Stone Algebras

Walter Guttmann

March 17, 2025

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

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

2.1 General Facts and Notations

The following results extend basic Isabelle/HOL facts.

lemma *imp-as-conj*:
  assumes $P\ x \Longrightarrow Q\ x$
  shows $P\ x \wedge Q\ x \longleftrightarrow P\ x$
  ⟨*proof*⟩

lemma *if-distrib-2*:
  $f\ (if\ c\ then\ x\ else\ y)\ (if\ c\ then\ z\ else\ w) = (if\ c\ then\ f\ x\ z\ else\ f\ y\ w)$
  ⟨*proof*⟩

lemma *left-invertible-inj*:
  $(\forall x\ .\ g\ (f\ x) = x) \Longrightarrow inj\ f$
  ⟨*proof*⟩

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

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

lemma *finite-set-of-finite-funs-pred*:
  assumes *finite* $\{\ x::'a\ .\ True\ \}$
    and *finite* $\{\ y::'b\ .\ P\ y\ \}$

shows *finite* { *f* . (∀ *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 $--x$ instead of $-(-x)$.

context *sup*
begin

notation *sup* (**infixl** ‹⊔› *65*)

definition *additive* :: (′*a* ⇒ ′*a*) ⇒ *bool*
where *additive f* ≡ ∀ *x y* . *f* (*x* ⊔ *y*) = *f x* ⊔ *f y*

end

context *inf*
begin

notation *inf* (**infixl** ‹⊓› *67*)

end

context *uminus*
begin

unbundle *no uminus-syntax*

notation *uminus* (‹(‹*open-block notation*=‹*prefix* −››− -)› [*80*] *80*)

end

2.2 Orders

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* ⇒ ′*a*) ⇒ *bool*
where *isotone f* ≡ ∀ *x y* . *x* ≤ *y* ⟶ *f x* ≤ *f y*

definition *galois* :: (′*a* ⇒ ′*a*) ⇒ (′*a* ⇒ ′*a*) ⇒ *bool*
where *galois l u* ≡ ∀ *x y* . *l x* ≤ *y* ⟷ *x* ≤ *u y*

definition *lifted-less-eq* :: (′*a* ⇒ ′*a*) ⇒ (′*a* ⇒ ′*a*) ⇒ *bool* (‹(- ≤≤ -)› [*51*, *51*] *50*)
where *f* ≤≤ *g* ≡ ∀ *x* . *f x* ≤ *g x*

end

context *order*
begin

lemma *order-lesseq-imp*:
$(\forall z . x \leq z \longrightarrow y \leq z) \longleftrightarrow y \leq x$
⟨*proof*⟩

lemma *galois-char*:
$galois\ l\ u \longleftrightarrow (\forall x . x \leq u\ (l\ x)) \wedge (\forall x . l\ (u\ x) \leq x) \wedge isotone\ l \wedge isotone\ u$
⟨*proof*⟩

lemma *galois-closure*:
$galois\ l\ u \Longrightarrow l\ x = l\ (u\ (l\ x)) \wedge u\ x = u\ (l\ (u\ x))$
⟨*proof*⟩

lemma *lifted-reflexive*:
$f = g \Longrightarrow f \leq\leq g$
⟨*proof*⟩

lemma *lifted-transitive*:
$f \leq\leq g \Longrightarrow g \leq\leq h \Longrightarrow f \leq\leq h$
⟨*proof*⟩

lemma *lifted-antisymmetric*:
$f \leq\leq g \Longrightarrow g \leq\leq f \Longrightarrow f = g$
⟨*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*:
assumes *finite* s
and $s \neq \{\}$
and $\forall x \in s . \forall y \in s . f\ x \leq f\ y \vee f\ y \leq f\ x$
shows $\exists m \in s . \forall z \in s . f\ m \leq f\ z$
⟨*proof*⟩

end

2.3 Semilattices

The following are basic facts in semilattices.

context *semilattice-sup*
begin

lemma *sup-left-isotone*:
$x \leq y \Longrightarrow x \sqcup z \leq y \sqcup z$

⟨*proof*⟩

lemma *sup-right-isotone*:
$x \le y \Longrightarrow z \sqcup x \le z \sqcup y$
⟨*proof*⟩

lemma *sup-left-divisibility*:
$x \le y \longleftrightarrow (\exists z . x \sqcup z = y)$
⟨*proof*⟩

lemma *sup-right-divisibility*:
$x \le y \longleftrightarrow (\exists z . z \sqcup x = y)$
⟨*proof*⟩

lemma *sup-same-context*:
$x \le y \sqcup z \Longrightarrow y \le x \sqcup z \Longrightarrow x \sqcup z = y \sqcup z$
⟨*proof*⟩

lemma *sup-relative-same-increasing*:
$x \le y \Longrightarrow x \sqcup z = x \sqcup w \Longrightarrow y \sqcup z = y \sqcup w$
⟨*proof*⟩

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*
⟨*proof*⟩

end

context *semilattice-inf*
begin

lemma *inf-same-context*:
$x \le y \sqcap z \Longrightarrow y \le x \sqcap z \Longrightarrow x \sqcap z = y \sqcap z$
⟨*proof*⟩

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*: $\exists z . x \le z \wedge y \le z$

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

context *bounded-semilattice-inf-top*
begin

subclass *directed-semilattice-inf*
⟨*proof*⟩

sublocale *inf*: *bounded-semilattice-sup-bot* **where** *sup* = *inf* **and** *less-eq* = *greater-eq* **and** *less* = *greater* **and** *bot* = *top*
⟨*proof*⟩

end

2.4 Lattices

context *lattice*
begin

subclass *directed-semilattice-inf*
⟨*proof*⟩

definition *dual-additive* :: $('a \Rightarrow 'a) \Rightarrow bool$
where *dual-additive* $f \equiv \forall x\ y\ .\ f\ (x \sqcup y) = f\ x \sqcap f\ y$

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* $x\ y \equiv x \sqcup y = top \wedge x \sqcap y = bot$

lemma *complement-symmetric*:
complement $x\ y \Longrightarrow$ *complement* $y\ x$
⟨*proof*⟩

definition *conjugate* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$
where *conjugate* $f\ g \equiv \forall x\ y\ .\ f\ x \sqcap y = bot \longleftrightarrow x \sqcap g\ y = bot$

end

class *dense-lattice* = *bounded-lattice* +
assumes *bot-meet-irreducible*: $x \sqcap y = bot \longrightarrow x = bot \vee y = bot$

context *distrib-lattice*

begin

lemma *relative-equality*:
$x \sqcup z = y \sqcup z \Longrightarrow x \sqcap z = y \sqcap z \Longrightarrow x = y$
⟨*proof*⟩

end

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* ⟨*proof*⟩

subclass *distrib-lattice-top* ⟨*proof*⟩

lemma *complement-shunting*:
assumes *complement z w*
shows $z \sqcap x \leq y \longleftrightarrow x \leq w \sqcup y$
⟨*proof*⟩

end

2.5 Linear Orders

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*:
$max\ x\ y = x \sqcup y$
⟨*proof*⟩

lemma *min-inf*:
$min\ x\ y = x \sqcap y$
⟨*proof*⟩

lemma *sup-inf-selective*:

$(x \sqcup y = x \wedge x \sqcap y = y) \vee (x \sqcup y = y \wedge x \sqcap y = x)$
⟨*proof*⟩

lemma *sup-selective*:
$x \sqcup y = x \vee x \sqcup y = y$
⟨*proof*⟩

lemma *inf-selective*:
$x \sqcap y = x \vee x \sqcap y = y$
⟨*proof*⟩

subclass *distrib-lattice*
⟨*proof*⟩

lemma *sup-less-eq*:
$x \leq y \sqcup z \longleftrightarrow x \leq y \vee x \leq z$
⟨*proof*⟩

lemma *inf-less-eq*:
$x \sqcap y \leq z \longleftrightarrow x \leq z \vee y \leq z$
⟨*proof*⟩

lemma *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*:
$x \neq top \Longrightarrow y \neq top \Longrightarrow x \sqcup y \neq top$
⟨*proof*⟩

lemma *inf-dense*:
$x \neq bot \Longrightarrow y \neq bot \Longrightarrow x \sqcap y \neq bot$
⟨*proof*⟩

lemma *sup-not-bot*:
$x \neq bot \Longrightarrow x \sqcup y \neq bot$

⟨*proof*⟩

lemma *inf-not-top*:
$x \neq top \Longrightarrow x \sqcap y \neq top$
⟨*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

2.6 Non-trivial Algebras

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.

