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 11

3.1 P-Algebras 12

3.1.1 Pseudocomplemented Lattices 12

3.1.2 Pseudocomplemented Distributive Lattices 18

3.2 Stone Algebras 19

3.3 Heyting Algebras 22

3.3.1 Heyting Semilattices 22

3.3.2 Heyting Lattices 25

3.3.3 Heyting Algebras 27

3.3.4 Brouwer Algebras 28

3.4 Boolean Algebras 28

4 Filters 30

4.1 Orders 31

4.2 Lattices 34

4.3 Distributive Lattices 38
5 Stone Construction

5.1 Triples ......................................................... 41
5.2 The Triple of a Stone Algebra ..................... 43
  5.2.1 Regular Elements .................................. 43
  5.2.2 Dense Elements .................................. 44
  5.2.3 The Structure Map ................................ 46
5.3 Properties of Triples .................................. 46
5.4 The Stone Algebra of a Triple .................... 49
5.5 The Stone Algebra of the Triple of a Stone Algebra 51
5.6 Stone Algebra Isomorphism ......................... 53
5.7 Triple Isomorphism ..................................... 54
  5.7.1 Boolean Algebra Isomorphism .................. 54
  5.7.2 Distributive Lattice Isomorphism ............... 56
  5.7.3 Structure Map Preservation ..................... 58

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 HOL/Algebra/Ideal.thy defines ring-theoretic ideals in locales with a carrier set. In the theory 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 HOL/Algebra/*.thy 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:
\( f (\text{if } c \text{ then } x \text{ else } y) (\text{if } c \text{ then } z \text{ else } w) = (\text{if } c \text{ then } f x z \text{ else } f y w) \)
⟨proof⟩

lemma left-invertible-inj:
\( (\forall x. g (f x) = x) \implies \text{inj } f \)
⟨proof⟩

lemma invertible-bij:
assumes \( \forall x. g (f x) = x \)
and \( \forall y. f (g y) = y \)
shows \( \text{bij } f \)
⟨proof⟩

lemma finite-ne-subset-induct [consumes 3, case-names singleton insert]:
assumes finite \( F \)
and \( F \neq \{\} \)
and \( F \subseteq S \)
and \( \text{singleton: } \forall x. P \{x\} \)
and \( \text{insert: } \forall x. \text{finite } F \implies F \neq \{\} \implies F \subseteq S \implies x \in S \implies x \notin F \)
\( \implies P F \implies P (\text{insert } x F) \)
shows \( P F \)
⟨proof⟩

lemma finite-set-of-finite-funs-pred:
assumes finite \( \{ x::'a . \text{True } \} \)
and finite \( \{ y::'b . P y \} \)
shows finite \( \{ f . (\forall x::'a . P (f x)) \} \)
⟨proof⟩

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 \( \neg\neg x \) instead of \( -(-x) \).

context sup
begin

notation sup \( \text{(infixl } \sqcup 65) \)
definition additive :: (\'a \Rightarrow \'a) \Rightarrow bool
  where additive f \equiv \forall x y . f (x \sqcup y) = f x \sqcup f y
end

context inf
begin

notation inf (infixl \cap 67)
end

context uminus
begin

no-notation uminus (\- [-81] 80)
notation uminus (\- [80] 80)
end

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

context ord
begin

definition isotone :: (\'a \Rightarrow \'a) \Rightarrow bool
  where isotone f \equiv \forall x y . x \leq y \rightarrow f x \leq f y

definition galois :: (\'a \Rightarrow \'a) \Rightarrow (\'a \Rightarrow \'a) \Rightarrow bool
  where galois l u \equiv \forall x y . l x \leq y \leftrightarrow x \leq u y

definition lifted-less-eq :: (\'a \Rightarrow \'a) \Rightarrow (\'a \Rightarrow \'a) \Rightarrow bool ((\leq\leq \leq) [51, 51] 50)
  where f \leq\leq g \equiv \forall x . f x \leq g x
end

context order
begin

lemma order-lesseq-imp:
  (\forall z . x \leq z \rightarrow y \leq z) \leftrightarrow y \leq x
  ⟨proof⟩

lemma galois-char:
  galois l u \leftrightarrow (\forall x . x \leq u (l x)) \land (\forall x . l (u x) \leq x) \land isotone l \land isotone u
  ⟨proof⟩
**lemma** galois-closure:
\[
galois l u \implies l x = l (u (l x)) \land u x = u (l (u x))
\]
⟨proof⟩

**lemma** lifted-reflexive:
\[
f = g \implies f \leq g
\]
⟨proof⟩

**lemma** lifted-transitive:
\[
f \leq g \implies g \leq h \implies f \leq h
\]
⟨proof⟩

**lemma** lifted-antisymmetric:
\[
f \leq g \implies g \leq f \implies f = g
\]
⟨proof⟩

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:
\[
\text{assumes finite } s
\]
and \( s \neq \emptyset \)
and \( \forall x \in s . \forall y \in s . f x \leq f y \lor f y \leq f x \)
\[
\text{shows } \exists m \in s . \forall z \in s . f m \leq f z
\]
⟨proof⟩

The following are basic facts in semilattices.

**context** semilattice-sup

**begin**

**lemma** sup-left-isotone:
\[
x \leq y \implies x \sqcup z \leq y \sqcup z
\]
⟨proof⟩

**lemma** sup-right-isotone:
\[
x \leq y \implies z \sqcup x \leq z \sqcup y
\]
⟨proof⟩

**lemma** sup-left-divisibility:
\[
x \leq y \iff (\exists z . x \sqcup z = y)
\]
⟨proof⟩

**lemma** sup-right-divisibility:
\[
x \leq y \iff (\exists z . z \sqcup x = y)
\]
⟨proof⟩

**lemma** sup-same-context:
\[
x \leq y \sqcup z \implies y \leq x \sqcup z \implies x \sqcup z = y \sqcup z
\]
lemma sup-relative-same-increasing:
$x \leq y \Rightarrow x \sqcup z = x \sqcup w \Rightarrow y \sqcup z = y \sqcup w$

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

end

context semilattice-inf
begin

lemma inf-same-context:
$x \leq y \sqcap z \Rightarrow y \leq x \sqcap z \Rightarrow x \sqcap z = y \sqcap z$

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: \exists z . x \leq z \land y \leq z

We extend the inf sublocale, which dualises the order in semilattices, to bounded semilattices.

context bounded-semilattice-inf-top
begin

subclass directed-semilattice-inf

end

context lattice
begin
subclass directed-semilattice-inf
  ⟨proof⟩

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.

class dense-lattice = bounded-lattice +
  assumes bot-meet-irreducible: x ∩ y = bot −→ x = bot ∨ y = bot
context distrib-lattice
begin

lemma relative-equality:
  x ⊔ z = y ⊔ z −→ x ∩ z = y ∩ z −→ x = y
  ⟨proof⟩
end

Distributive lattices with a greatest element are widely used in the con-
struction 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 ⟨proof⟩

8
subclass distrib-lattice-top ⟨proof⟩

lemma complement-shunting:
  assumes complement z w
  shows z \cap x \leq y \iff x \leq w \cup y
  ⟨proof⟩
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 automatically distributive.

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

class linear-lattice = lattice + linorder
begin

lemma max-sup:
  max x y = x \cup y
  ⟨proof⟩

lemma min-inf:
  min x y = x \cap y
  ⟨proof⟩

lemma sup-inf-selective:
  (x \cup y = x \land x \cap y = y) \lor (x \cup y = y \land x \cap y = x)
  ⟨proof⟩

lemma sup-selective:
  x \cup y = x \lor x \cup y = y
  ⟨proof⟩

lemma inf-selective:
  x \cap y = x \lor x \cap y = y
  ⟨proof⟩

subclass distrib-lattice
  ⟨proof⟩

lemma sup-less-eq:
  x \leq y \cup z \iff x \leq y \lor x \leq z
  ⟨proof⟩

lemma inf-less-eq:
  x \cap y \leq z \iff x \leq z \lor y \leq z
  ⟨proof⟩
**Lemma** \( \text{sup-inf-sup}: \)
\[
x \sqcup y = (x \sqcup y) \sqcup (x \sqcap y)
\]
⟨proof⟩

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 ⟨proof⟩

subclass bounded-linorder ⟨proof⟩

subclass bounded-distrib-lattice ⟨proof⟩

**Lemma** sup-dense:
\[
x \neq \text{top} \implies y \neq \text{top} \implies x \sqcup y \neq \text{top}
\]
⟨proof⟩

**Lemma** inf-dense:
\[
x \neq \text{bot} \implies y \neq \text{bot} \implies x \sqcap y \neq \text{bot}
\]
⟨proof⟩

**Lemma** sup-not-bot:
\[
x \neq \text{bot} \implies x \sqcup y \neq \text{bot}
\]
⟨proof⟩

**Lemma** inf-not-top:
\[
x \neq \text{top} \implies x \sqcap y \neq \text{top}
\]
⟨proof⟩

subclass dense-lattice ⟨proof⟩

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 ⟨proof⟩

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, $bot = top$ may hold in bounded lattices.

```plaintext
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:
  $bot \neq top$

⟨proof⟩

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.

