{-lifted-pair}\ (Rep\text{-regular}\ (pf\ \sqcup\ qf))\ f))) = Abs\text{-lifted-boolean-algebra}\ (\lambda f\ :\ fst\ (Rep\text{-lifted-pair}\ (Rep\text{-regular}\ pf)\ f))\ \sqcap Abs\text{-lifted-boolean-algebra}\ (\lambda f\ :\ fst\ (Rep\text{-lifted-pair}\ (Rep\text{-regular}\ qf)\ f))\)

**proof** (intro allI)
\[
\text{fix } pf qf :: ('a,'b) lifted-pair regular
\{
\text{fix } f
\begin{align*}
& \text{obtain } x y z w \text{ where } I: (x,y) = \text{Rep-lifted-pair (Rep-regular } pf ) f \land (z,w) \\
& = \text{Rep-lifted-pair (Rep-regular } qf ) f
\end{align*}
\text{using prod.collapse by blast}
\begin{align*}
& \text{have triple (Rep-phi } f )
\end{align*}
\text{by (simp add: triple-def)}
\begin{align*}
& \text{hence } fst (\text{triple.pairs-inf } (x,y) (z,w)) = \text{fst } (x,y) \cap \text{fst } (z,w)
\end{align*}
\text{using triple.pairs-inf.simps by force}
\begin{align*}
& \text{hence } fst (\text{triple.pairs-inf } (\text{Rep-lifted-pair (Rep-regular } pf ) f )) = \text{fst } (\text{Rep-lifted-pair (Rep-regular } pf ) f ) \cap \text{fst } (\text{Rep-lifted-pair (Rep-regular } qf ) f )
\end{align*}
\text{using I by simp}
\begin{align*}
& \text{thus } \text{Abs-lifted-boolean-algebra } (\lambda f . \text{fst } (\text{Rep-lifted-pair (Rep-regular } pf \cap qf ) f )) = \text{Abs-lifted-boolean-algebra } (\lambda f . \text{fst } (\text{Rep-lifted-pair (Rep-regular } pf ) f ))
\end{align*}
\begin{align*}
& \text{abs-lifted-boolean-algebra } (\lambda f . \text{fst } (\text{Rep-lifted-pair (Rep-regular } pf ) f ))
\end{align*}
\text{by (simp add: eq-onp-same-args inf-lifted-boolean-algebra.abs-eq inf-regular.rep-eq inf-lifted-boolean-algebra.rep-eq)}
\text{qed}
\end{align*}
\text{next}
\begin{align*}
& \text{show } \forall pf . \text{Abs-lifted-boolean-algebra } (\lambda f . (\text{fst } (\text{Rep-lifted-pair (Rep-regular } (-pf ) f ))) = -\text{Abs-lifted-boolean-algebra } (\lambda f . \text{fst } (\text{Rep-lifted-pair (Rep-regular } pf ) f ))
\end{align*}
\text{proof}
\begin{align*}
& \text{fix } pf :: ('a,'b) lifted-pair regular
\begin{align*}
& \text{fix } f
\end{align*}
\begin{align*}
& \text{obtain } x y \text{ where } I: (x,y) = \text{Rep-lifted-pair (Rep-regular } pf ) f
\end{align*}
\text{using prod.collapse by blast}
\begin{align*}
& \text{have triple (Rep-phi } f )
\end{align*}
\text{by (simp add: triple-def)}
\begin{align*}
& \text{hence } fst (\text{triple.pairs-uminus } (\text{Rep-phi } f ) (x,y)) = -\text{fst } (x,y)
\end{align*}
\text{using triple.pairs-uminus.simps by force}
\begin{align*}
& \text{hence } fst (\text{triple.pairs-uminus } (\text{Rep-phi } f ) (\text{Rep-lifted-pair (Rep-regular } pf ) f )) = -\text{fst } (\text{Rep-lifted-pair (Rep-regular } pf ) f )
\end{align*}
\text{using I by simp}
\begin{align*}
& \text{hence } fst (\text{Rep-lifted-pair (Rep-regular } (-pf ) f )) = -\text{fst } (\text{Rep-lifted-pair (Rep-regular } pf ) f )
\end{align*}
\text{by (unfold uminus-regular.rep-eq uminus-lifted-pair.rep-eq simp)}
\end{align*}
\text{thus } \text{Abs-lifted-boolean-algebra } (\lambda f . \text{fst } (\text{Rep-lifted-pair (Rep-regular } (-pf ) f ))) = -\text{Abs-lifted-boolean-algebra } (\lambda f . \text{fst } (\text{Rep-lifted-pair (Rep-regular } pf ) f ))
\text{by (simp add: eq-onp-same-args uminus-lifted-boolean-algebra.abs-eq uminus-regular.rep-eq uminus-lifted-boolean-algebra.rep-eq)}
\]
5.7.2 Distributive Lattice Isomorphism

We carry out a similar development for the isomorphism of distributive lattices. Again, the original distributive lattice with a greatest element needs to be lifted to match the lifted pairs.