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

2.7 Homomorphisms

This section gives definitions of lattice homomorphisms and isomorphisms and basic properties.

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 *boolean-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 *bot-homomorphism* :: ('*a*::*bot* ⇒ '*b*::*bot*) ⇒ *bool*
 where *bot-homomorphism f* ≡ *f bot* = *bot*

abbreviation *top-homomorphism* :: ('*a*::*top* ⇒ '*b*::*top*) ⇒ *bool*
 where *top-homomorphism f* ≡ *f top* = *top*

abbreviation *minus-homomorphism* :: ('*a*::*minus* ⇒ '*b*::*minus*) ⇒ *bool*
 where *minus-homomorphism f* ≡ ∀ *x y* . *f* (*x* − *y*) = *f x* − *f y*

abbreviation *uminus-homomorphism* :: ('*a*::*uminus* ⇒ '*b*::*uminus*) ⇒ *bool*
 where *uminus-homomorphism f* ≡ ∀ *x* . *f* (−*x*) = −*f x*

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* ∧ *top-homomorphism f*

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

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* ∧ *uminus-homomorphism f*

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-uminus-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 *boolean-algebra-isomorphism* :: ('*a*::*boolean-algebra* ⇒ '*b*::*boolean-algebra*) ⇒ *bool*
where *boolean-algebra-isomorphism f* ≡ *sup-inf-top-bot-uminus-isomorphism f* ∧ *minus-homomorphism 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*⟩

lemma *minus-homomorphism-default*:
assumes ∀ *x y*::'*a*::{*inf*,*minus*,*uminus*} . *x* − *y* = *x* ⊓ −*y*
and ∀ *x y*::'*b*::{*inf*,*minus*,*uminus*} . *x* − *y* = *x* ⊓ −*y*
and *inf-homomorphism* (*f*::'*a* ⇒ '*b*)
and *uminus-homomorphism f*
shows *minus-homomorphism f*
⟨*proof*⟩

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.

class *p-algebra* = *bounded-lattice* + *uminus* +
  assumes *pseudo-complement*: $x \sqcap y = bot \longleftrightarrow x \leq -y$
begin

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

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* $x \equiv \exists y . x \sqcap y = bot \wedge x \sqcup y = top$
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*]:
$-bot = top$
⟨*proof*⟩

lemma *p-top* [*simp*]:
$-top = bot$
⟨*proof*⟩

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

lemma *inf-p* [*simp*]:
$x \sqcap -x = bot$
⟨*proof*⟩

lemma *p-inf* [*simp*]:
$-x \sqcap x = bot$
⟨*proof*⟩

lemma *pp-inf-p*:
$--x \sqcap -x = bot$
⟨*proof*⟩

The double complement is a closure operation.

lemma *pp-increasing*:
$x \leq --x$
⟨*proof*⟩

lemma *ppp* [*simp*]:
$---x = -x$
⟨*proof*⟩

lemma *pp-idempotent*:
$----x = --x$
⟨*proof*⟩

lemma *regular-in-p-image-iff*:
regular $x \longleftrightarrow$ *in-p-image* x
⟨*proof*⟩

lemma *pseudo-complement-pp*:
$x \sqcap y = bot \longleftrightarrow --x \leq -y$
⟨*proof*⟩

lemma *p-antitone*:
$x \leq y \Longrightarrow -y \leq -x$
⟨*proof*⟩

lemma *p-antitone-sup*:
$-(x \sqcup y) \leq -x$
⟨*proof*⟩

lemma *p-antitone-inf*:
$-x \leq -(x \sqcap y)$
⟨*proof*⟩

lemma *p-antitone-iff*:
$x \leq -y \longleftrightarrow y \leq -x$
⟨*proof*⟩

lemma *pp-isotone*:
$x \leq y \Longrightarrow --x \leq --y$
⟨*proof*⟩

lemma *pp-isotone-sup*:
$--x \leq --(x \sqcup y)$
⟨*proof*⟩

lemma *pp-isotone-inf*:
$--(x \sqcap y) \leq --x$
⟨*proof*⟩

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

lemma *p-dist-sup* [*simp*]:
$-(x \sqcup y) = -x \sqcap -y$
⟨*proof*⟩

lemma *p-supdist-inf*:
$-x \sqcup -y \leq -(x \sqcap y)$
⟨*proof*⟩

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

lemma *p-sup-p* [*simp*]:
$-(x \sqcup -x) = bot$
⟨*proof*⟩

lemma *pp-sup-p* [*simp*]:
$--(x \sqcup -x) = top$
⟨*proof*⟩

lemma *dense-pp*:
$dense\ x \longleftrightarrow --x = top$
⟨*proof*⟩

lemma *dense-sup-p*:
  $dense\ (x \sqcup -x)$
  ⟨*proof*⟩

lemma *regular-char*:
  $regular\ x \longleftrightarrow (\exists\, y\ .\ x = -y)$
  ⟨*proof*⟩

lemma *pp-inf-bot-iff*:
  $x \sqcap y = bot \longleftrightarrow --x \sqcap y = bot$
  ⟨*proof*⟩

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

lemma *p-shunting-swap*:
  $x \sqcap y \leq -z \longleftrightarrow x \sqcap z \leq -y$
  ⟨*proof*⟩

lemma *pp-inf-below-iff*:
  $x \sqcap y \leq -z \longleftrightarrow --x \sqcap y \leq -z$
  ⟨*proof*⟩

lemma *p-inf-pp* [*simp*]:
  $-(x \sqcap --y) = -(x \sqcap y)$
  ⟨*proof*⟩

lemma *p-inf-pp-pp* [*simp*]:
  $-(--x \sqcap --y) = -(x \sqcap y)$
  ⟨*proof*⟩

lemma *regular-closed-inf*:
  $regular\ x \Longrightarrow regular\ y \Longrightarrow regular\ (x \sqcap y)$
  ⟨*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*:
  $regular\ top$

$\langle proof \rangle$

lemma *pp-dist-inf* [*simp*]:
$--(x \sqcap y) = --x \sqcap --y$
$\langle proof \rangle$

lemma *inf-import-p* [*simp*]:
$x \sqcap -(x \sqcap y) = x \sqcap -y$
$\langle proof \rangle$

Pseudocomplements are unique.

lemma *p-unique*:
$(\forall x . x \sqcap y = bot \longleftrightarrow x \leq z) \Longrightarrow z = -y$
$\langle proof \rangle$

lemma *maddux-3-5*:
$x \sqcup x = x \sqcup -(y \sqcup -y)$
$\langle proof \rangle$

lemma *shunting-1-pp*:
$x \leq --y \longleftrightarrow x \sqcap -y = bot$
$\langle proof \rangle$

lemma *pp-pp-inf-bot-iff*:
$x \sqcap y = bot \longleftrightarrow --x \sqcap --y = bot$
$\langle proof \rangle$

lemma *inf-pp-semi-commute*:
$x \sqcap --y \leq --(x \sqcap y)$
$\langle proof \rangle$

lemma *inf-pp-commute*:
$--(--x \sqcap y) = --x \sqcap --y$
$\langle proof \rangle$

lemma *sup-pp-semi-commute*:
$x \sqcup --y \leq --(x \sqcup y)$
$\langle proof \rangle$

lemma *regular-sup*:
$regular\ z \Longrightarrow (x \leq z \wedge y \leq z \longleftrightarrow --(x \sqcup y) \leq z)$
$\langle proof \rangle$

lemma *dense-closed-inf*:
$dense\ x \Longrightarrow dense\ y \Longrightarrow dense\ (x \sqcap y)$
$\langle proof \rangle$

lemma *dense-closed-sup*:
$dense\ x \Longrightarrow dense\ y \Longrightarrow dense\ (x \sqcup y)$