```plaintext
class p-algebra = bounded-lattice + uminus +
  assumes pseudo-complement: \( x \sqcap y = \text{bot} \iff x \leq -y \)
begin

  Regular elements and dense elements are frequently used in pseudocomplemented algebras.

  abbreviation regular \( x \equiv x = --x \)
  abbreviation dense \( x \equiv -x = \text{bot} \)
  abbreviation complemented \( x \equiv \exists y, x \sqcap y = \text{bot} \land x \sqcup y = \text{top} \)
  abbreviation in-p-image \( x \equiv \exists y, x = -y \)
  abbreviation selection \( s x \equiv s = --s \sqcap x \)

  abbreviation dense-elements \( \equiv \{ x, \text{dense } x \} \)
  abbreviation regular-elements \( \equiv \{ x, \text{in-p-image } x \} \)

  lemma p-bot [simp]:
  \( -\text{bot} = \text{top} \)
  ⟨proof⟩

  lemma p-top [simp]:
  \( -\text{top} = \text{bot} \)
  ⟨proof⟩

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

  lemma inf-p [simp]:
  \( x \sqcap -x = \text{bot} \)
  ⟨proof⟩

  lemma p-inf [simp]:
  \( -x \sqcap x = \text{bot} \)
  ⟨proof⟩
```

12
lemma \( pp-inf-p \):
\[
-x \cap -x = \text{bot}
\]
(\text{proof})

The double complement is a closure operation.

lemma \( pp-increasing \):
\[
x \leq -x
\]
(\text{proof})

lemma \( ppp \ [simp] \):
\[
-\neg\neg x = -x
\]
(\text{proof})

lemma \( pp-idempotent \):
\[
-\neg\neg\neg x = -x
\]
(\text{proof})

lemma \( regular-in-p-image-iff \):
\[
\text{regular} \ x \leftrightarrow \text{in-p-image} \ x
\]
(\text{proof})

lemma \( pseudo-complement-pp \):
\[
x \cap y = \text{bot} \leftrightarrow -x \leq -y
\]
(\text{proof})

lemma \( p-antitone \):
\[
x \leq y \implies -y \leq -x
\]
(\text{proof})

lemma \( p-antitone-sup \):
\[
-(x \cup y) \leq -x
\]
(\text{proof})

lemma \( p-antitone-inf \):
\[
-x \leq -(x \cap y)
\]
(\text{proof})

lemma \( p-antitone-if \):
\[
x \leq -y \iff y \leq -x
\]
(\text{proof})

lemma \( pp-isotone \):
\[
x \leq y \implies -x \leq -y
\]
(\text{proof})

lemma \( pp-isotone-sup \):
\[
-\neg x \leq -\neg(x \cup y)
\]
(\text{proof})
lemma pp-isotone-inf:
\[-(x \cap y) \leq -x\]
(proof)

One of De Morgan’s laws holds in pseudocomplemented lattices.

lemma p-dist-sup [simp]:
\[-(x \cup y) = -x \cap -y\]
(proof)

lemma p-supdist-inf:
\[-x \cup -y \leq -(x \cap y)\]
(proof)

lemma pp-dist-pp-sup [simp]:
\[-(-x \cup -y) = -(x \cup y)\]
(proof)

lemma p-sup-p [simp]:
\[-(x \cup -x) = \text{bot}\]
(proof)

lemma pp-sup-p [simp]:
\[-(x \cup -x) = \text{top}\]
(proof)

lemma dense-pp:
\[\text{dense } x \iff -x = \text{top}\]
(proof)

lemma dense-sup-p:
\[\text{dense } (x \cup -x)\]
(proof)

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

lemma pp-inf-bot-iff:
\[x \cap y = \text{bot} \iff -x \cap y = \text{bot}\]
(proof)

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

lemma p-shunting-swap:
\[x \cap y \leq -z \iff x \cap z \leq -y\]
(proof)

lemma pp-inf-below-iff:
\[x \cap y \leq -z \iff -(x \cap y) \leq -z\]
proof

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

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

lemma regular-closed-inf:
regular \(x\) \implies regular \(y\) \implies regular \((x \cap y)\)
proof

lemma regular-closed-p:
regular \((-x)\)
proof

lemma regular-closed-pp:
regular \((-x)\)
proof

lemma regular-closed-bot:
regular bot
proof

lemma regular-closed-top:
regular top
proof

lemma pp-dist-inf [simp]:
\((x \cap y) = -x \cap -y\)
proof

lemma inf-import-p [simp]:
\(x \cap -(x \cap y) = x \cap -y\)
proof

Pseudocomplements are unique.

lemma p-unique:
\((\forall x. x \cap y = bot \iff x \leq z) \implies z = -y\)
proof

lemma maddux-3-5:
\(x \cup x = x \cup -(y \cup -y)\)
proof

lemma shunting-1-pp:
\(x \leq -y \iff x \cap -y = bot\)

proof

lemma
lemma pp-pp-inf-bot-iff:
\[ x \cap y = \bot \iff \neg x \cap \neg y = \bot \]
(\text{proof})

lemma inf-pp-semi-commute:
\[ x \cap \neg y \leq \neg (x \cap y) \]
(\text{proof})

lemma inf-pp-commute:
\[ \neg (\neg x \cap y) = \neg x \cap \neg y \]
(\text{proof})

lemma sup-pp-semi-commute:
\[ x \cup \neg y \leq \neg (x \cup y) \]
(\text{proof})

lemma regular-sup:
\[ \text{regular } z \implies (x \leq z \land y \leq z \iff \neg (x \cup y) \leq z) \]
(\text{proof})

lemma dense-closed-inf:
\[ \text{dense } x \implies \text{dense } y \implies \text{dense } (x \cap y) \]
(\text{proof})

lemma dense-closed-sup:
\[ \text{dense } x \implies \text{dense } y \implies \text{dense } (x \cup y) \]
(\text{proof})

lemma dense-closed-pp:
\[ \text{dense } x \implies \text{dense } (\neg \neg x) \]
(\text{proof})

lemma dense-closed-top:
\[ \text{dense } \top \]
(\text{proof})

lemma dense-up-closed:
\[ \text{dense } x \implies x \leq y \implies \text{dense } y \]
(\text{proof})

lemma regular-dense-top:
\[ \text{regular } x \implies \text{dense } x \implies x = \top \]
(\text{proof})

lemma selection-char:
\[ \text{selection } s \ x \iff (\exists y \ . \ s = \neg y \cap x) \]
(\text{proof})
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

Lemma conjugate-symmetric:
Conjugate \( f \) \( g \) \( \Rightarrow \) Conjugate \( g \) \( f \)
Proof

Lemma additive-isotone:
Additive \( f \) \( \Rightarrow \) Isotone \( f \)