typedef (overloaded) ('a,'b) lifted-distrib-lattice-top = {
  xf:('a::non-trivial-boolean-algebra,'b::distrib-lattice-top) phi => 'b . True }
  by simp

setup-lifting type-definition-lifted-distrib-lattice-top

instance lifted-distrib-lattice-top ::
  (non-trivial-boolean-algebra,distrib-lattice-top) distrib-lattice-top
  begin
    lift-definition sup-lifted-distrib-lattice-top :: ('a,'b) lifted-distrib-lattice-top =>
      ('a,'b) lifted-distrib-lattice-top => ('a,'b) lifted-distrib-lattice-top is 
      λxf yf f . sup (xf f) (yf f) .

    lift-definition inf-lifted-distrib-lattice-top :: ('a,'b) lifted-distrib-lattice-top =>
      ('a,'b) lifted-distrib-lattice-top => ('a,'b) lifted-distrib-lattice-top is 
      λxf yf f . inf (xf f) (yf f) .

    lift-definition top-lifted-distrib-lattice-top :: ('a,'b) lifted-distrib-lattice-top is 
      λf . top .

    lift-definition less-eq-lifted-distrib-lattice-top :: ('a,'b) lifted-distrib-lattice-top =>
      ('a,'b) lifted-distrib-lattice-top => bool is 
      λxf yf . ∀f . less-eq (xf f) (yf f) .

    lift-definition less-lifted-distrib-lattice-top :: ('a,'b) lifted-distrib-lattice-top =>
      ('a,'b) lifted-distrib-lattice-top => bool is 
      λxf yf . (∀f . less-eq (xf f) (yf f)) ∧ ¬
      (∀f . less-eq (yf f) (xf f)) .

instance
  apply intro-classes
  apply (simp add: less-eq-lifted-distrib-lattice-top.rep-eq
  less-lifted-distrib-lattice-top.rep-eq)
  apply (simp add: less-eq-lifted-distrib-lattice-top.rep-eq)
  using less-eq-lifted-distrib-lattice-top.rep-eq order-trans apply fastforce
  apply (metis less-eq-lifted-distrib-lattice-top.rep-eq antisym ext
  Rep-lifted-distrib-lattice-top.inject)
  apply (simp add: inf-lifted-distrib-lattice-top.rep-eq
  less-lifted-distrib-lattice-top.rep-eq)

qed
The following function extracts the least element of the filter of a dense pair, which turns out to be a principal filter. It is used to define one of the isomorphisms below.

fun get-dense :: ('a::non-trivial-boolean-algebra,'b::distrib-lattice-top) lifted-pair dense ⇒ ('a,'b) phi ⇒ 'b
where get-dense pf f = (SOME z. Rep-lifted-pair (Rep-dense pf) f = (top,up-filter z))

lemma get-dense-char:
Rep-lifted-pair (Rep-dense pf) f = (top,up-filter (get-dense pf f))
proof –
obtain x y where 1: (x,y) = Rep-lifted-pair (Rep-dense pf) f ∧ (x,y) ∈ triple.pairs (Rep-phi f) ∧ triple.pairs-uminus (Rep-phi f) (x,y) = triple.pairs-bot
by (metis bot-lifted-pair.rep-eq prod-collapse simp-dense simp-lifted-pairuminus-lifted-pair.rep-eq)

hence 2: x = top
by (simp add: triple.intro triple.pairs-uminus.simps dense-pp)

have triple (Rep-phi f)
by (simp add: triple-def)

hence ∃ z. y = Rep-phi f (∼x) ⊔ up-filter z
using 1 triple.pairs-def by blast

then obtain z where y = up-filter z
using 2 by auto

hence Rep-lifted-pair (Rep-dense pf) f = (top,up-filter z)
using 1 2 by simp
thus ?thesis
by (metis (mono-tags, lifting) tfl-some dense.simps)

qed

The following two definitions give the distributive lattice isomorphism.
abbreviation dl-iso-inv :: ('a::non-trivial-boolean-algebra,'b::distrib-lattice-top) lifted-distrib-lattice-top ⇒ ('a,'b) lifted-pair dense
  where dl-iso-inv ≡ λxf . Abs-dense (Abs-lifted-pair (λf . (top,up-filter (Rep-lifted-distrib-lattice-top xf f))))

abbreviation dl-iso :: ('a::non-trivial-boolean-algebra,'b::distrib-lattice-top) lifted-pair dense ⇒ ('a,'b) lifted-distrib-lattice-top
  where dl-iso ≡ λpf . Abs-lifted-distrib-lattice-top (get-dense pf)

lemma dl-iso-inv-lifted-pair:
  (top,up-filter (Rep-lifted-distrib-lattice-top xf f)) ∈ triple.pairs (Rep-phi f)
  by (metis (no-types, hide-lams) compl-bot-eq double-compl simp-phi sup-bot.left-neutral triple.sa-iso-pair triple-def)