⟨*proof*⟩

lemma *dense-closed-pp*:
dense $x \Longrightarrow$ *dense* $(--x)$
⟨*proof*⟩

lemma *dense-closed-top*:
dense top
⟨*proof*⟩

lemma *dense-up-closed*:
dense $x \Longrightarrow x \leq y \Longrightarrow$ *dense* y
⟨*proof*⟩

lemma *regular-dense-top*:
regular $x \Longrightarrow$ *dense* $x \Longrightarrow x =$ *top*
⟨*proof*⟩

lemma *selection-char*:
selection $s\ x \longleftrightarrow (\exists\, y\ .\ s = -y \sqcap x)$
⟨*proof*⟩

lemma *selection-closed-inf*:
selection $s\ x \Longrightarrow$ *selection* $t\ x \Longrightarrow$ *selection* $(s \sqcap t)\ x$
⟨*proof*⟩

lemma *selection-closed-pp*:
regular $x \Longrightarrow$ *selection* $s\ x \Longrightarrow$ *selection* $(--s)\ x$
⟨*proof*⟩

lemma *selection-closed-bot*:
selection bot x
⟨*proof*⟩

lemma *selection-closed-id*:
selection $x\ x$
⟨*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 $f\ g \Longrightarrow$ *conjugate* $g\ f$
⟨*proof*⟩

lemma *additive-isotone*:
$additive\ f \Longrightarrow isotone\ f$
⟨*proof*⟩

lemma *dual-additive-antitone*:
assumes $dual\text{-}additive\ f$
shows $isotone\ (uminus \circ f)$
⟨*proof*⟩

lemma *conjugate-dual-additive*:
assumes $conjugate\ f\ g$
shows $dual\text{-}additive\ (uminus \circ f)$
⟨*proof*⟩

lemma *conjugate-isotone-pp*:
$conjugate\ f\ g \Longrightarrow isotone\ (uminus \circ uminus \circ f)$
⟨*proof*⟩

lemma *conjugate-char-1-pp*:
$conjugate\ f\ g \longleftrightarrow (\forall x\ y\ .\ f(x \sqcap -(g\ y)) \le --f\ x \sqcap -y \wedge g(y \sqcap -(f\ x)) \le --g\ y \sqcap -x)$
⟨*proof*⟩

lemma *conjugate-char-1-isotone*:
$conjugate\ f\ g \Longrightarrow isotone\ f \Longrightarrow isotone\ g \Longrightarrow f(x \sqcap -(g\ y)) \le f\ x \sqcap -y \wedge g(y \sqcap -(f\ x)) \le g\ y \sqcap -x$
⟨*proof*⟩

lemma *dense-lattice-char-1*:
$(\forall x\ y\ .\ x \sqcap y = bot \longrightarrow x = bot \vee y = bot) \longleftrightarrow (\forall x\ .\ x \ne bot \longrightarrow dense\ x)$
⟨*proof*⟩

lemma *dense-lattice-char-2*:
$(\forall x\ y\ .\ x \sqcap y = bot \longrightarrow x = bot \vee y = bot) \longleftrightarrow (\forall x\ .\ regular\ x \longrightarrow x = bot \vee x = top)$
⟨*proof*⟩

lemma *restrict-below-Rep-eq*:
$x \sqcap --y \le z \Longrightarrow x \sqcap y = x \sqcap z \sqcap y$
⟨*proof*⟩

end

The following class gives equational axioms for the pseudocomplement operation.

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

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*:
$x \sqcap -x = bot$
⟨*proof*⟩

subclass *p-algebra*
⟨*proof*⟩

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*:
$-x \sqcap (x \sqcup y) \leq y$
⟨*proof*⟩

lemma *pp-inf-sup-p* [*simp*]:
$--x \sqcap (x \sqcup -x) = x$
⟨*proof*⟩

lemma *complement-p*:
$x \sqcap y = bot \Longrightarrow x \sqcup y = top \Longrightarrow -x = y$
⟨*proof*⟩

lemma *complemented-regular*:
complemented $x \Longrightarrow$ *regular* x
⟨*proof*⟩

lemma *regular-inf-dense*:
$\exists y\ z\ .\ regular\ y \wedge dense\ z \wedge x = y \sqcap z$
⟨*proof*⟩

lemma *maddux-3-12* [*simp*]:
$(x \sqcup -y) \sqcap (x \sqcup y) = x$
⟨*proof*⟩

lemma *maddux-3-13* [*simp*]:
$(x \sqcup y) \sqcap -x = y \sqcap -x$
⟨*proof*⟩

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*⟩

lemma *order-char-1*:
$x \leq y \longleftrightarrow x \leq y \sqcup -x$
⟨*proof*⟩

lemma *order-char-2*:
$x \leq y \longleftrightarrow x \sqcup -x \leq y \sqcup -x$
⟨*proof*⟩

lemma *half-shunting*:
$x \leq y \sqcup z \Longrightarrow x \sqcap -z \leq y$
⟨*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*]:
$--(x \sqcup y) = --x \sqcup --y$
⟨*proof*⟩

lemma *regular-closed-sup*:
regular $x \Longrightarrow$ *regular* $y \Longrightarrow$ *regular* $(x \sqcup y)$
⟨*proof*⟩

The regular elements are precisely the ones having a complement.

lemma *regular-complemented-iff*:
regular $x \longleftrightarrow$ *complemented* x
⟨*proof*⟩

lemma *selection-closed-sup*:

$selection\ s\ x \Longrightarrow selection\ t\ x \Longrightarrow selection\ (s \sqcup t)\ x$
⟨proof⟩

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

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

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

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

lemma *compl-inter-eq-pp*:
$--x \sqcap y = --x \sqcap z \Longrightarrow -x \sqcap y = -x \sqcap z \Longrightarrow y = z$
⟨proof⟩

lemma *maddux-3-21-pp* [*simp*]:
$--x \sqcup (-x \sqcap y) = --x \sqcup y$
⟨proof⟩

lemma *shunting-2-pp*:
$x \leq --y \longleftrightarrow -x \sqcup --y = top$
⟨proof⟩

lemma *shunting-p*:
$x \sqcap y \leq -z \longleftrightarrow x \leq -z \sqcup -y$
⟨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*:
$x \sqcap -y \leq z \longleftrightarrow x \leq z \sqcup --y$
⟨proof⟩

lemma *conjugate-char-2-pp*:
$conjugate\ f\ g \longleftrightarrow f\ bot = bot \wedge g\ bot = bot \wedge (\forall x\ y\ .\ f\ x \sqcap y \leq --(f(x \sqcap --(g\ y))) \wedge g\ y \sqcap x \leq --(g(y \sqcap --(f\ x))))$
⟨proof⟩

lemma *conjugate-char-2-pp-additive*:
assumes *conjugate f g*

and *additive* f
and *additive* g
shows $f\,x \sqcap y \leq f(x \sqcap --(g\ y)) \wedge g\ y \sqcap x \leq g(y \sqcap --(f\ x))$
⟨*proof*⟩

end

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

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*]: $-x = (\textit{if } x = \textit{bot then top else bot})$
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*:
$\textit{regular } x \longleftrightarrow x = \textit{bot} \vee x = \textit{top}$
⟨*proof*⟩

lemma *not-bot-dense*:
$x \neq \textit{bot} \Longrightarrow --x = \textit{top}$
⟨*proof*⟩