Proof

Lemma dual-additive-antitone:
Assumes dual-additive \( f \)
Shows Isotone \( (\uminus \circ f) \)
Proof

Lemma conjugate-dual-additive:
Assumes conjugate \( f \) \( g \)
Shows dual-additive \( (\uminus \circ f) \)
Proof

Lemma conjugate-isotone-pp:
Conjugate \( f \) \( g \) \( \Rightarrow \) Isotone \( (\uminus \circ \uminus \circ f) \)
Proof

Lemma conjugate-char-1-pp:
Conjugate \( f \) \( g \) \( \leftrightarrow \) (\( \forall \) \( x \) \( y \) . \( f(x \cap \neg (g \ y)) \leq \neg \neg f \ x \cap \neg y \land g(y \cap \neg (f \ x)) \leq \))
\[ g \cap -x \]

\textit{proof}

\textbf{lemma} conjugate-char-1-isotone:
\begin{align*}
\text{conjugate } f g & \implies \text{isotone } f \\
& \implies f(x \cap -(g \cap -f(x))) \leq f(x \cap -y) \land g(y \cap -x) \leq g \cap -x
\end{align*}

\textit{proof}

\textbf{lemma} dense-lattice-char-1:
\begin{align*}
(\forall x y . x \cap y = \bot \implies x = \bot \lor y = \bot) & \iff (\forall x . x \neq \bot \implies \text{dense } x)
\end{align*}

\textit{proof}

\textbf{lemma} dense-lattice-char-2:
\begin{align*}
(\forall x y . x \cap y = \bot \implies x = \bot \lor y = \bot) & \iff (\forall x . \text{regular } x \implies x = \bot \lor x = \top)
\end{align*}

\textit{proof}

\textbf{lemma} restrict-below-Rep-eq:
\[ x \cap --y \leq z \implies x \cap y = x \cap z \cap y \]

\textit{proof}

\textend

The following class gives equational axioms for the pseudocomplement operation.

\textbf{class} p-algebra-eq = bounded-lattice + uminus +

\textbf{assumes} p-bot-eq: \(-\bot = \top\)

\textbf{and} p-top-eq: \(-\top = \bot\)

\textbf{and} inf-import-p-eq: \[ x \cap -(x \cap y) = x \cap -y \]

\textbf{begin}

\textbf{lemma} inf-p-eq:
\[ x \cap -x = \bot \]

\textit{proof}

\textbf{subclass} p-algebra

\textit{proof}

\textbf{end}

\subsection{Pseudocomplemented Distributive Lattices}

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

\textbf{class} pd-algebra = p-algebra + bounded-distrib-lattice

\textbf{begin}

18
lemma p-inf-sup-below:
\[ -x \cap (x \cup y) \leq y \]
⟨proof⟩

lemma pp-inf-sup-p [simp]:
\[ -x \cap (x \cup -x) = x \]
⟨proof⟩

lemma complement-p:
\[ x \cap y = \text{bot} \implies x \cup y = \text{top} \implies -x = y \]
⟨proof⟩

lemma complemented-regular:
\[ \text{complemented } x \implies \text{regular } x \]
⟨proof⟩

lemma regular-inf-dense:
\[ \exists y z . \text{regular } y \land \text{dense } z \land x = y \cap z \]
⟨proof⟩

lemma maddux-3-12 [simp]:
\[ (x \cup -y) \cap (x \cup y) = x \]
⟨proof⟩

lemma maddux-3-13 [simp]:
\[ (x \cup y) \cap -x = y \cap -x \]
⟨proof⟩

lemma maddux-3-20:
\[ (v \cap w) \cup (-v \cap x) \cap -((v \cap y) \cup (-v \cap z)) = (v \cap w \cap -y) \cup (-v \cap x \cap -z) \]
⟨proof⟩

lemma order-char-1:
\[ x \leq y \iff x \leq y \cup -x \]
⟨proof⟩

lemma order-char-2:
\[ x \leq y \iff x \cup -x \leq y \cup -x \]
⟨proof⟩

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]:
  \(-(x \sqcap y) = -x \sqcup -y\)
  ⟨proof⟩

lemma pp-dist-sup [simp]:
  \(-\neg(x \sqcup y) = -\neg x \sqcup -\neg y\)
  ⟨proof⟩

lemma regular-closed-sup:
  regular x \Rightarrow regular y \Rightarrow regular (x \sqcup y)
  ⟨proof⟩

  The regular elements are precisely the ones having a complement.

lemma regular-complemented-iff:
  regular x \iff complemented x
  ⟨proof⟩

lemma selection-closed-sup:
  selection s x \Rightarrow selection t x \Rightarrow selection (s \sqcup t) x
  ⟨proof⟩

lemma huntington-3-pp [simp]:
  \(-(-x \sqcup y) \sqcup \neg(-x \sqcup y) = \neg x\)
  ⟨proof⟩

lemma maddux-3-3 [simp]:
  \(-(x \sqcup y) \sqcup -(x \sqcup y) = -x\)
  ⟨proof⟩

lemma maddux-3-11-pp:
  \((x \sqcap -y) \sqcup (x \sqcap -\neg y) = x\)
  ⟨proof⟩

lemma maddux-3-19-pp:
  \((-x \sqcap y) \sqcup (\neg-x \sqcap z) = (\neg-x \sqcup y) \cap (\neg-x \sqcup z)\)
  ⟨proof⟩

lemma compl-inter-eq-pp:
  \(-\neg x \sqcap y = -\neg x \sqcap z \Rightarrow -x \sqcap y = -x \sqcap z \Rightarrow y = z\)
  ⟨proof⟩

lemma maddux-3-21-pp [simp]:
  \(-\neg x \sqcup (\neg x \sqcap y) = -\neg x \sqcup y\)

\text{20}
proof

lemma shunting-2-pp:
\[ x \leq \neg\neg y \iff -x \sqcup \neg\neg y = \top \]
proof

lemma shunting-p:
\[ x \sqcap y \leq \neg z \iff x \leq \neg z \sqcup \neg y \]
proof

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:
\[ x \neg \neg y \leq z \iff x \leq z \sqcup \neg y \]
proof

lemma conjugate-char-2-pp:
\[ \text{conjugate } f g \iff f \bot = \bot \land g \bot = \bot \land (\forall x y . f x \sqcap y \leq \neg(f x \sqcap \neg(g y)) \land g y \sqcap x \leq \neg(g y \sqcap \neg(f x))) \]
proof

lemma conjugate-char-2-pp-additive:
assumes conjugate f g
and additive f
and additive g
shows \[ f x \sqcap y \leq f(x \sqcap \neg(g y)) \land g y \sqcap x \leq g(y \sqcap \neg(f x)) \]
proof

end

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

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

subclass stone-algebra
proof

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

lemma regular-bot-top:
\[ \text{regular } x \iff x = \bot \lor x = \top \]
proof

lemma not-bot-dense:
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.