lemma dl-iso-inv-dense:
  dense (Abs-lifted-pair (λf . (top,up-filter (Rep-lifted-distrib-lattice-top xf f))))
  proof
    have ∀f . triple.pairs-uminus (Rep-phi f) (top,up-filter (Rep-lifted-distrib-lattice-top xf f)) = triple.pairs-bot
      by (simp add: top-filter.abs-eq triple.pairs-uminus.simps triple-def)
    hence bot = -Abs-lifted-pair (λf . (top,up-filter (Rep-lifted-distrib-lattice-top xf f))
      by (simp add: eq-onp-def uminus-lifted-pair.abs-eq dl-iso-inv-lifted-pair bot-lifted-pair-def)
    thus ?thesis
    by simp
  qed

The following two results prove that the isomorphisms are mutually inverse.

lemma dl-iso-left-invertible:
  dl-iso-inv (dl-iso pf) = pf
  proof
    have dl-iso-inv (dl-iso pf) = Abs-dense (Abs-lifted-pair (λf . (top,up-filter (get-dense pf f))))
      by (metis Abs-lifted-distrib-lattice-top-inverse UNIV-I UNIV-def)
    also have ... = Abs-dense (Abs-lifted-pair (Rep-lifted-pair (Rep-dense pf)))
      by (metis get-dense-char)
    also have ... = pf
      by (simp add: Rep-dense-inverse Rep-lifted-pair-inverse)
    finally show ?thesis
  .
  qed

lemma dl-iso-right-invertible:
  dl-iso (dl-iso-inv xf) = xf
  proof
    let xf = Rep-lifted-distrib-lattice-top xf
    let pf = Abs-dense (Abs-lifted-pair (λf . (top,up-filter (?rf f))))
have 1: \( \forall f . \ (\text{top}, \text{up-filter} \ (\mathcal{R}f)) \in \text{triple.pairs} \ (\text{Rep-phi} f) \)
proof
  fix \( f :: (\mathcal{a}, \mathcal{b}) \) phi
  have triple (\text{Rep-phi} f)
    by (simp add: triple-def)
  thus \((\text{top}, \text{up-filter} \ (\mathcal{R}f)) \in \text{triple.pairs} \ (\text{Rep-phi} f) \)
    using triple.pairs-def by force
qed

have 2: dense \((\text{Abs-lifted-pair} \ (\lambda f . \ (\text{top}, \text{up-filter} \ (\mathcal{R}f))))\)
proof
  have \(-\text{Abs-lifted-pair} \ (\lambda f . \ (\text{top}, \text{up-filter} \ (\mathcal{R}f))) = \text{Abs-lifted-pair} \ (\lambda f . \ \text{triple.pairs-uminus} \ (\text{Rep-phi} f) \ (\text{top}, \text{up-filter} \ (\mathcal{R}f)))\)
    using 1 by (simp add: eq-onp-same-args uminus-lifted-pair.abs-eq)
  also have \(... = \text{Abs-lifted-pair} \ (\lambda f . \ (\text{bot}, \text{Rep-phi} f \ \text{top}))\)
    by (simp add: triple.pairs-uminus.simps triple-def)
  also have \(... = \text{Abs-lifted-pair} \ (\lambda f . \ \text{triple.pairs-bot})\)
    by (metis no-types hide-lams simp-phi triple.phi-top triple-def)
  also have \(... = \text{bot}\)
    by (simp add: bot-lifted-pair-def)
  finally show \(?\text{thesis}\)
    by simp
qed

have \(\text{get-dense} \ ?pf = \mathcal{R}f\)
proof
  fix \( f \)
  have \((\text{top}, \text{up-filter} \ (\text{get-dense} \ ?pf f)) = \text{Rep-lifted-pair} \ (\text{Rep-dense} \ ?pf) f\)
    by (metis get-dense-char)
  also have \(... = \text{Rep-lifted-pair} \ (\text{Abs-lifted-pair} \ (\lambda f . \ (\text{top}, \text{up-filter} \ (\mathcal{R}f)))) f\)
    using Abs-dense-inverse 2 by force
  also have \(... = \text{Abs-lifted-pair} \ (\lambda f . \ (\text{top}, \text{up-filter} \ (\mathcal{R}f)))\)
    using \( f \) by (simp add: Abs-lifted-pair-inverse)
  finally show \(\text{get-dense} \ ?pf f = \mathcal{R}f f\)
    using up-filter-injective by auto
qed

thus \(?\text{thesis}\)
  by (simp add: Rep-lifted-distrib-lattice-top-inverse)
qed

To obtain the isomorphism, it remains to show the homomorphism properties of lattices with a greatest element.