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 $z = \textit{bot}$ 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** ‹$\rightsquigarrow$› *65*)

class *heyting-semilattice* = *semilattice-inf* + *implies* +
assumes *implies-galois*: $x \sqcap y \le z \longleftrightarrow x \le y \rightsquigarrow z$
begin

lemma *implies-below-eq* [*simp*]:
$y \sqcap (x \rightsquigarrow y) = y$
⟨*proof*⟩

lemma *implies-increasing*:
$x \le y \rightsquigarrow x$
⟨*proof*⟩

lemma *implies-galois-swap*:
$x \le y \rightsquigarrow z \longleftrightarrow y \le x \rightsquigarrow z$
⟨*proof*⟩

lemma *implies-galois-var*:
$x \sqcap y \le z \longleftrightarrow y \le x \rightsquigarrow z$
⟨*proof*⟩

lemma *implies-galois-increasing*:
$x \le y \rightsquigarrow (x \sqcap y)$
⟨*proof*⟩

lemma *implies-galois-decreasing*:
$(y \rightsquigarrow x) \sqcap y \le x$
⟨*proof*⟩

lemma *implies-mp-below*:
$x \sqcap (x \rightsquigarrow y) \le y$
⟨*proof*⟩

lemma *implies-isotone*:
$x \le y \Longrightarrow z \rightsquigarrow x \le z \rightsquigarrow y$
⟨*proof*⟩

lemma *implies-antitone*:
$x \le y \Longrightarrow y \rightsquigarrow z \le x \rightsquigarrow z$
⟨*proof*⟩

lemma *implies-isotone-inf*:
$x \rightsquigarrow (y \sqcap z) \le x \rightsquigarrow y$
⟨*proof*⟩

lemma *implies-antitone-inf*:
$x \rightsquigarrow z \le (x \sqcap y) \rightsquigarrow z$
⟨*proof*⟩

lemma *implies-curry*:
$x \rightsquigarrow (y \rightsquigarrow z) = (x \sqcap y) \rightsquigarrow z$
⟨*proof*⟩

lemma *implies-curry-flip*:
$x \rightsquigarrow (y \rightsquigarrow z) = y \rightsquigarrow (x \rightsquigarrow z)$
⟨*proof*⟩

lemma *triple-implies* [*simp*]:
$((x \rightsquigarrow y) \rightsquigarrow y) \rightsquigarrow y = x \rightsquigarrow y$
⟨*proof*⟩

lemma *implies-mp-eq* [*simp*]:
$x \sqcap (x \rightsquigarrow y) = x \sqcap y$
⟨*proof*⟩

lemma *implies-dist-implies*:
$x \rightsquigarrow (y \rightsquigarrow z) \leq (x \rightsquigarrow y) \rightsquigarrow (x \rightsquigarrow z)$
⟨*proof*⟩

lemma *implies-import-inf* [*simp*]:
$x \sqcap ((x \sqcap y) \rightsquigarrow (x \rightsquigarrow z)) = x \sqcap (y \rightsquigarrow z)$
⟨*proof*⟩

lemma *implies-dist-inf*:
$x \rightsquigarrow (y \sqcap z) = (x \rightsquigarrow y) \sqcap (x \rightsquigarrow z)$
⟨*proof*⟩

lemma *implies-itself-top*:
$y \leq x \rightsquigarrow x$
⟨*proof*⟩

lemma *inf-implies-top*:
$z \leq (x \sqcap y) \rightsquigarrow x$
⟨*proof*⟩

lemma *inf-inf-implies* [*simp*]:
$z \sqcap ((x \sqcap y) \rightsquigarrow x) = z$
⟨*proof*⟩

lemma *le-implies-top*:
$x \leq y \Longrightarrow z \leq x \rightsquigarrow y$
⟨*proof*⟩

lemma *le-iff-le-implies*:
$x \leq y \longleftrightarrow x \leq x \rightsquigarrow y$
⟨*proof*⟩

lemma *implies-inf-isotone*:
$x \leadsto y \le (x \sqcap z) \leadsto (y \sqcap z)$
⟨*proof*⟩

lemma *implies-transitive*:
$(x \leadsto y) \sqcap (y \leadsto z) \le x \leadsto z$
⟨*proof*⟩

lemma *implies-inf-absorb* [*simp*]:
$x \leadsto (x \sqcap y) = x \leadsto y$
⟨*proof*⟩

lemma *implies-implies-absorb* [*simp*]:
$x \leadsto (x \leadsto y) = x \leadsto y$
⟨*proof*⟩

lemma *implies-inf-identity*:
$(x \leadsto y) \sqcap y = y$
⟨*proof*⟩

lemma *implies-itself-same*:
$x \leadsto x = y \leadsto y$
⟨*proof*⟩

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*: $x \sqcap (x \leadsto y) \le y$
and *implies-galois-increasing*: $x \le y \leadsto (x \sqcap y)$
and *implies-isotone-inf*: $x \leadsto (y \sqcap z) \le x \leadsto y$
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*]:
$x \leadsto x = top$
⟨*proof*⟩

lemma *implies-order*:
$x \leq y \longleftrightarrow x \leadsto y = top$
⟨*proof*⟩

lemma *inf-implies* [*simp*]:
$(x \sqcap y) \leadsto x = top$
⟨*proof*⟩

lemma *top-implies* [*simp*]:
$top \leadsto x = x$
⟨*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*:
$(x \sqcup y) \sqcap (x \sqcup z) \leq x \sqcup (y \sqcap z)$
⟨*proof*⟩

subclass *distrib-lattice*
⟨*proof*⟩

lemma *implies-isotone-sup*:
$x \leadsto y \leq x \leadsto (y \sqcup z)$
⟨*proof*⟩

lemma *implies-antitone-sup*:
$(x \sqcup y) \leadsto z \leq x \leadsto z$
⟨*proof*⟩

lemma *implies-sup*:
$x \leadsto z \leq (y \leadsto z) \leadsto ((x \sqcup y) \leadsto z)$
⟨*proof*⟩

lemma *implies-dist-sup*:
$(x \sqcup y) \leadsto z = (x \leadsto z) \sqcap (y \leadsto z)$
⟨*proof*⟩

lemma *implies-antitone-isotone*:
$(x \sqcup y) \leadsto (x \sqcap y) \leq x \leadsto y$
⟨*proof*⟩

lemma *implies-antisymmetry*:

$(x \leadsto y) \sqcap (y \leadsto x) = (x \sqcup y) \leadsto (x \sqcap y)$
⟨*proof*⟩

lemma *sup-inf-implies* [*simp*]:
$(x \sqcup y) \sqcap (x \leadsto y) = y$
⟨*proof*⟩

lemma *implies-subdist-sup*:
$(x \leadsto y) \sqcup (x \leadsto z) \leq x \leadsto (y \sqcup z)$
⟨*proof*⟩

lemma *implies-subdist-inf*:
$(x \leadsto z) \sqcup (y \leadsto z) \leq (x \sqcap y) \leadsto z$
⟨*proof*⟩

lemma *implies-sup-absorb*:
$(x \leadsto y) \sqcup z \leq (x \sqcup z) \leadsto (y \sqcup z)$
⟨*proof*⟩

lemma *sup-below-implies-implies*:
$x \sqcup y \leq (x \leadsto y) \leadsto y$
⟨*proof*⟩

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*: $-x = x \leadsto$ *bot*
begin

subclass *pd-algebra*
⟨*proof*⟩

lemma *boolean-implies-below*:
$-x \sqcup y \leq x \leadsto y$
⟨*proof*⟩

lemma *negation-implies*:
$-(x \leadsto y) = --x \sqcap -y$
⟨*proof*⟩

lemma *double-negation-dist-implies*:
$--(x \leadsto y) = --x \leadsto --y$
⟨*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*: $-x = x \leadsto bot$
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.

class *heyting-stone-algebra* = *heyting-algebra* +
assumes *heyting-stone*: $-x \sqcup --x = top$
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).