```plaintext
class implies =
  fixes implies :: 'a ⇒ 'a ⇒ 'a (infixl 65)

class heyting-semilattice = semilattice-inf + implies +
  assumes implies-galois: x ⊓ y ≤ z ⇐⇒ x ≤ y ⇒ z
begin

lemma implies-below-eq [simp]:
  y ⊓ (x ⇒ y) = y
  ⟨proof⟩

lemma implies-increasing:
  x ≤ y ⇒ x
  ⟨proof⟩

lemma implies-galois-swap:
  x ≤ y ⇒ z ⇐⇒ y ≤ x ⇒ z
  ⟨proof⟩

lemma implies-galois-var:
  x ⊓ y ≤ z ⇐⇒ y ≤ x ⇒ z
  ⟨proof⟩

lemma implies-galois-increasing:
  x ≤ y ⇒ (x ⊓ y)
  ⟨proof⟩

lemma implies-galois-decreasing:
  (y ⇒ x) ⊓ y ≤ x
  ⟨proof⟩
```

22
\textbf{lemma} \textit{implies-mp-below}:
\[ x \cap (x \rightarrow y) \leq y \]
\textit{(proof)}

\textbf{lemma} \textit{implies-isotone}:
\[ x \leq y \Rightarrow z \rightarrow x \leq z \rightarrow y \]
\textit{(proof)}

\textbf{lemma} \textit{implies-antitone}:
\[ x \leq y \Rightarrow y \rightarrow z \leq x \rightarrow z \]
\textit{(proof)}

\textbf{lemma} \textit{implies-isotone-inf}:
\[ x \rightarrow (y \cap z) \leq x \rightarrow y \]
\textit{(proof)}

\textbf{lemma} \textit{implies-antitone-inf}:
\[ x \rightarrow z \leq (x \cap y) \rightarrow z \]
\textit{(proof)}

\textbf{lemma} \textit{implies-curry}:
\[ x \rightarrow (y \rightarrow z) = (x \cap y) \rightarrow z \]
\textit{(proof)}

\textbf{lemma} \textit{implies-curry-flip}:
\[ x \rightarrow (y \rightarrow z) = y \rightarrow (x \rightarrow z) \]
\textit{(proof)}

\textbf{lemma} \textit{triple-implies [simp]}:
\[ ((x \rightarrow y) \rightarrow y) \rightarrow y = x \rightarrow y \]
\textit{(proof)}

\textbf{lemma} \textit{implies-mp-eq [simp]}:
\[ x \cap (x \rightarrow y) = x \cap y \]
\textit{(proof)}

\textbf{lemma} \textit{implies-dist-implies}:
\[ x \rightarrow (y \rightarrow z) \leq (x \rightarrow y) \rightarrow (x \rightarrow z) \]
\textit{(proof)}

\textbf{lemma} \textit{implies-import-inf [simp]}:
\[ x \cap ((x \cap y) \rightarrow (x \rightarrow z)) = x \cap (y \rightarrow z) \]
\textit{(proof)}

\textbf{lemma} \textit{implies-dist-inf}:
\[ x \rightarrow (y \cap z) = (x \rightarrow y) \cap (x \rightarrow z) \]
\textit{(proof)}

\textbf{lemma} \textit{implies-itself-top}:
\[ y \leq x \Rightarrow x \]
\begin{proof}
\end{proof}

\textbf{lemma inf-implies-top}:
\[ z \leq (x \cap y) \Rightarrow x \]
\begin{proof}
\end{proof}

\textbf{lemma inf-inf-implies [simp]}:
\[ z \cap ((x \cap y) \Rightarrow x) = z \]
\begin{proof}
\end{proof}

\textbf{lemma le-implies-top}:
\[ x \leq y \Rightarrow z \leq x \Rightarrow y \]
\begin{proof}
\end{proof}

\textbf{lemma le-iff-le-implies}:
\[ x \leq y \iff x \leq x \Rightarrow y \]
\begin{proof}
\end{proof}

\textbf{lemma implies-inf-isotone}:
\[ x \Rightarrow y \leq (x \cap z) \Rightarrow (y \cap z) \]
\begin{proof}
\end{proof}

\textbf{lemma implies-transitive}:
\[ (x \Rightarrow y) \cap (y \Rightarrow z) \leq x \Rightarrow z \]
\begin{proof}
\end{proof}

\textbf{lemma implies-inf-absorb [simp]}:
\[ x \Rightarrow (x \cap y) = x \Rightarrow y \]
\begin{proof}
\end{proof}

\textbf{lemma implies-implies-absorb [simp]}:
\[ x \Rightarrow (x \Rightarrow y) = x \Rightarrow y \]
\begin{proof}
\end{proof}

\textbf{lemma implies-inf-identity}:
\[ (x \Rightarrow y) \cap y = y \]
\begin{proof}
\end{proof}

\textbf{lemma implies-itself-same}:
\[ x \Rightarrow x = y \Rightarrow y \]
\begin{proof}
\end{proof}

\textbf{end}

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

\textbf{class} heyting-semilattice-eq = semilattice-inf + implies +

\textbf{assumes} implies-mp-below: \( x \cap (x \Rightarrow y) \leq y \)
and implies-galois-increasing: \( x \leq y \Rightarrow (x \cap y) \leq \ldots \)
and implies-isotone-inf: \( x \Rightarrow (y \cap z) \leq x \Rightarrow y \)

begin
subclass heyting-semilattice
(proof)
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]:
\( x \Rightarrow x = \top \)
(proof)

lemma implies-order:
\( x \leq y \iff x \Rightarrow y = \top \)
(proof)

lemma inf-implies [simp]:
\( (x \cap y) \Rightarrow x = \top \)
(proof)

lemma top-implies [simp]:
\( \top \Rightarrow x = x \)
(proof)

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:
\( (x \sqcup y) \cap (x \sqcup z) \leq x \sqcup (y \cap z) \)
(proof)

subclass distrib-lattice
(proof)

lemma implies-isotone-sup:
\( x \Rightarrow y \leq x \Rightarrow (y \sqcup z) \)
proof

lemma implies-antitone-sup:
\[(x \sqcup y) \rightsquigarrow z \leq x \rightsquigarrow z\]

lemma implies-sup:
\[x \rightsquigarrow z \leq (y \rightsquigarrow z) \rightsquigarrow ((x \sqcup y) \rightsquigarrow z)\]

lemma implies-dist-sup:
\[(x \sqcup y) \rightsquigarrow z = (x \rightsquigarrow z) \sqcap (y \rightsquigarrow z)\]

lemma implies-antitone-isotone:
\[(x \sqcup y) \rightsquigarrow (x \sqcap y) \leq x \rightsquigarrow y\]

lemma implies-antisymmetry:
\[(x \rightsquigarrow y) \sqcap (y \rightsquigarrow x) = (x \sqcup y) \rightsquigarrow (x \sqcap y)\]

lemma sup-inf-implies [simp]:
\[(x \sqcup y) \sqcap (x \rightsquigarrow y) = y\]

lemma implies-subdist-sup:
\[(x \rightsquigarrow y) \sqcup (x \rightsquigarrow z) \leq x \rightsquigarrow (y \sqcup z)\]

lemma implies-subdist-inf:
\[(x \rightsquigarrow z) \sqcup (y \rightsquigarrow z) \leq (x \sqcap y) \rightsquigarrow z\]

lemma implies-sup-absorb:
\[(x \rightsquigarrow y) \sqcup z \leq (x \sqcup z) \rightsquigarrow (y \sqcup z)\]

lemma sup-below-implies-implies:
\[x \sqcup y \leq (x \rightsquigarrow y) \rightsquigarrow y\]

end

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

subclass bounded-heyting-semilattice (proof)
lemma implies-bot [simp]:
  bot \leadsto x = \top
  ⟨proof⟩
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: \neg x = x \leadsto \bot
begin
subclass pd-algebra
  ⟨proof⟩
lemma boolean-implies-below:
  \neg x \sqcup y \leq x \leadsto y
  ⟨proof⟩
lemma negation-implies:
  \neg(x \leadsto y) = \neg\neg x \sqcap \neg y
  ⟨proof⟩
lemma double-negation-dist-implies:
  \neg\neg(x \leadsto y) = \neg\neg x \leadsto \neg\neg y
  ⟨proof⟩
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 \leadsto y) = x \sqcap y
  and implies-import-inf: x \sqcap (\{x \sqcap y\} \leadsto (x \leadsto z)) = x \sqcap (y \leadsto z)
  and inf-inf-implies: z \sqcap (\{x \sqcap y\} \leadsto x) = z
  and uminus-eq-eq: \neg x = x \leadsto \bot
begin
subclass heyting-algebra
  ⟨proof⟩
end
A relative pseudocomplement is not enough to obtain the Stone equation, so we add it in the following class.

```plaintext
class heyting-stone-algebra = heyting-algebra +
  assumes heyting-stone: \(-x \sqcup \neg\neg x = \top\)
begin
subclass stone-algebra
  ⟨proof⟩
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 = \(\lambda x y . y - x\)
  ⟨proof⟩
lemma curry-minus:
  \(x - (y \sqcup z) = (x - y) - z\)
  ⟨proof⟩
lemma minus-subdist-sup:
  \((x - z) \sqcup (y - z) \leq (x \sqcup y) - z\)
  ⟨proof⟩
lemma inf-sup-minus:
  \((x \cap y) \sqcup (x - y) = x\)
  ⟨proof⟩
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
⟨proof⟩

sublocale heyting: heyting-algebra where implies = λx y. ¬x ⊔ y
⟨proof⟩

subclass brouwer-algebra
⟨proof⟩

lemma huntington-3 [simp]:
(¬(¬x ⊔ ¬y) ⊔ (¬(¬x ⊔ y)) = x
⟨proof⟩

lemma maddux-3-1:
x ⊔ ¬x = y ⊔ ¬y
⟨proof⟩

lemma maddux-3-4:
x ⊔ (y ⊔ ¬y) = z ⊔ ¬z
⟨proof⟩

lemma maddux-3-11 [simp]:
(x ∩ y) ⊔ (x ∩ ¬y) = x
⟨proof⟩

lemma maddux-3-19:
(¬x ∩ y) ⊔ (x ∩ z) = (x ∩ y) ⊔ (¬x ⊔ z)
⟨proof⟩

lemma compl-inter-eq:
x ∩ y = x ∩ z → ¬x ∩ y = ¬x ∩ z → y = z
⟨proof⟩

lemma maddux-3-21 [simp]:
x ⊔ (¬x ∩ y) = x ∩ y
⟨proof⟩

lemma shunting-1:
x ≤ y ↔ x ∩ ¬y = bot
⟨proof⟩

lemma uminus-involutive:
uminus ∘ uminus = id
⟨proof⟩


lemma \text{uminus-injective}:
\text{uminus} \circ f = \text{uminus} \circ g \implies f = g
⟨proof⟩

lemma \text{conjugate-unique}:
\text{conjugate} f g \implies \text{conjugate} f h \implies g = h
⟨proof⟩

lemma \text{dual-additive-additive}:
\text{dual-additive} (\text{uminus} \circ f) \implies \text{additive} f
⟨proof⟩

lemma \text{conjugate-additive}:
\text{conjugate} f g \implies \text{additive} f
⟨proof⟩

lemma \text{conjugate-isotone}:
\text{conjugate} f g \implies \text{isotone} f
⟨proof⟩

lemma \text{conjugate-char-1}:
\text{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⟩

lemma \text{conjugate-char-2}:
\text{conjugate} f g \iff \text{bot} = \text{bot} \land \text{bot} = \text{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))
⟨proof⟩

lemma \text{shunting}:
x \cap y \leq z \iff x \leq z \cup -y
⟨proof⟩

lemma \text{shunting-var}:
x \cap -y \leq z \iff x \leq z \cup y
⟨proof⟩

end

class \text{non-trivial-stone-algebra} = \text{non-trivial-bounded-order} + \text{stone-algebra}

class \text{non-trivial-boolean-algebra} = \text{non-trivial-stone-algebra} + \text{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 . ∃ 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 ≡ ∃ x . X = ↓x