lemma \(\text{dl-iso}:\)
\(\text{bounded-lattice-top-isomorphism} \ \text{dl-iso} \)
proof (intro conjI)
  have \(\text{get-dense} \text{top} = (\lambda f::(\mathcal{a}, \mathcal{b}) \) phi . \text{top}\)
  proof
    fix \( f :: (\mathcal{a}, \mathcal{b}) \) phi
    have \(\text{Rep-lifted-pair} \ (\text{Rep-dense} \text{top}) f = (\text{top}, \text{Abs-filter} \ {\text{top}})\)
      by (simp add: top-dense.rep-eq top-lifted-pair.rep-eq)
    hence \(\text{up-filter} \ (\text{get-dense} \text{top} f) = \text{Abs-filter} \ {\text{top}}\)
by (metis prod.inject get-dense-char)
hence Rep-filter (up-filter (get-dense top f)) = {top}
  by (metis bot-filter.abs-eq bot-filter.rep-eq)
thus get-dense top f = top
  by (metis self-in-upset singletonD Abs-filter-inverse mem-Collect-eq up-filter)
qed
thus Abs-lifted-distrib-lattice-top (get-dense top f :: ('a', 'b) phi ⇒ 'b) = top
  by (metis top-lifted-distrib-lattice-top-def)

next
show ∀ pf qf :: ('a', 'b) lifted-pair dense. Abs-lifted-distrib-lattice-top (get-dense (pf ⊔ qf)) = Abs-lifted-distrib-lattice-top (get-dense pf) ⊔ Abs-lifted-distrib-lattice-top (get-dense qf)
proof (intro allI)
  fix pf qf :: ('a', 'b) lifted-pair dense
  have 1: Abs-lifted-distrib-lattice-top (get-dense pf) ⊔ Abs-lifted-distrib-lattice-top (get-dense qf) = (λf. get-dense pf f ⊔ get-dense qf f)
    by (simp add: eq-onp-same-args sup-lifted-distrib-lattice-top.abs-eq)
  have (λf. get-dense (pf ⊔ qf) f) = (λf. get-dense pf f ⊔ get-dense qf f)
    proof
      fix f
      have (top, up-filter (get-dense (pf ⊔ qf) f)) = Rep-lifted-pair (Rep-dense (pf ⊔ qf) f)
        by (metis get-dense-char)
      also have ... = triple.pairs-sup (Rep-lifted-pair (Rep-dense pf f) f)
        (Rep-lifted-pair (Rep-dense qf f) f)
        by (simp add: sup-lifted-pair.rep-eq sup-dense.rep-eq)
      also have ... = triple.pairs-sup (top, up-filter (get-dense pf f)) (top, up-filter (get-dense qf f))
        by (metis (no-types, lifting) calculation prod.simps(1) simp-phi triple.pairs-sup.simps triple-def)
      also have ... = (top, up-filter (get-dense pf f) ⊔ up-filter (get-dense qf f))
        by (metis up-filter-dist-sup)
      finally show get-dense (pf ⊔ qf) f = get-dense pf f ⊔ get-dense qf f
        using up-filter-injective by blast
    qed
  thus Abs-lifted-distrib-lattice-top (get-dense (pf ⊔ qf)) = Abs-lifted-distrib-lattice-top (get-dense pf) ⊔ Abs-lifted-distrib-lattice-top (get-dense qf)
    using 1 by metis
qed

next
show ∀ pf qf :: ('a', 'b) lifted-pair dense. Abs-lifted-distrib-lattice-top (get-dense (pf ⊓ qf)) = Abs-lifted-distrib-lattice-top (get-dense pf) ⊓ Abs-lifted-distrib-lattice-top (get-dense qf)
proof (intro allI)
  fix pf qf :: ('a', 'b) lifted-pair dense
have 1: Abs-lifted-distrib-lattice-top (get-dense pf) ∩
Abs-lifted-distrib-lattice-top (get-dense qf) = Abs-lifted-distrib-lattice-top (λf .
get-dense pf f ∩ get-dense qf f)
  by (simp add: eq-onp-same-args inf-lifted-distrib-lattice-top.abs-eq)
have (λf . get-dense (pf ∩ qf) f) = (λf . get-dense pf f ∩ get-dense qf f)
proof
  fix f
  have (top, up-filter (get-dense (pf ∩ qf) f)) = Rep-lifted-pair (Rep-dense (pf
∩ qf)) f
  by (metis get-dense-char)
  also have ... = triple.pairs-inf (Rep-lifted-pair (Rep-dense pf) f)
(Rep-lifted-pair (Rep-dense qf) f)
  by (simp add: inf-lifted-pair.rep-eq inf-dense.rep-eq)
  also have ... = triple.pairs-inf (top, up-filter (get-dense pf f)) (top, up-filter
(get-dense qf f))
  by (metis get-dense-char)
  also have ... = (top, up-filter (get-dense pf f) ⊔ up-filter (get-dense qf f))
  by (metis (no-types, lifting) calculation prod.simps(1) simp-phi)
triple.pairs-inf.simps triple-def
  also have ... = (top, up-filter (get-dense pf f ∩ get-dense qf f))
  by (metis up-filter-dist-inf)
  finally show get-dense (pf ∩ qf) f = get-dense pf f ∩ get-dense qf f
  using up-filter-injective by blast
qed
thus Abs-lifted-distrib-lattice-top (get-dense (pf ∩ qf)) =
Abs-lifted-distrib-lattice-top (get-dense pf) ∩ Abs-lifted-distrib-lattice-top
(get-dense qf)
  using 1 by metis
qed
next
show bij dl-iso
  by (rule invertible-bij[where g=dl-iso-inv]) (simp-all add:
dl-iso-left-invertible dl-iso-right-invertible)
qed