class *brouwer-algebra* = *bounded-lattice* + *minus* + *uminus* +
assumes *minus-galois*: $x \le y \sqcup z \longleftrightarrow x - y \le z$
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*:
$(x - z) \sqcup (y - z) \le (x \sqcup y) - z$
⟨*proof*⟩

lemma *inf-sup-minus*:
$(x \sqcap y) \sqcup (x - y) = x$
⟨*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* = $\lambda x\ y\ .\ -x \sqcup y$
⟨*proof*⟩

subclass *brouwer-algebra*
⟨*proof*⟩

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

lemma *maddux-3-1*:
$x \sqcup -x = y \sqcup -y$
⟨*proof*⟩

lemma *maddux-3-4*:
$x \sqcup (y \sqcup -y) = z \sqcup -z$
⟨*proof*⟩

lemma *maddux-3-11* [*simp*]:
$(x \sqcap y) \sqcup (x \sqcap -y) = x$
⟨*proof*⟩

lemma *maddux-3-19*:
$(-x \sqcap y) \sqcup (x \sqcap z) = (x \sqcup y) \sqcap (-x \sqcup z)$
⟨*proof*⟩

lemma *compl-inter-eq*:
$x \sqcap y = x \sqcap z \Longrightarrow -x \sqcap y = -x \sqcap z \Longrightarrow y = z$
⟨*proof*⟩

lemma *maddux-3-21* [*simp*]:
$x \sqcup (-x \sqcap y) = x \sqcup y$
⟨*proof*⟩

lemma *shunting-1*:
$x \le y \longleftrightarrow x \sqcap -y = bot$
⟨*proof*⟩

lemma *uminus-involutive*:
$uminus \circ uminus = id$
⟨*proof*⟩

lemma *uminus-injective*:
$uminus \circ f = uminus \circ g \Longrightarrow f = g$
⟨*proof*⟩

lemma *conjugate-unique*:
$conjugate\ f\ g \Longrightarrow conjugate\ f\ h \Longrightarrow g = h$
⟨*proof*⟩

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

lemma *conjugate-additive*:
$conjugate\ f\ g \Longrightarrow additive\ f$
⟨*proof*⟩

lemma *conjugate-isotone*:
$conjugate\ f\ g \Longrightarrow isotone\ f$
⟨*proof*⟩

lemma *conjugate-char-1*:
$conjugate\ f\ g \longleftrightarrow (\forall x\ y\ .\ f(x \sqcap -(g\ y)) \leq f\ x \sqcap -y \wedge g(y \sqcap -(f\ x)) \leq g\ y \sqcap -x)$
⟨*proof*⟩

lemma *conjugate-char-2*:
$conjugate\ f\ g \longleftrightarrow f\ bot = bot \wedge g\ bot = bot \wedge (\forall x\ y\ .\ f\ x \sqcap y \leq f(x \sqcap g\ y) \wedge g\ y \sqcap x \leq g(y \sqcap f\ x))$
⟨*proof*⟩

lemma *shunting*:
$x \sqcap y \leq z \longleftrightarrow x \leq z \sqcup -y$
⟨*proof*⟩

lemma *shunting-var*:
$x \sqcap -y \leq z \longleftrightarrow x \leq z \sqcup y$
⟨*proof*⟩

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* . ∃ *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*)

abbreviation *filters* :: *'a set set*
  where *filters* ≡ { *F*::*'a set* . *filter F* }

lemma *filter-map-filter*:
assumes *filter* F
and *mono* f
and $\forall\, x\ y\ .\ f\ x \leq y \longrightarrow (\exists\, z\ .\ x \leq z \wedge y = f\ z)$
shows *filter* $(f\ `\ F)$
$\langle proof \rangle$

end

context *order*
begin

lemma *self-in-downset* [*simp*]:
$x \in {\downarrow}x$
$\langle proof \rangle$

lemma *self-in-upset* [*simp*]:
$x \in {\uparrow}x$
$\langle proof \rangle$

lemma *up-filter* [*simp*]:
filter $({\uparrow}x)$
$\langle proof \rangle$

lemma *up-set-up-set* [*simp*]:
is-up-set $({\Uparrow}X)$
$\langle proof \rangle$

lemma *up-injective*:
${\uparrow}x = {\uparrow}y \Longrightarrow x = y$
$\langle proof \rangle$

lemma *up-antitone*:
$x \leq y \longleftrightarrow {\uparrow}y \subseteq {\uparrow}x$
$\langle proof \rangle$

end

context *order-bot*
begin

lemma *bot-in-downset* [*simp*]:
$bot \in {\downarrow}x$
$\langle proof \rangle$

lemma *down-bot* [*simp*]:
${\downarrow}bot = \{bot\}$
$\langle proof \rangle$

lemma *up-bot* [*simp*]:
$\uparrow$*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 $\exists$ *G* . *ultra-filter G* $\wedge$ *F* $\subseteq$ *G*
⟨*proof*⟩

end

context *order-top*
begin

lemma *down-top* [*simp*]:
$\downarrow$*top* = *UNIV*
⟨*proof*⟩

lemma *top-in-upset* [*simp*]:
top $\in$ $\uparrow$*x*
⟨*proof*⟩

lemma *up-top* [*simp*]:
$\uparrow$*top* = {*top*}
⟨*proof*⟩

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

lemma *top-in-filter* [*simp*]:
filter F $\Longrightarrow$ *top* $\in$ *F*
⟨*proof*⟩

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*⟩

end

context *non-trivial-order-bot*
begin

lemma *ultra-filter-exists*:
$\exists F$. *ultra-filter F*
⟨*proof*⟩

end

context *non-trivial-bounded-order*
begin

lemma *proper-filter-top*:
proper-filter {*top*}
⟨*proof*⟩

lemma *ultra-filter-top*:
$\exists G$. *ultra-filter G* $\wedge$ *top* $\in G$
⟨*proof*⟩

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* $\Rightarrow$ *bool*
where *prime-filter F* $\equiv$ *proper-filter F* $\wedge$ $(\forall x\ y\ .\ x \sqcup y \in F \longrightarrow x \in F \vee y \in F)$

end

context *semilattice-inf*
begin

lemma *filter-inf-closed*:
filter F $\Longrightarrow x \in F \Longrightarrow y \in F \Longrightarrow x \sqcap y \in F$
⟨*proof*⟩

lemma *filter-univ*:
filter UNIV
⟨*proof*⟩

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

definition $\mathit{filter\text{-}sup}\ F\ G \equiv \{\ z\ .\ \exists x{\in}F\ .\ \exists y{\in}G\ .\ x \sqcap y \leq z\ \}$

lemma *filter-sup*:
  assumes *filter* F
    and *filter* G
   shows $\mathit{filter}\ (\mathit{filter\text{-}sup}\ F\ G)$
⟨*proof*⟩

lemma *filter-sup-left-upper-bound*:
  assumes *filter* G
   shows $F \subseteq \mathit{filter\text{-}sup}\ F\ G$
⟨*proof*⟩

lemma *filter-sup-symmetric*:
  $\mathit{filter\text{-}sup}\ F\ G = \mathit{filter\text{-}sup}\ G\ F$
  ⟨*proof*⟩

lemma *filter-sup-right-upper-bound*:
  $\mathit{filter}\ F \Longrightarrow G \subseteq \mathit{filter\text{-}sup}\ F\ G$
  ⟨*proof*⟩

lemma *filter-sup-least-upper-bound*:
  assumes *filter* H
    and $F \subseteq H$
    and $G \subseteq H$
   shows $\mathit{filter\text{-}sup}\ F\ G \subseteq H$
⟨*proof*⟩

lemma *filter-sup-left-isotone*:
  $G \subseteq H \Longrightarrow \mathit{filter\text{-}sup}\ G\ F \subseteq \mathit{filter\text{-}sup}\ H\ F$
  ⟨*proof*⟩

lemma *filter-sup-right-isotone*:
  $G \subseteq H \Longrightarrow \mathit{filter\text{-}sup}\ F\ G \subseteq \mathit{filter\text{-}sup}\ F\ H$
  ⟨*proof*⟩

lemma *filter-sup-right-isotone-var*:
  $\mathit{filter\text{-}sup}\ F\ (G \cap H) \subseteq \mathit{filter\text{-}sup}\ F\ H$
  ⟨*proof*⟩

lemma *up-dist-inf*:
  $\uparrow(x \sqcap y) = \mathit{filter\text{-}sup}\ (\uparrow x)\ (\uparrow y)$
⟨*proof*⟩