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

abbreviation up-set :: 'a set ⇒ 'a set (⇑- [81] 80)
where ↑X ≡ { y . ∃ x∈X . x ≤ y }
abbreviation is-up-set :: 'a set ⇒ bool
  where is-up-set X ≡ ∀ x∈X . ∀ y . x ≤ y → y∈X

abbreviation is-principal-up :: 'a set ⇒ bool
  where is-principal-up X ≡ ∃ x . X = ↑ x

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

definition filter :: 'a set ⇒ bool
  where filter F ≡ (F ≠ { }) ∧ (∀ x∈F . ∀ y∈F . ∃ z∈F . z ≤ x ∧ z ≤ y) ∧
  is-up-set F

abbreviation proper-filter :: 'a set ⇒ bool
  where proper-filter F ≡ filter F ∧ F ≠ UNIV

abbreviation ultra-filter :: 'a set ⇒ bool
  where ultra-filter F ≡ proper-filter F ∧ (∀ G . proper-filter G ∧ F ⊆ G → F = G)

end

context order
begin

lemma self-in-downset [simp]:
  x ∈ ↓ x
  ⟨proof⟩

lemma self-in-upset [simp]:
  x ∈ ↑ x
  ⟨proof⟩

lemma up-filter [simp]:
  filter (↑ x)
  ⟨proof⟩

lemma up-set-up-set [simp]:
  is-up-set (↑ X)
  ⟨proof⟩

lemma up-injective:
  ↑ x = ↑ y → x = y
  ⟨proof⟩

lemma up-antitone:
  x ≤ y ↔ ↑ y ⊆ ↑ x
  ⟨proof⟩

end
context order-bot
begin

lemma bot-in-downset [simp]:
bot ∈ ↓x
⟨proof⟩

lemma down-bot [simp]:
↓bot = {bot}
⟨proof⟩

lemma up-bot [simp]:
↑bot = UNIV
⟨proof⟩

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 ∃ G. ultra-filter G ∧ F ⊆ G
⟨proof⟩
end

context order-top
begin

lemma down-top [simp]:
↓top = UNIV
⟨proof⟩

lemma top-in-upset [simp]:
top ∈ ↑x
⟨proof⟩

lemma up-top [simp]:
↑top = {top}
⟨proof⟩

lemma filter-top [simp]:
filter {top}
⟨proof⟩

lemma top-in-filter [simp]:
filter F ⊢ top ∈ F
⟨proof⟩

33
The existence of proper filters and ultrafilters requires that the underlying order contains at least two elements.

**Context non-trivial-order**

**Lemma proper-filter-exists:**

\[ \exists F . \text{proper-filter } F \]

(\emph{proof})

**Context non-trivial-order-bot**

**Lemma ultra-filter-exists:**

\[ \exists F . \text{ultra-filter } F \]

(\emph{proof})

**Context non-trivial-bounded-order**

**Lemma proper-filter-top:**

\[ \text{proper-filter } \{ \text{top} \} \]

(\emph{proof})

**Lemma ultra-filter-top:**

\[ \exists G . \text{ultra-filter } G \land \text{top } \in G \]

(\emph{proof})

**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**

**Abbreviation prime-filter : 'a set => bool**

\[ \text{where prime-filter } F \equiv \text{proper-filter } F \land (\forall x y . x \sqcup y \in F \rightarrow x \in F \lor y \in F) \]
end

context semilattice-inf
begin

lemma filter-inf-closed:
filter F \Rightarrow x \in F \Rightarrow y \in F \Rightarrow x \sqcap y \in F
⟨proof⟩

lemma filter-univ:
filter UNIV
⟨proof⟩

The operation \textit{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 \sqcap y \leq z \}

lemma filter-sup:
assumes filter F
and filter G
shows filter (filter-sup F G)
⟨proof⟩

lemma filter-sup-left-upper-bound:
assumes filter G
shows F \subseteq filter-sup F G
⟨proof⟩

lemma filter-sup-symmetric:
filter-sup F G = filter-sup G F
⟨proof⟩

lemma filter-sup-right-upper-bound:
filter F \Rightarrow G \subseteq filter-sup F G
⟨proof⟩

lemma filter-sup-least-upper-bound:
assumes filter H
and F \subseteq H
and G \subseteq H
shows filter-sup F G \subseteq H
⟨proof⟩

lemma filter-sup-left-isotone:
G \subseteq H \Rightarrow filter-sup G F \subseteq filter-sup H F
⟨proof⟩

lemma filter-sup-right-isotone:
G \subseteq H \Rightarrow filter-sup F G \subseteq filter-sup F H
proof

lemma filter-sup-right-isotone-var:
filter-sup F (G ∩ H) ⊆ filter-sup F H

proof

lemma up-dist-inf:
↑(x ∩ y) = filter-sup (↑x) (↑y)

proof

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

lemma filter-inf-filter [simp]:
assumes filter F
shows filter (⇑{ y . ∃ z ∈ F . x ⊓ z = y})

proof

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 ∩ G)

proof

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 }
proof

lemma simp-filter [simp]:
filter (Rep-filter x)
proof

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
proof


lift-definition sup-filter :: 'a filter ⇒ 'a filter ⇒ 'a filter is filter-sup
⟨proof⟩

lift-definition inf-filter :: 'a filter ⇒ 'a filter ⇒ 'a filter is inter
⟨proof⟩

lift-definition less-eq-filter :: 'a filter ⇒ 'a filter ⇒ bool is subset-eq (proof)

lift-definition less-filter :: 'a filter ⇒ 'a filter ⇒ bool is subset (proof)

instance
⟨proof⟩
end

context bounded-semilattice-inf-top
begin

abbreviation filter-complements F G ≡ filter F ∧ filter G ∧ filter-sup F G = UNIV ∧ F ∩ G = {top}

end

The set of filters over a bounded semilattice forms a bounded lattice.

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

lift-definition bot-filter :: 'a filter is {top}
⟨proof⟩

instance
⟨proof⟩
end

context lattice
begin

lemma up-dist-sup:
↑(x ⊔ y) = ↑x ∩ ↑y
⟨proof⟩

end

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 \( \text{up-filter } x \equiv \text{Abs-filter } (\uparrow x) \)

**Lemma** up-filter-dist-inf:
\[ \text{up-filter } ((x::'a::lattice) \cap y) = \text{up-filter } x \sqcup \text{up-filter } y \]

**Lemma** up-filter-dist-sup:
\[ \text{up-filter } ((x::'a::lattice) \sqcup y) = \text{up-filter } x \sqcap \text{up-filter } y \]

**Lemma** up-filter-injective:
\[ \text{up-filter } x = \text{up-filter } y \implies x = y \]

**Lemma** up-filter-antitone:
\[ x \leq y \iff \text{up-filter } y \leq \text{up-filter } x \]

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 \Rightarrow "b::order) \Rightarrow "a filter \Rightarrow "b filter

**Where** filter-map \( f \ F \equiv \text{Abs-filter } (f \cdot \text{Rep-filter } F) \)

**Lemma** filter-map-filter:
assumes mono \( f \)
and \( \forall x \ y \cdot f x \leq y \implies (\exists z \cdot x \leq z \land y = f z) \)
shows filter \( f \cdot \text{Rep-filter } F \)

**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 filter \( F \)
and filter \( G \)
and filter \( H \)
shows filter-sup \( F \) \((G \cap H) = \text{filter-sup } F \ G \cap \text{filter-sup } F \ H \)