5.7.3 Structure Map Preservation

We finally show that the isomorphisms are compatible with the structure
maps. This involves lifting the distributive lattice isomorphism to filters of
distributive lattices (as these are the targets of the structure maps). To this
end, we first show that the lifted isomorphism preserves filters.

lemma phi-iso-filter:
  filter ((λqf::('a::non-trivial-boolean-algebra,'b::distrib-lattice-top) lifted-pair
dense . Rep-lifted-distrib-lattice-top (dl-iso qf) f) ' Rep-filter (stone-phi pf))
proof (rule filter-map-filter)
  show mono (λqf::('a::non-trivial-boolean-algebra,'b::distrib-lattice-top)
lifted-pair dense . Rep-lifted-distrib-lattice-top (dl-iso qf) f)
  by (metis (no-types, lifting) mono-def dl-iso le-iff-sup
sup-lifted-distrib-lattice-top.rep-eq)
next
  show \( \forall y. \text{Rep-lifted-distrib-lattice-top} (dl\text{-iso} qf) \ f \leq y \rightarrow (\exists rf. y \leq rf) \\\n\wedge y = \text{Rep-lifted-distrib-lattice-top} (dl\text{-iso} rf) f \) 
proof (intro allI, rule impI)
  fix \( qf :: ('a,'b) lifted-pair_dense \)
  fix \( y :: 'b \)
  assume 1: \( \text{Rep-lifted-distrib-lattice-top} (dl\text{-iso} qf) f \leq y \)
  let \( ?rf = \text{Abs-dense} (\text{Abs-lifted-pair} (\lambda g. \text{if } g = f \text{ then } (\text{top}, \text{up-filter} y) \text{ else } \text{Rep-lifted-pair} (\text{Rep-dense} qf) g)) \)
  have 2: \( \forall g. (\text{if } g = f \text{ then } (\text{top}, \text{up-filter} y) \text{ else } \text{Rep-lifted-pair} (\text{Rep-dense} qf) g) \in \text{triple.pairs} (\text{Rep-phi} g) \)
    by (metis \text{Abs-lifted-distrib-lattice-top-inverse} \text{dl-iso-inv-lifted-pair} \text{mem-Collect-eq} \text{simp-lifted-pair})
  hence \( \neg \text{Abs-lifted-pair} (\lambda g. \text{if } g = f \text{ then } (\text{top}, \text{up-filter} y) \text{ else } \text{Rep-lifted-pair} (\text{Rep-dense} qf) g) = \text{Abs-lifted-pair} (\lambda g. \text{triple.pairs-uminus} (\text{Rep-phi} g) \ (\text{if } g = f \text{ then } (\text{top}, \text{up-filter} y) \text{ else } \text{Rep-lifted-pair} (\text{Rep-dense} qf) g)) \)
    by (simp add: eq-onp-def \text{uminus-lifted-pair.abs-eq})
  also have \( \neg \neg \text{Abs-lifted-pair} (\lambda g. \text{if } g = f \text{ then } \text{triple.pairs-uminus} (\text{Rep-phi} g) \ (\text{top}, \text{up-filter} y) \text{ else } \text{triple.pairs-uminus} (\text{Rep-phi} g) \ (\text{Rep-lifted-pair} (\text{Rep-dense} qf) g)) \)
    by (simp add: if-distrib)
  also have \( \neg \neg \text{Abs-lifted-pair} (\lambda g. \text{if } g = f \text{ then } (\text{bot}, \text{top}) \text{ else } \text{triple.pairs-uminus} (\text{Rep-phi} g) \ (\text{Rep-lifted-pair} (\text{Rep-dense} qf) g)) \)
    by (simp add: triple.pairs-uminus.simps simp add: triple_def metis compl-top-eq simp-phi)
  also have \( \neg \neg \text{Abs-lifted-pair} (\lambda g. \text{if } g = f \text{ then } (\text{bot}, \text{top}) \text{ else } (\text{bot}, \text{top})) \)
    by (metis \text{bot-lifted-pair.rep-eq} simp dense top_filter.abs-eq \text{uminus-lifted-pair.rep-eq})
  also have \( \neg \neg \text{Abs-lifted-pair} (\lambda g. \text{if } g = f \text{ then } \text{bot} \text{ else } \text{bot}) \)
    by (simp add: bot-lifted-pair.abs-eq top_filter.abs-eq simp-phi)
  finally have 3: \( \text{Abs-lifted-pair} (\lambda g. \text{if } g = f \text{ then } (\text{top}, \text{up-filter} y) \text{ else } \text{Rep-lifted-pair} (\text{Rep-dense} qf) g) \in \text{dense-elements} \)
    by blast
  hence \( (\text{top}, \text{up-filter} \ (\text{get-dense} (\text{Abs-dense} (\text{Abs-lifted-pair} (\lambda g. \text{if } g = f \text{ then } (\text{top}, \text{up-filter} y) \text{ else } \text{Rep-lifted-pair} (\text{Rep-dense} qf) g)) f))) = \text{Rep-lifted-pair} (\text{Rep-dense} (\text{Abs-lifted-pair} (\lambda g. \text{if } g = f \text{ then } (\text{top}, \text{up-filter} y) \text{ else } \text{Rep-lifted-pair} (\text{Rep-dense} qf) g)) f) \)
    by (metis \text{mono-tags} lifting \text{get-dense-char})
  also have \( \neg \neg \text{Rep-lifted-pair} (\text{Abs-lifted-pair} (\lambda g. \text{if } g = f \text{ then } (\text{top}, \text{up-filter} y) \text{ else } \text{Rep-lifted-pair} (\text{Rep-dense} qf) g)) f \)
    using 3 by (simp add: Abs-dense-inverse)
  also have \( \neg \neg \text{Abs-lifted-pair} (\lambda g. \text{if } g = f \text{ then } (\text{top}, \text{up-filter} y) \text{ else } \text{Rep-lifted-pair} (\text{Rep-dense} qf) g) \)
    using 2 by (simp add: Abs-lifted-pair-inverse)
  finally have \( \text{get-dense} (\text{Abs-dense} (\text{Abs-lifted-pair} (\lambda g. \text{if } g = f \text{ then } (\text{top}, \text{up-filter} y) \text{ else } \text{Rep-lifted-pair} (\text{Rep-dense} qf) g)) f) = y \)
    using up_filter.injective by blast
  hence 4: \( \text{Rep-lifted-distrib-lattice-top} (dl\text{-iso} ?rf) f = y \)
    by (simp add: Abs-lifted-distrib-lattice-top-inverse)
fix \( g \)