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

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

⟨*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* $\equiv$ *filter F* $\wedge$ *filter G* $\wedge$ *filter-sup F G* $=$ *UNIV* $\wedge$ *F* $\cap$ *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*:
$\uparrow(x \sqcup y) = \uparrow x \cap \uparrow y$
⟨*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* $\Rightarrow$ *'a filter*
where *up-filter x* $\equiv$ *Abs-filter* $(\uparrow x)$

lemma *up-filter-dist-inf*:
up-filter $((x::'a::lattice) \sqcap y)$ = *up-filter x* $\sqcup$ *up-filter y*
⟨*proof*⟩

lemma *up-filter-dist-sup*:
up-filter $((x::'a::lattice) \sqcup y)$ = *up-filter x* $\sqcap$ *up-filter y*
⟨*proof*⟩

lemma *up-filter-injective*:
up-filter x = *up-filter y* $\Longrightarrow x = y$
⟨*proof*⟩

lemma *up-filter-antitone*:
  $x \leq y \longleftrightarrow$ *up-filter* $y \leq$ *up-filter* x
  ⟨*proof*⟩

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 $F \equiv$ *Abs-filter* (f ‘ *Rep-filter* F)

lemma *filter-map-filter*:
  assumes *mono* f
    and $\forall x\ y\ .\ f\ x \leq y \longrightarrow (\exists z\ .\ x \leq z \wedge y = f\ z)$
  shows *filter* (f ‘ *Rep-filter* F)
  ⟨*proof*⟩

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 $(G \cap H) =$ *filter-sup* F $G \cap$ *filter-sup* F H
⟨*proof*⟩

lemma *filter-inf-principal-rep*:
  $F \cap G = \uparrow z \Longrightarrow (\exists x \in F\ .\ \exists y \in G\ .\ z = x \sqcup y)$
  ⟨*proof*⟩

lemma *filter-sup-principal-rep*:
  assumes *filter* F
    and *filter* G
    and *filter-sup* F $G = \uparrow z$
  shows $\exists x \in F\ .\ \exists y \in G\ .\ z = x \sqcap y$
⟨*proof*⟩

lemma *inf-sup-principal-aux*:
  assumes *filter* F
    and *filter* G
    and *is-principal-up* (*filter-sup* F G)

and *is-principal-up* $(F \cap G)$
shows *is-principal-up* F
⟨*proof*⟩

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* $(F \cap G)$
shows *is-principal-up* $F \wedge$ *is-principal-up* G
⟨*proof*⟩

lemma *filter-sup-absorb-inf*: *filter* $F \Longrightarrow$ *filter* $G \Longrightarrow$ *filter-sup* $(F \cap G)$ $G = G$
⟨*proof*⟩

lemma *filter-inf-absorb-sup*: *filter* $F \Longrightarrow$ *filter* $G \Longrightarrow$ *filter-sup* F $G \cap G = G$
⟨*proof*⟩

lemma *filter-inf-right-dist-sup*:
assumes *filter* F
and *filter* G
and *filter* H
shows *filter-sup* F $G \cap H =$ *filter-sup* $(F \cap H)$ $(G \cap H)$
⟨*proof*⟩

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

lemma *ultra-filter-prime*:
assumes *ultra-filter* F
shows *prime-filter* F
⟨*proof*⟩

lemma *up-dist-inf-inter*:
assumes *is-up-set* S
shows $\uparrow(x \sqcap y) \cap S =$ *filter-sup* $(\uparrow x \cap S)$ $(\uparrow y \cap S) \cap S$
⟨*proof*⟩

end

context *distrib-lattice-bot*
begin

lemma *prime-filter*:
proper-filter $F \Longrightarrow \exists G$. *prime-filter* $G \wedge F \subseteq G$
⟨*proof*⟩

end

context *distrib-lattice-top*
begin

lemma *complemented-filter-inf-principal*:
  assumes *filter-complements* F G
    shows *is-principal-up* $(F \cap \uparrow x)$
$\langle proof \rangle$

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
  $\langle proof \rangle$

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 φ, 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

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*

5.1 The Triple of a Stone Algebra

In this section we construct the triple associated to a Stone algebra.

5.1.1 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*]:
$\exists\, y$. *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** $\lambda x\ y\ .\ x$
$\sqcap -y$
⟨*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

5.1.2 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 $(x \sqcup -x \sqcup y) \wedge$ *dense* $(x \sqcup -x \sqcup y \sqcup z)$
⟨*proof*⟩

lemma *up-filter-dense-antitone*:
up-filter (*Abs-dense* $(x \sqcup -x \sqcup y \sqcup z)$) ≤ *up-filter* (*Abs-dense* $(x \sqcup -x \sqcup y)$)
⟨*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*⟩

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

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

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

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

instance
⟨*proof*⟩

end

5.1.3 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-base* :: *'a::stone-algebra regular* ⇒ *'a dense set*
where *stone-phi-base* $x \equiv \{\ y\ .\ -$*Rep-regular* $x \leq$ *Rep-dense* $y\ \}$

lemma *stone-phi-base-filter*:
filter (*stone-phi-base* x)
⟨*proof*⟩

definition *stone-phi* :: *'a::stone-algebra regular* ⇒ *'a dense-filter*
where *stone-phi* x = *Abs-filter* (*stone-phi-base* 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*⟩

5.2 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.

context *triple*
begin

lemma *phi-bot*:
$phi\ bot = Abs\text{-}filter\ \{top\}$
⟨*proof*⟩

lemma *phi-top*:
$phi\ top = Abs\text{-}filter\ UNIV$
⟨*proof*⟩

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

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

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

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

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

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

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

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

fun $pairs\text{-}uminus :: ('a \times 'b\ filter) \Rightarrow ('a \times 'b\ filter)$
where $pairs\text{-}uminus\ (x,y) = (-x, phi\ x)$

abbreviation $pairs\text{-}bot :: ('a \times 'b\ filter)$
where $pairs\text{-}bot \equiv (bot, Abs\text{-}filter\ UNIV)$

abbreviation $pairs\text{-}top :: ('a \times 'b\ filter)$
where $pairs\text{-}top \equiv (top, Abs\text{-}filter\ \{top\})$

lemma *pairs-top-in-set*:

$(x,y) \in pairs \Longrightarrow top \in Rep\text{-}filter\ y$
⟨*proof*⟩

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

lemma *phi-inf-principal*:
$\exists z\ .\ up\text{-}filter\ z = phi\ x \sqcap up\text{-}filter\ y$
⟨*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* :: $'a \Rightarrow 'b \Rightarrow 'b$
where $rho\ x\ y = (SOME\ z\ .\ up\text{-}filter\ z = phi\ x \sqcap up\text{-}filter\ y)$

lemma *rho-char*:
$up\text{-}filter\ (rho\ x\ y) = phi\ x \sqcap up\text{-}filter\ y$
⟨*proof*⟩