**Lemma** filter-inf-principal-rep:
\( F \cap G = \uparrow z \implies (\exists x \in F . \exists y \in G . z = x \cup y) \)

\[ \langle \text{proof} \rangle \]

\textbf{lemma} \ filter-sup-principal-rep:
\textbf{assumes} \ filter F \\
\textbf{and} \ filter G \\
\textbf{and} \ filter-sup F G = \uparrow z \\
\textbf{shows} \ \exists x \in F . \exists y \in G . z = x \cap y 

\[ \langle \text{proof} \rangle \]

\textbf{lemma} \ inf-sup-principal-aux:
\textbf{assumes} \ filter F \\
\textbf{and} \ filter G \\
\textbf{and} \ is-principal-up (filter-sup F G) \\
\textbf{and} \ is-principal-up (F \cap G) \\
\textbf{shows} \ is-principal-up F 

\[ \langle \text{proof} \rangle \]

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

\textbf{lemma} \ inf-sup-principal:
\textbf{assumes} \ filter F \\
\textbf{and} \ filter G \\
\textbf{and} \ is-principal-up (filter-sup F G) \\
\textbf{and} \ is-principal-up (F \cap G) \\
\textbf{shows} \ is-principal-up F \land is-principal-up G 

\[ \langle \text{proof} \rangle \]

\textbf{lemma} \ filter-sup-absorb-inf:
\textbf{assumes} \ filter F = \implies filter G = \implies filter-sup (F \cap G) G = G 

\[ \langle \text{proof} \rangle \]

\textbf{lemma} \ filter-inf-absorb-sup:
\textbf{assumes} \ filter F = \implies filter G = \implies filter-sup F G \cap H = filter-sup (F \cap H) (G \cap H) 

\[ \langle \text{proof} \rangle \]

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

\textbf{lemma} \ ultra-filter-prime:
\textbf{assumes} \ ultra-filter F \\
\textbf{shows} \ prime-filter F 

\[ \langle \text{proof} \rangle \]

end
context distrib-lattice-bot
begin

lemma prime-filter:
proper-filter F \implies \exists G . prime-filter G \land F \subseteq G
⟨proof⟩
end

context distrib-lattice-top
begin

lemma complemented-filter-inf-principal:
assumes filter-complements F G
shows is-principal-up (F \cap \rceil x)
⟨proof⟩
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⟩
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 \( \varphi(x) = \rceil x \cap D(S) \).

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

41
context p-algebra
begin

subclass sup-inf-top-bot-uminus-ord ⟨proof⟩

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 bounded-lattice-homomorphism :: ('a::bounded-lattice ⇒ 'b::bounded-lattice) ⇒ bool
where bounded-lattice-homomorphism f ≡ sup-inf-top-bot-homomorphism f

abbreviation sup-inf-top-bot-uminus-homomorphism :: ('a::sup-inf-top-bot-uminus ⇒ 'b::sup-inf-top-bot-uminus) ⇒ bool
where sup-inf-top-bot-uminus-homomorphism f ≡ sup-inf-top-bot-homomorphism f ∧ (∀ x . f (-x) = -f x)

abbreviation sup-inf-top-bot-uminus-ord-homomorphism :: ('a::sup-inf-top-bot-uminus-ord ⇒ 'b::sup-inf-top-bot-uminus-ord) ⇒ bool
where sup-inf-top-bot-uminus-ord-homomorphism f ≡ sup-inf-top-bot-homomorphism f ∧ (∀ x y . x ≤ y → f x ≤ f y)

abbreviation sup-inf-top-isomorphism :: ('a::sup,inf,top ⇒ 'b::sup,inf,top)) ⇒ bool
where sup-inf-top-isomorphism f ≡ sup-inf-top-homomorphism f ∧ bij f

abbreviation bounded-lattice-top-isomorphism :: ('a::bounded-lattice-top ⇒ 'b::bounded-lattice-top) ⇒ bool
where bounded-lattice-top-isomorphism f ≡ sup-inf-top-isomorphism f

abbreviation sup-inf-top-bot-uminus-isomorphism :: ('a::sup-inf-top-bot-uminus
⇒ 'b::sup-inf-top-bot-uminus) ⇒ bool
  where sup-inf-top-bot-uminus-isomorphism f ≡ sup-inf-top-bot-uminus-homomorphism f ∧ bij f

abbreviation stone-algebra-isomorphism :: ('a::stone-algebra ⇒ 'b::stone-algebra) ⇒ bool
  where stone-algebra-isomorphism f ≡ sup-inf-top-bot-uminus-isomorphism f

abbreviation boolean-algebra-isomorphism :: ('a::boolean-algebra ⇒ 'b::boolean-algebra) ⇒ bool
  where boolean-algebra-isomorphism f ≡ sup-inf-top-bot-uminus-isomorphism f

lemma sup-homomorphism-mono
  sup-homomorphism (f::'a::semilattice-sup ⇒ 'b::semilattice-sup) ⇒ mono f
⟨proof⟩

lemma sup-isomorphism-ord-isomorphism
  assumes sup-homomorphism (f::'a::semilattice-sup ⇒ 'b::semilattice-sup) and bij f
  shows x ≤ y ⟷ f x ≤ f y
⟨proof⟩

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
⟨proof⟩

lemma simp-regular [simp]:
  ∃ y . Rep-regular x = − y
⟨proof⟩

setup-lifting type-definition-regular
instantiation regular :: (stone-algebra) boolean-algebra
begin

lift-definition sup-regular :: 'a regular ⇒ 'a regular ⇒ 'a regular is sup
⟨proof⟩

lift-definition inf-regular :: 'a regular ⇒ 'a regular ⇒ 'a regular is inf
⟨proof⟩

lift-definition minus-regular :: 'a regular ⇒ 'a regular ⇒ 'a regular is λx y . x ∩ −y
⟨proof⟩

lift-definition uminus-regular :: 'a regular ⇒ 'a regular is uminus
⟨proof⟩

lift-definition bot-regular :: 'a regular is bot
⟨proof⟩

lift-definition top-regular :: 'a regular is top
⟨proof⟩

lift-definition less-eq-regular :: 'a regular ⇒ 'a regular ⇒ bool is less-eq ⟨proof⟩

lift-definition less-regular :: 'a regular ⇒ 'a regular ⇒ bool is less ⟨proof⟩

instance ⟨proof⟩
end

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

instance ⟨proof⟩
end

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
⟨proof⟩

lemma simp-dense [simp]:
− Rep-dense x = bot
⟨proof⟩
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
⟨proof⟩

lift-definition inf-dense :: 'a dense ⇒ 'a dense ⇒ 'a dense is inf
⟨proof⟩

lift-definition top-dense :: 'a dense is top
⟨proof⟩

lift-definition less-eq-dense :: 'a dense ⇒ 'a dense ⇒ bool is less-eq ⟨proof⟩

lift-definition less-dense :: 'a dense ⇒ 'a dense ⇒ bool is less ⟨proof⟩

instance ⟨proof⟩

end

lemma up-filter-dense-antitone-dense:
dense (x ⊔ -x ⊔ y) ∧ dense (x ⊔ -x ⊔ y ⊔ z)
⟨proof⟩

lemma up-filter-dense-antitone:
up-filter (Abs-dense (x ⊔ -x ⊔ y ⊔ z)) ≤ up-filter (Abs-dense (x ⊔ -x ⊔ y))
⟨proof⟩

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 }
⟨proof⟩

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 ⇒ 'a dense-filter-type
⇒ 'a dense-filter-type is sup ⟨proof⟩

lift-definition inf-dense-filter-type :: 'a dense-filter-type ⇒ 'a dense-filter-type ⇒
'a dense-filter-type is inf ⟨proof⟩
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**

\[ \text{stone-phi-set} :: 'a::stone-algebra regular \Rightarrow 'a dense set \]
where \( \text{stone-phi-set} x \equiv \{ y . \neg \text{Rep-regular} x \leq \text{Rep-dense} y \} \)

**Lemma**

\[ \text{stone-phi-set-filter} : \text{filter (stone-phi-set x)} \]

**Definition**

\[ \text{stone-phi} :: 'a::stone-algebra regular \Rightarrow 'a dense-filter \]
where \( \text{stone-phi} x = \text{Abs-filter (stone-phi-set x)} \)

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

**Interpretation**

\[ \text{stone-phi}: \text{triple stone-phi} \]