have \( \text{Rep-lifted-distrib-lattice-top} (\text{dl-iso } qf) \; g \leq \text{Rep-lifted-distrib-lattice-top} (\text{dl-iso } ?rf) \; g \)

proof (cases \( g = f \))
  assume \( g = f \)
  thus \( \text{thesis} \)
    using 1 4 by simp

next
  assume 5: \( g \neq f \)
  have \( \text{(top,up-filter (get-dense } ?rf \, g)) = \text{Rep-lifted-pair} (\text{Rep-dense} (\text{Abs-dense} (\text{Abs-lifted-pair} (\lambda g . \text{ if } g = f \text{ then } \text{(top,up-filter } y) \text{ else } \text{Rep-lifted-pair} (\text{Rep-dense} qf) \, g)) \, g) \)
    by (metis (mono-tags, lifting) get-dense-char)
  also have \( \ldots = \text{Rep-lifted-pair} (\text{Abs-lifted-pair} (\lambda g . \text{ if } g = f \text{ then } \text{(top,up-filter } y) \text{ else } \text{Rep-lifted-pair} (\text{Rep-dense} qf) \, g)) \, g \)
    using 3 by (simp add: Abs-dense-inverse)
  also have \( \ldots = \text{Rep-lifted-pair} (\text{Rep-dense} qf) \, g \)
    using 2 5 by (simp add: Abs-lifted-pair-inverse)
  also have \( \ldots = (\text{top,up-filter (get-dense } qf \, g)) \)
    using get-dense-char by auto
  finally have \( \text{get-dense } ?rf \, g = \text{get-dense } qf \, g \)
    using up-filter-injective by blast
  thus \( \text{Rep-lifted-distrib-lattice-top} (\text{dl-iso } qf) \, g \leq \text{Rep-lifted-distrib-lattice-top} (\text{dl-iso } ?rf) \, g \)
    by (simp add: Abs-lifted-distrib-lattice-top-inverse)
qed

hence \( \text{Rep-lifted-distrib-lattice-top} (\text{dl-iso } qf) \leq \text{Rep-lifted-distrib-lattice-top} (\text{dl-iso } ?rf) \)
by (simp add: le-funI)

hence 6: \( \text{dl-iso } qf \leq \text{dl-iso } ?rf \)
by (simp add: le-funD less-eq-lifted-distrib-lattice-top.rep-eq)

hence \( qf \leq \ ?rf \)
by (metis (no-types, lifting) dl-iso sup-isomorphism-ord-isomorphism)

thus \( \exists \, rf . \ qf \leq \ ?rf \land y = \text{Rep-lifted-distrib-lattice-top} (\text{dl-iso } rf) \, f \)
using 4 by auto
qed

The commutativity property states that the same result is obtained in two ways by starting with a regular lifted pair \( pf \):