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

lemma *pairs-sup-closed*:
assumes $(x,y) \in pairs$
and $(z,w) \in pairs$
shows $pairs\text{-}sup\ (x,y)\ (z,w) \in pairs$
⟨*proof*⟩

lemma *pairs-inf-closed*:
assumes $(x,y) \in pairs$
and $(z,w) \in pairs$
shows $pairs\text{-}inf\ (x,y)\ (z,w) \in pairs$
⟨*proof*⟩

lemma *pairs-uminus-closed*:
$pairs\text{-}uminus\ (x,y) \in pairs$
⟨*proof*⟩

lemma *pairs-bot-closed*:
$pairs\text{-}bot \in pairs$
⟨*proof*⟩

lemma *pairs-top-closed*:
$pairs\text{-}top \in pairs$
⟨*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*:
$(x,y) \in pairs \Longrightarrow (z,w) \in pairs \Longrightarrow (u,v) \in pairs \Longrightarrow pairs\text{-}sup\ (x,y)\ (pairs\text{-}inf\ (z,w)\ (u,v)) = pairs\text{-}inf\ (pairs\text{-}sup\ (x,y)\ (z,w))\ (pairs\text{-}sup\ (x,y)\ (u,v))$
⟨*proof*⟩

lemma *pairs-phi-less-eq*:
$(x,y) \in pairs \Longrightarrow phi\ (-x) \le y$
⟨*proof*⟩

lemma *pairs-uminus-galois*:
assumes $(x,y) \in pairs$
and $(z,w) \in pairs$
shows $pairs\text{-}inf\ (x,y)\ (z,w) = pairs\text{-}bot \longleftrightarrow pairs\text{-}less\text{-}eq\ (x,y)\ (pairs\text{-}uminus\ (z,w))$
⟨*proof*⟩

lemma *pairs-stone*:
$(x,y) \in pairs \Longrightarrow pairs\text{-}sup\ (pairs\text{-}uminus\ (x,y))\ (pairs\text{-}uminus\ (pairs\text{-}uminus\ (x,y))) = pairs\text{-}top$
⟨*proof*⟩

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

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

lemma *dense-pairs*:
$dense\text{-}pairs = \{\ (x,y)\ .\ x = top \wedge is\text{-}principal\text{-}up\text{-}filter\ y\ \}$
⟨*proof*⟩

lemma *regular-pairs*:
$regular\text{-}pairs = \{\ (x,y)\ .\ y = phi\ (-x)\ \}$
⟨*proof*⟩

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

fun $rho\text{-}pair :: 'a \times 'b\ filter \Rightarrow 'b$
where $rho\text{-}pair\ (x,y) = (SOME\ z\ .\ up\text{-}filter\ z = phi\ x \sqcap y)$

lemma *get-rho-pair-char*:
assumes $(x,y) \in pairs$
shows $up\text{-}filter\ (rho\text{-}pair\ (x,y)) = phi\ x \sqcap y$
⟨*proof*⟩

lemma *sa-iso-pair*:
$(--x,phi\ (-x) \sqcup up\text{-}filter\ y) \in pairs$

⟨*proof*⟩

end

5.3 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**) ('*a*,'*b*) *phi* = { *f*::'*a*::*non-trivial-boolean-algebra* ⇒ '*b*::*distrib-lattice-top filter* . *bounded-lattice-homomorphism f* }
⟨*proof*⟩

lemma *simp-phi* [*simp*]:
bounded-lattice-homomorphism (*Rep-phi x*)
⟨*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 *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* . ∀ *f* . *pf f* ∈ *triple.pairs* (*Rep-phi f*) }
⟨*proof*⟩

lemma *simp-lifted-pair* [*simp*]:
∀ *f* . *Rep-lifted-pair pf f* ∈ *triple.pairs* (*Rep-phi f*)
⟨*proof*⟩

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* ⇒ ('*a*,'*b*) *lifted-pair* ⇒ ('*a*,'*b*) *lifted-pair* **is** λ*xf yf f* . *triple.pairs-sup* (*xf f*) (*yf f*)
⟨*proof*⟩

lift-definition *inf-lifted-pair* :: ('*a*,'*b*) *lifted-pair* ⇒ ('*a*,'*b*) *lifted-pair* ⇒ ('*a*,'*b*) *lifted-pair* **is** λ*xf yf f* . *triple.pairs-inf* (*xf f*) (*yf f*)
⟨*proof*⟩

lift-definition *uminus-lifted-pair* :: ('*a*,'*b*) *lifted-pair* ⇒ ('*a*,'*b*) *lifted-pair* **is** λ*xf f* . *triple.pairs-uminus* (*Rep-phi f*) (*xf f*)
⟨*proof*⟩

lift-definition *bot-lifted-pair* :: ('*a*,'*b*) *lifted-pair* **is** λ*f* . *triple.pairs-bot*
⟨*proof*⟩

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

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

instance
⟨*proof*⟩

end

5.4 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*
⟨*proof*⟩

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

lift-definition *less-eq-stone-phi-pair* :: *'a stone-phi-pair* ⇒ *'a stone-phi-pair* ⇒ *bool* **is** *triple.pairs-less-eq* ⟨*proof*⟩

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

instance ⟨*proof*⟩

end

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.

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* (λ*f* . *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.

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-phi f*) (*Rep-stone-phi-pair x*))) ∈
triple.pairs (*Rep-phi f*)
⟨*proof*⟩

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.

lemma *stone-phi-embed-homomorphism*:
sup-inf-top-bot-uminus-ord-homomorphism stone-phi-embed
⟨*proof*⟩

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*:
inj stone-phi-embed
⟨*proof*⟩

lemma *stone-phi-embed-order-injective*:
assumes *stone-phi-embed* $x \leq$ *stone-phi-embed* y
shows $x \leq y$
⟨*proof*⟩

lemma *stone-phi-embed-strict-order-isomorphism*:
$x < y \longleftrightarrow$ *stone-phi-embed* $x <$ *stone-phi-embed* y
⟨*proof*⟩

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
⟨*proof*⟩

end

5.5 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*
where *sa-iso-inv* ≡ λp . *Rep-regular* (*fst* (*Rep-stone-phi-pair* *p*)) ⊓ *Rep-dense* (*triple.rho-pair stone-phi* (*Rep-stone-phi-pair* *p*))

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* (x ⊔ $-x$))) ∈ *triple.pairs stone-phi*
⟨*proof*⟩

lemma *stone-phi-inf-dense*:
stone-phi (*Abs-regular* ($-x$)) ⊓ *up-filter* (*Abs-dense* (y ⊔ $-y$)) ≤ *up-filter* (*Abs-dense* (y ⊔ $-y$ ⊔ x))
⟨*proof*⟩

lemma *stone-phi-complement*:
complement (*stone-phi* (*Abs-regular* ($-x$))) (*stone-phi* (*Abs-regular* ($--x$)))
⟨*proof*⟩

lemma *up-dense-stone-phi*:
up-filter (*Abs-dense* (x ⊔ $-x$)) ≤ *stone-phi* (*Abs-regular* ($--x$))
⟨*proof*⟩

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

lemma *sa-iso-left-invertible*:
sa-iso-inv (*sa-iso* x) = x
⟨*proof*⟩

lemma *sa-iso-right-invertible*:
sa-iso (*sa-iso-inv* p) = p
⟨*proof*⟩

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

lemma *sa-iso*:
stone-algebra-isomorphism sa-iso
⟨*proof*⟩

5.6 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.