**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.
lemma phi-top:
phi top = Abs-filter UNIV
⟨proof⟩

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

definition pairs :: ('a × 'b filter) set
where pairs = { (x,y) . ∃ z . y = phi (-x) ★ 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 × 'b filter) ⇒ ('a × 'b filter) ⇒ bool
where pairs-less-eq (x,y) (z,w) = (x ≤ z ∧ w ≤ y)

fun pairs-less :: ('a × 'b filter) ⇒ ('a × 'b filter) ⇒ bool
where pairs-less (x,y) (z,w) = (pairs-less-eq (x,y) (z,w) ∧ ¬ pairs-less-eq (z,w) (x,y))

fun pairs-sup :: ('a × 'b filter) ⇒ ('a × 'b filter) ⇒ ('a × 'b filter)
where pairs-sup (x,y) (z,w) = (x ⊔ z, y ⊓ w)

fun pairs-inf :: ('a × 'b filter) ⇒ ('a × 'b filter) ⇒ ('a × 'b filter)
where pairs-inf (x,y) (z,w) = (x ⊓ z, y ⊔ w)

fun pairs-uminus :: ('a × 'b filter) ⇒ ('a × 'b filter)
where pairs-uminus (x,y) = (-x,phi x)

abbreviation pairs-bot :: ('a × 'b filter)
where pairs-bot ≡ (bot,Abs-filter UNIV)

abbreviation pairs-top :: ('a × 'b filter)
where pairs-top ≡ (top,Abs-filter {top})

lemma pairs-top-in-set:
(x,y) ∈ pairs ⇒ top ∈ Rep-filter y
⟨proof⟩

lemma phi-complemented:
complement (phi x) (phi (-x))
⟨proof⟩

lemma phi-inf-principal:
∃ z . up-filter z = phi x ⊓ up-filter y
⟨proof⟩

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 :: 'a ⇒ 'b ⇒ 'b
where rho x y = (SOME z . up-filter z = phi x ∩ up-filter y)

lemma rho-char:
  up-filter (rho x y) = phi x ∩ up-filter y
⟨proof⟩
  The following results show that the pairs are closed under the given operations.

lemma pairs-sup-closed:
  assumes (x,y) ∈ pairs
  and (z,w) ∈ pairs
  shows pairs-sup (x,y) (z,w) ∈ pairs
⟨proof⟩

lemma pairs-inf-closed:
  assumes (x,y) ∈ pairs
  and (z,w) ∈ pairs
  shows pairs-inf (x,y) (z,w) ∈ pairs
⟨proof⟩

lemma pairs-uminus-closed:
  pairs-uminus (x,y) ∈ pairs
⟨proof⟩

lemma pairs-bot-closed:
  pairs-bot ∈ pairs
⟨proof⟩

lemma pairs-top-closed:
  pairs-top ∈ pairs
⟨proof⟩

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) ∈ pairs ⇒ (z,w) ∈ pairs ⇒ (u,v) ∈ pairs ⇒ pairs-sup (x,y) (pairs-inf (z,w) (u,v)) = pairs-inf (pairs-sup (x,y) (z,w)) (pairs-sup (x,y) (u,v))
⟨proof⟩

lemma pairs-phi-less-eq:
  (x,y) ∈ pairs ⇒ phi (−x) ≤ y
⟨proof⟩

lemma pairs-uminus-galois:
  assumes (x,y) ∈ pairs
  and (z,w) ∈ pairs
  shows pairs-inf (x,y) (z,w) = pairs-bot ⇐⇒ pairs-less-eq (x,y) (pairs-uminus (z,w))

48
The following results show how the regular elements and the dense elements among the pairs look like.

**Lemma pairs-stone:**

\[(x, y) \in \text{pairs} \implies \text{pairs-sup} \left( \text{pairs-uminus} (x, y) \right) = \text{pairs-top} \]

**Proof**

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

**Function rho-pair:**

\[ \text{rho-pair} :: 'a \times 'b \text{ filter} \Rightarrow 'b \]

**Where**

\[ \text{rho-pair} (x, y) = (\text{SOME} z . \text{up-filter} z = \text{phi} x \sqcap y) \]

**Lemma get-rho-pair-char:**

\[ \text{assumes} (x, y) \in \text{pairs} \]
\[ \text{shows} \ \text{up-filter} (\text{rho-pair} (x, y)) = \text{phi} x \sqcap y \]

**Proof**

**Lemma sa-iso-pair:**

\[ (\neg \neg x, \text{phi} (\neg x) \sqcup \text{up-filter} y) \in \text{pairs} \]

**Proof**

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

(\text{proof})

lemma simp-\(\phi\) [simp]:
\text{bounded-lattice-homomorphism} (\text{Rep-}\(\phi\) \(x\))
(\text{proof})

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 \(\text{triple}\) and instantiated to Stone algebras there, there would be no need for the lifting and we could work with triples directly.

typedef (overloaded) \((a, b)\) \(\text{lifted-pair} = \{ \)
\(p f : (a : \text{non-trivial-boolean-algebra}, b : \text{distrib-lattice-top}) \phi \Rightarrow a \times b \text{ filter} . \forall f . p f f \in \text{triple.pairs} (\text{Rep-}\(\phi\) f) \} \)

(\text{proof})

lemma simp-\(\text{lifted-pair}\) [simp]:
\(\forall f . \text{Rep-}\(\text{lifted-pair}\) p f f \in \text{triple.pairs} (\text{Rep-}\(\phi\) f) \)
(\text{proof})

setup-lifting type-definition-\(\text{lifted-pair}\)

The lifted pairs form a Stone algebra.

instantiation \(\text{lifted-pair} :: (\text{non-trivial-boolean-algebra, distrib-lattice-top})\)
\(\text{stone-algebra}\)

begin

All operations are lifted point-wise.

lift-definition sup-\(\text{lifted-pair}\) :: \((a, b)\) \(\text{lifted-pair} \Rightarrow (a, b)\) \(\text{lifted-pair} \Rightarrow (a, b)\)
\(\text{lifted-pair} \text{ is } \lambda x f y f . \text{triple.pairs-sup} (x f) (y f) \)
(\text{proof})

lift-definition inf-\(\text{lifted-pair}\) :: \((a, b)\) \(\text{lifted-pair} \Rightarrow (a, b)\) \(\text{lifted-pair} \Rightarrow (a, b)\)
\(\text{lifted-pair} \text{ is } \lambda x f y f . \text{triple.pairs-inf} (x f) (y f) \)
(\text{proof})

lift-definition uminus-\(\text{lifted-pair}\) :: \((a, b)\) \(\text{lifted-pair} \Rightarrow (a, b)\) \(\text{lifted-pair} \Rightarrow (a, b)\)
\(\text{lifted-pair} \text{ is } \lambda x f . \text{triple.pairs-uminus} (\text{Rep-}\(\phi\) f) (x f) \)
(\text{proof})
lift-definition bot-lifted-pair :: ('a,'b) lifted-pair is λf . triple.pairs-bot
⟨proof⟩

lift-definition top-lifted-pair :: ('a,'b) lifted-pair is λf . triple.pairs-top
⟨proof⟩

lift-definition less-eq-lifted-pair :: ('a,'b) lifted-pair ⇒ ('a,'b) lifted-pair ⇒ bool is λxf yf . ∀ f . triple.pairs-less-eq (xf f) (yf f) ⟨proof⟩

lift-definition less-lifted-pair :: ('a,'b) lifted-pair ⇒ ('a,'b) lifted-pair ⇒ bool is λxf yf . (∀ f . triple.pairs-less-eq (xf f) (yf f)) ∧ ¬ (∀ f . triple.pairs-less-eq (yf f) (xf f)) ⟨proof⟩

instance ⟨proof⟩

end

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) ⟨proof⟩

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 ⟨proof⟩

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

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

lift-definition bot-stone-phi-pair :: 'a stone-phi-pair is triple.pairs-bot
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 \(~-~x\), which is essentially a projection to the regular element component of \( x \), whence the image has the structure of a Boolean algebra.

\[
\text{fun stone-phi-embed} :: \text{'a::non-trivial-stone-algebra stone-phi-pair} \Rightarrow (\text{'a regular,'a dense}) \text{lifted-pair}
\]

\[
\text{where} \quad \text{stone-phi-embed} x = \text{Abs-lifted-pair} (\lambda f . \text{if Rep-phi f = stone-phi then Rep-stone-phi-pair x else triple.pairs-uminus (Rep-phi f) (triple.pairs-uminus (Rep-phi f) (Rep-stone-phi-pair x)))}
\]

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

\[
\text{lemma stone-phi-embed-triple-pair:}
\]

\[
(\text{if Rep-phi f = stone-phi then Rep-stone-phi-pair x else triple.pairs-uminus (Rep-phi f) (triple.pairs-uminus (Rep-phi f) (Rep-stone-phi-pair x)))} \in \text{triple.pairs (Rep-phi f)}
\]

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.

\[
\text{lemma stone-phi-embed-homomorphism:}
\]

\[
\text{sup-inf-top-bot-uminus-ord-homomorphism stone-phi-embed}
\]

The following lemmas show that the embedding is injective and reflects the order. The latter allows us to easily inherit properties involving in-
equalities from the target of the embedding, without transforming them to equations.