* apply the Boolean algebra isomorphism to the pair; then apply a structure map \( f \) to obtain a filter of dense elements; or,

* apply the structure map \( \text{stone-phi} \) to the pair; then apply the distributive lattice isomorphism lifted to the resulting filter.

lemma phi-iso:
\[ \text{Rep-\textit{phi} } f (\text{Rep-lifted-boolean-algebra (ba-iso pf) } f) = \text{filter-map} \\
(\lambda qf: ('a::non-trivial-boolean-algebra,'b::distrib-lattice-top) \text{lif\textit{ted-pair dense}} . \\
\text{Rep-lifted-distrib-lattice-top (dl-iso qf) } f (\text{stone-phi pf}) \]

**proof**

- let \( ?r = \text{Rep-\textit{phi} } f \)
- let \( ?\text{ppf} = \lambda g . \text{triple.pairs-uminus (Rep-\textit{phi} } g) (\text{Rep-lifted-pair (Rep-regular pf) } g) \)

  have 1: triple \( ?r \)
  
  by (simp add: triple-def)

  have 2: \( \text{Rep-filter} ( ?r (\text{fst (Rep-lifted-pair (Rep-regular pf) } f)))) \subseteq \{ z . \exists qf . \\
  -\text{Rep-regular pf } \leq \text{Rep-dense qf } \land z = \text{get-dense qf } f \} \)

  proof
  
  fix \( z \) obtain \( x \) where 3: \( x = \text{fst (Rep-lifted-pair (Rep-regular pf) } f) \)
  
  by simp

  assume \( z \in \text{Rep-filter} ( ?r (\text{fst (Rep-lifted-pair (Rep-regular pf) } f))) \)

  hence \( \uparrow z \subseteq \text{Rep-filter} ( ?r x) \)

  using 3 filter-def by fastforce

  hence 4: \( \text{up-filter} z \leq ?r x \)

  by (metis \text{Rep-filter-cases Rep-filter-inverse less-eq-filter}.

  mem-Collect-eq up-filter)

  have 5: \( \forall g . ?\text{ppf} g \in \text{triple.pairs (Rep-\textit{phi} } g) \)

  by (metis (no-types) simp-lifted-pair uminus-lifted-pair.

  rep-eq)

  let \( ?zf = \lambda g . \text{if } g = f \text{ then } (\text{top,up-filter } z) \text{ else triple.pairs-top} \)

  have 6: \( \forall g . ?zf g \in \text{triple.pairs (Rep-\textit{phi} } g) \)

  proof
  
  fix \( g :: ('a,'b) \text{ phi} \)

  have triple (Rep-\textit{phi} } g)

  by (simp add: triple-def)

  hence \( (\text{top,up-filter } z) \in \text{triple.pairs (Rep-\textit{phi} } g) \)

  using triple.pairs-def by force

  thus \( ?zf g \in \text{triple.pairs (Rep-\textit{phi} } g) \)

  by (metis simp-lifted-pair top-lifted-pair.rep-eq)

  qed

  hence \( -\text{Abs-lifted-pair } ?zf = \text{Abs-lifted-pair } (\lambda g . \text{triple.pairs-uminus (Rep-\textit{phi} } g) (\text{Rep-regular pf) } f)) \)

  by (subst uminus-lifted-pair.abs-eq) (simp-all add: eq-onp-same-args)

  also have \( ... = \text{Abs-lifted-pair } (\lambda g . \text{if } g = f \text{ then triple.pairs-uminus (Rep-\textit{phi} } g) \text{ triple.pairs-top} \)

  by (rule arg-cong [where \( f=\text{Abs-lifted-pair} \]) auto

  also have \( ... = \text{Abs-lifted-pair } (\lambda g . \text{triple.pairs-bot}) \)

  using \( f \) by (metis bot-lifted-pair.rep-eq dense-closed-top top-lifted-pair.rep-eq.

  triple.pairs-uminus.simps uminus-lifted-pair.rep-eq)

  finally have \( 7: \text{Abs-lifted-pair } ?zf \in \text{dense-elements} \)

  by (simp add: bot-lifted-pair.abs-eq)

  let \( ?gf = \text{Abs-dense } (\text{Abs-lifted-pair } ?zf) \)

  have \( \forall g . \text{triple.pairs-less-eq } (\text{?ppf } g) (\text{?zf } g) \)

  proof
  
  fix \( g \)
show \(\text{triple.pairs-less-eq} (\forall ppf \; g) (\forall zf \; g)\)
proof (cases \(g = f\))
  assume \(8: \; g = f\)
  hence \(9: \; \forall ppf \; g = (-x, \forall r \; x)\)
    using \(1 \; 3\) by (metis prod.collapse triple.pairs-uminus.simps)
  have \(\text{triple.pairs-less-eq} (-x, \forall r \; x) (\text{top}, \text{up-filter} \; z)\)
    using \(1 \; 4\) by (meson inf.botLeast triple.pairs-less-eq.simps)
  thus \(?\text{thesis}\)
    using \(8 \; 9\) by simp
next
  assume \(10: \; g \neq f\)
  have \(\text{triple.pairs-less-eq} (\forall ppf \; g) \; \text{triple.pairs-top}\)
    using \(1\) by (metis (no-types, hide-lams) bot.extremum top-greatest prod.collapse triple-def triple.pairs-less-eq.simps triple.phi-bot)
  thus \(?\text{thesis}\)
    using \(10\) by simp
qed