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

\( \text{inj } \text{stone-phi-embed} \)

\( \langle \text{proof} \rangle \)

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

**assumes** stone-phi-embed \( x \leq y \)

**shows** \( x \leq y \)

\( \langle \text{proof} \rangle \)

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

\( \langle \text{proof} \rangle \)

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 \( \Rightarrow \) 'a

**where** sa-iso-inv \( \equiv \lambda p . \text{Rep-regular } (\text{fst } (\text{Rep-stone-phi-pair } p)) \cap \text{Rep-dense } (\text{triple.rho-pair } \text{stone-phi} (\text{Rep-stone-phi-pair } p)) \)

**abbreviation** sa-iso :: 'a::non-trivial-stone-algebra \( \Rightarrow \) 'a stone-phi-pair

**where** sa-iso \( \equiv \lambda x . \text{Abs-stone-phi-pair } (\text{Abs-regular } (\neg x), \text{stone-phi } (\text{Abs-regular } (\neg x)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup \neg x))) \)

**lemma** sa-isotriple-pair:

(\( \text{Abs-regular } (\neg x), \text{stone-phi } (\text{Abs-regular } (\neg x)) \sqcup \text{up-filter } (\text{Abs-dense } (x \sqcup \neg x))) \) \( \in \) triple.pairs stone-phi

\( \langle \text{proof} \rangle \)

**lemma** stone-phi-inf-dense:

\( \text{stone-phi } (\text{Abs-regular } (\neg x)) \sqcap \text{up-filter } (\text{Abs-dense } (y \sqcup \neg y)) \leq \text{up-filter } (\text{Abs-dense } (y \sqcup \neg y \sqcup x)) \)

\( \langle \text{proof} \rangle \)

**lemma** stone-phi-complement:
complement \( (\text{stone-phi} (\text{Abs-regular} (-x))) (\text{stone-phi} (\text{Abs-regular} (-x))) \)

\[ \langle \text{proof} \rangle \]

\textbf{lemma} \ up-dense-stone-phi:
\[ \text{up-filter} (\text{Abs-dense} (x \sqcup -x)) \leq \text{stone-phi} (\text{Abs-regular} (-x)) \]

\[ \langle \text{proof} \rangle \]

The following two results prove that the isomorphisms are mutually inverse.

\textbf{lemma} \ sa-iso-left-invertible:
\[ \text{sa-iso-inv} (\text{sa-iso} x) = x \]

\[ \langle \text{proof} \rangle \]

\textbf{lemma} \ sa-iso-right-invertible:
\[ \text{sa-iso} (\text{sa-iso-inv} p) = p \]

\[ \langle \text{proof} \rangle \]

It remains to show the homomorphism properties, which is done in the following result.

\textbf{lemma} \ sa-iso:
\[ \text{stone-algebra-isomorphism} sa-iso \]

\[ \langle \text{proof} \rangle \]

### 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.

\textbf{typedef} \ (\text{overloaded}) \ (a,b) \ \text{lifted-boolean-algebra} = \{
\begin{align*}
\text{xf} &: (\text{a}::\text{non-trivial-boolean-algebra}, \text{b}::\text{distrib-lattice-top}) \ \text{phi} \Rightarrow \ 'a \ . \ True \\
\text{proof}
\end{align*}
\}

\[ \langle \text{proof} \rangle \]

\textbf{setup-lifting} \ type-definition-lifted-boolean-algebra

\textbf{instantiation} \ \text{lifted-boolean-algebra} ::
\[ (\text{non-trivial-boolean-algebra}, \text{distrib-lattice-top}) \ \text{boolean-algebra} \]

\[ \text{begin} \]

54
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) ⟨proof⟩

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) ⟨proof⟩

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) ⟨proof⟩

lift-definition uminus-lifted-boolean-algebra :: (′a,′b) lifted-boolean-algebra is λxf f . uminus (xf f) ⟨proof⟩

lift-definition bot-lifted-boolean-algebra :: (′a,′b) lifted-boolean-algebra is λf . bot ⟨proof⟩

lift-definition top-lifted-boolean-algebra :: (′a,′b) lifted-boolean-algebra is λf . top ⟨proof⟩

lift-definition less-eq-lifted-boolean-algebra :: (′a,′b) lifted-boolean-algebra ⇒ (′a,′b) lifted-boolean-algebra ⇒ bool is λxf yf f . ∀f . less-eq (xf f) (yf f) ⟨proof⟩

lift-definition less-lifted-boolean-algebra :: (′a,′b) lifted-boolean-algebra ⇒ (′a,′b) lifted-boolean-algebra ⇒ bool is λxf yf f . (∀f . less-eq (xf f) (yf f)) ∧ ¬(∀f . less-eq (yf f) (xf f)) ⟨proof⟩

instance ⟨proof⟩

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
where ba-iso ≡ λpf . Abs-lifted-boolean-algebra (λf . fst (Rep-lifted-pair (Rep-regular pf f)))

lemma ba-iso-inv-lifted-pair:
 (Rep-lifted-boolean-algebra xf f,Rep-phi f (−Rep-lifted-boolean-algebra xf f)) ∈ triple.pairs (Rep-phi f) ⟨proof⟩
lemma ba-iso-inv-regular:
  regular (Abs-lifted-pair (λ f . (Rep-lifted-boolean-algebra xf f . Rep-phi f
  (¬ Rep-lifted-boolean-algebra xf f))))
⟨proof⟩

The following two results prove that the isomorphisms are mutually inverse.

lemma ba-iso-left-invertible:
  ba-iso-inv (ba-iso pf) = pf
⟨proof⟩

lemma ba-iso-right-invertible:
  ba-iso (ba-iso-inv xf) = xf
⟨proof⟩

The isomorphism is established by proving the remaining Boolean algebra homomorphism properties.

lemma ba-iso:
  boolean-algebra-isomorphism ba-iso
⟨proof⟩

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 }
⟨proof⟩

setup-lifting type-definition-lifted-distrib-lattice-top

instantiation 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) ⟨proof⟩

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) ⟨proof⟩

lift-definition top-lifted-distrib-lattice-top :: (‘a, ‘b) lifted-distrib-lattice-top is λf . top ⟨proof⟩

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) ⟨proof⟩

56
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)) ⟨proof⟩

instance ⟨proof⟩ end

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⟩

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) ⟨proof⟩

lemma dl-iso-inv-dense:
  dense (Abs-lifted-pair (λf . (top,up-filter (Rep-lifted-distrib-lattice-top xf f)))) ⟨proof⟩

The following two results prove that the isomorphisms are mutually inverse.

lemma dl-iso-left-invertible:
  dl-iso-inv (dl-iso pf) = pf ⟨proof⟩

lemma dl-iso-right-invertible:
  dl-iso (dl-iso-inv xf) = xf ⟨proof⟩
To obtain the isomorphism, it remains to show the homomorphism properties of lattices with a greatest element.

**lemma** dl-iso:

\[ \text{bounded-lattice-top-isomorphism dl-iso} \]

\[ \langle \text{proof} \rangle \]

### 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:

\[ \text{filter } ( (\lambda qf :: (a::\text{non-trivial-boolean-algebra}, b::\text{distrib-lattice-top}) \text{lifted-pair dense}) . \text{Rep.lifted-distrib-lattice-top } (\text{dl-iso qf}) f) \cdot \text{Rep.filter } (\text{stone-phi } pf) ) \]

\[ \langle \text{proof} \rangle \]

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:

\[ \text{Rep.phi } f (\text{Rep.lifted-boolean-algebra } (\text{ba-iso } pf) f) = \text{filter-map } (\lambda qf :: (a::\text{non-trivial-boolean-algebra}, b::\text{distrib-lattice-top}) \text{lifted-pair dense}) . \text{Rep.lifted-distrib-lattice-top } (\text{dl-iso qf}) f) (\text{stone-phi } pf) \]

\[ \langle \text{proof} \rangle \]

**end**

### References