hence \(\text{Abs-lifted-pair} \; \forall ppf \leq \text{Abs-lifted-pair} \; \forall zf\)
using \(5 \; 6\) by (subst less-eq-lifted-pair.abs-eq) (simp-all add: eq-onp-same-args)

hence \(11: \; \neg \text{Rep-regular} \; pf \leq \text{Rep-dense} \; \forall qf\)
using \(7\) by (simp add: uminus-lifted-pair-def Abs-dense-inverse)

have \((\text{top}, \text{up-filter} \; (\neg\text{get-dense} \; \forall qf \; f)) = \text{Rep-lifted-pair} \; (\text{Rep-dense} \; \forall qf \; f)\)
by (metis get-dense-char)
also have \(... = (\text{top}, \text{up-filter} \; z)\)
using \(6 \; 7\) Abs-dense-inverse Abs-lifted-pair-inverse by force
finally have \(z = \text{get-dense} \; \forall qf \; f\)
using up-filter-injective by force
thus \(z \in \{ z . \exists qf . \neg \text{Rep-regular} \; pf \leq \text{Rep-dense} qf \land z = \text{get-dense} qf \; f \}\)
using \(11\) by auto

qed

have \(12: \; \text{Rep-filter} \; (\forall r \; (\text{fst} \; (\text{Rep-lifted-pair} \; (\text{Rep-regular} \; pf) \; f))) \supseteq \{ z . \exists qf . \neg \text{Rep-regular} \; pf \leq \text{Rep-dense} qf \land z = \text{get-dense} qf \; f \}\)
proof
fix \(z\)
assume \(z \in \{ z . \exists qf . \neg \text{Rep-regular} \; pf \leq \text{Rep-dense} qf \land z = \text{get-dense} qf \; f \}\)

hence \(\exists qf . \neg \text{Rep-regular} \; pf \leq \text{Rep-dense} qf \land z = \text{get-dense} qf \; f\)
by auto

hence \(\text{triple.pairs-less-eq} \; (\text{Rep-lifted-pair} \; (\neg \text{Rep-regular} \; pf) \; f) \; (\text{top}, \text{up-filter} \; z)\)
by (metis less-eq-lifted-pair.rep-eq get-dense-char)

hence \(\text{up-filter} \; z \leq \text{snd} \; (\text{Rep-lifted-pair} \; (\neg \text{Rep-regular} \; pf) \; f)\)
using \(1\) by (metis (no-types, hide-lams) prod.collapse triple.pairs-less-eq.simps)
also have \(\neg = \text{snd} \; (\forall ppf \; f)\)
by (metis uminus-lifted-pair.rep-eq)
also have \(\neg = \forall r \; (\text{fst} \; (\text{Rep-lifted-pair} \; (\text{Rep-regular} \; pf) \; f))\)
using \(1\) by (metis (no-types) prod.collapse prod.inject)
triple.pairs-uminus.simps)

finally have Rep-filter (up-filter z) ⊆ Rep-filter (?r (fst (Rep-lifted-pair (Rep-regular pf) f)))
  by (simp add: less-eq-filter.rep-eq)

hence ?z ⊆ Rep-filter (?r (fst (Rep-lifted-pair (Rep-regular pf) f)))
  by (metis Abs-filter-inverse mem-Collect-eq up-filter)

thus z ∈ Rep-filter (?r (fst (Rep-lifted-pair (Rep-regular pf) f)))
  by blast

have 13: ∀ qf ∈ Rep-filter (stone-phi pf). Rep-lifted-distrib-lattice-top (Abs-lifted-distrib-lattice-top (get-dense qf)) f = get-dense qf f
  by (metis Abs-lifted-distrib-lattice-top-inverse UNIV-I UNIV-def)

have Rep-filter (?r (fst (Rep-lifted-pair (Rep-regular pf) f))) = { z . ∃ qf ∈ stone-phi-set pf . z = get-dense qf f }
  using 2 12 by simp

hence ?r (fst (Rep-lifted-pair (Rep-regular pf) f)) = Abs-filter { z . ∃ qf ∈ stone-phi-set pf . z = get-dense qf f }
  by (metis Rep-filter-inverse)

hence ?r (Rep-lifted-boolean-algebra (ba-iso pf) f) = Abs-filter { z . ∃ qf ∈ Rep-filter (stone-phi pf) . z = Rep-lifted-distrib-lattice-top (dl-iso qf) f }

thus ?thesis
  by (simp add: image-def)

qed

end

References