5.6.1 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* }
⟨*proof*⟩

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* ⇒ ('a,'b) *lifted-boolean-algebra* **is** λ*xf yf f* . *sup* (*xf f*) (*yf f*) ⟨*proof*⟩

lift-definition *inf-lifted-boolean-algebra* :: ('a,'b) *lifted-boolean-algebra* ⇒ ('a,'b) *lifted-boolean-algebra* ⇒ ('a,'b) *lifted-boolean-algebra* **is** λ*xf yf f* . *inf* (*xf f*) (*yf f*) ⟨*proof*⟩

lift-definition *minus-lifted-boolean-algebra* :: ('a,'b) *lifted-boolean-algebra* ⇒ ('a,'b) *lifted-boolean-algebra* ⇒ ('a,'b) *lifted-boolean-algebra* **is** λ*xf yf f* . *minus* (*xf f*) (*yf f*) ⟨*proof*⟩

lift-definition *uminus-lifted-boolean-algebra* :: ('a,'b) *lifted-boolean-algebra* ⇒ ('a,'b) *lifted-boolean-algebra* **is** λ*xf f* . *uminus* (*xf f*) ⟨*proof*⟩

lift-definition *bot-lifted-boolean-algebra* :: ('a,'b) *lifted-boolean-algebra* **is** λ*f* . *bot* ⟨*proof*⟩

lift-definition *top-lifted-boolean-algebra* :: ('a,'b) *lifted-boolean-algebra* **is** λ*f* . *top* ⟨*proof*⟩

lift-definition *less-eq-lifted-boolean-algebra* :: ('a,'b) *lifted-boolean-algebra* ⇒ ('a,'b) *lifted-boolean-algebra* ⇒ *bool* **is** λ*xf yf* . ∀ *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** λ*xf yf* . (∀ *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* ≡ λ*xf* . *Abs-regular* (*Abs-lifted-pair* (λ*f* . (*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* ≡ λ*pf* . *Abs-lifted-boolean-algebra* (λ*f* . *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*⟩

5.6.2 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* $\Rightarrow$ $('a,'b)$ *lifted-distrib-lattice-top* $\Rightarrow$ $('a,'b)$ *lifted-distrib-lattice-top* **is** $\lambda xf\ yf\ f\ .\ sup\ (xf\ f)\ (yf\ f)$ ⟨*proof*⟩

lift-definition *inf-lifted-distrib-lattice-top* :: $('a,'b)$ *lifted-distrib-lattice-top* $\Rightarrow$ $('a,'b)$ *lifted-distrib-lattice-top* $\Rightarrow$ $('a,'b)$ *lifted-distrib-lattice-top* **is** $\lambda xf\ yf\ f\ .\ inf\ (xf\ f)\ (yf\ f)$ ⟨*proof*⟩

lift-definition *top-lifted-distrib-lattice-top* :: $('a,'b)$ *lifted-distrib-lattice-top* **is** $\lambda f\ .\ top$ ⟨*proof*⟩

lift-definition *less-eq-lifted-distrib-lattice-top* :: $('a,'b)$ *lifted-distrib-lattice-top* $\Rightarrow$ $('a,'b)$ *lifted-distrib-lattice-top* $\Rightarrow$ *bool* **is** $\lambda xf\ yf\ .\ \forall f\ .\ less\text{-}eq\ (xf\ f)\ (yf\ f)$ ⟨*proof*⟩

lift-definition *less-lifted-distrib-lattice-top* :: $('a,'b)$ *lifted-distrib-lattice-top* $\Rightarrow$ $('a,'b)$ *lifted-distrib-lattice-top* $\Rightarrow$ *bool* **is** $\lambda xf\ yf\ .\ (\forall f\ .\ less\text{-}eq\ (xf\ f)\ (yf\ f)) \wedge \neg (\forall f\ .\ less\text{-}eq\ (yf\ f)\ (xf\ f))$ ⟨*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* :: $('a::non\text{-}trivial\text{-}boolean\text{-}algebra,'b::distrib\text{-}lattice\text{-}top)$ *lifted-pair dense* $\Rightarrow$ $('a,'b)$ *phi* $\Rightarrow$ $'b$
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* :: $('a::non\text{-}trivial\text{-}boolean\text{-}algebra,'b::distrib\text{-}lattice\text{-}top)$ *lifted-distrib-lattice-top* $\Rightarrow$ $('a,'b)$ *lifted-pair dense*
where *dl-iso-inv* $\equiv$ λxf . *Abs-dense* (*Abs-lifted-pair* (λf . (*top*,*up-filter* (*Rep-lifted-distrib-lattice-top xf f*))))

abbreviation *dl-iso* :: $('a::non\text{-}trivial\text{-}boolean\text{-}algebra,'b::distrib\text{-}lattice\text{-}top)$ *lifted-pair dense* $\Rightarrow$ $('a,'b)$ *lifted-distrib-lattice-top*
where *dl-iso* $\equiv$ 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*)) $\in$ *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*:
bounded-lattice-top-isomorphism dl-iso
⟨*proof*⟩

5.6.3 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 ((λ*qf*::('*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*⟩

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 *stone-phi* to the pair; then apply the distributive lattice isomorphism lifted to the resulting filter.

lemma *phi-iso*:
Rep-phi f (*Rep-lifted-boolean-algebra* (*ba-iso pf*) *f*) = *filter-map*
(λ*qf*::('*a*::*non-trivial-boolean-algebra*,'*b*::*distrib-lattice-top*) *lifted-pair dense* .
Rep-lifted-distrib-lattice-top (*dl-iso qf*) *f*) (*stone-phi pf*)
⟨*proof*⟩

end

References

[1] A. Armstrong, S. Foster, G. Struth, and T. Weber. Relation algebra. *Archive of Formal Proofs*, 2016, first version 2014.

[2] A. Armstrong, V. B. F. Gomes, and G. Struth. Kleene algebra with tests and demonic refinement algebras. *Archive of Formal Proofs*, 2016, first version 2014.

[3] A. Armstrong, V. B. F. Gomes, G. Struth, and T. Weber. Kleene algebra. *Archive of Formal Proofs*, 2016, first version 2013.

[4] R. Balbes and P. Dwinger. *Distributive Lattices*. University of Missouri Press, 1974.

[5] G. Birkhoff. *Lattice Theory*, volume XXV of *Colloquium Publications*. American Mathematical Society, third edition, 1967.

[6] T. S. Blyth. *Lattices and Ordered Algebraic Structures*. Springer, 2005.

[7] C. C. Chen and G. Grätzer. Stone lattices. I: Construction theorems. *Canadian Journal of Mathematics*, 21:884–894, 1969.

[8] H. B. Curry. *Foundations of Mathematical Logic*. Dover Publications, 1977.

[9] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, second edition, 2002.

[10] J. Divasón and J. Aransay. Echelon form. *Archive of Formal Proofs*, 2016, first version 2015.

[11] S. Foster and G. Struth. Regular algebras. *Archive of Formal Proofs*, 2016, first version 2014.

[12] S. Foster, G. Struth, and T. Weber. Automated engineering of relational and algebraic methods in Isabelle/HOL. In H. de Swart, editor, *Relational and Algebraic Methods in Computer Science*, volume 6663 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 2011.

[13] H. Furusawa and G. Struth. Binary multirelations. *Archive of Formal Proofs*, 2016, first version 2015.

[14] G. Georgescu, L. Leustean, and V. Preoteasa. Pseudo-hoops. *Archive of Formal Proofs*, 2016, first version 2011.

[15] V. B. F. Gomes, W. Guttmann, P. Höfner, G. Struth, and T. Weber. Kleene algebras with domain. *Archive of Formal Proofs*, 2016.

[16] V. B. F. Gomes and G. Struth. Residuated lattices. *Archive of Formal Proofs*, 2016, first version 2015.

[17] G. Grätzer. *Lattice Theory: First Concepts and Distributive Lattices*. W. H. Freeman and Co., 1971.

[18] G. Grätzer and E. T. Schmidt. On ideal theory for lattices. *Acta Scientiarium Mathematicarum*, 19(1–2):82–92, 1958.

[19] W. Guttmann. Isabelle/HOL theories of algebras for iteration, infinite executions and correctness of sequential computations. Technical Report TR-COSC 02/15, University of Canterbury, 2015.

[20] W. Guttmann. Relation-algebraic verification of Prim's minimum spanning tree algorithm. In A. Sampaio and F. Wang, editors, *Theoretical Aspects of Computing – ICTAC 2016*, volume 9965 of *Lecture Notes in Computer Science*, pages 51–68. Springer, 2016.

[21] T. Katriňák. A new proof of the construction theorem for Stone algebras. *Proceedings of the American Mathematical Society*, 40(1):75–78, 1973.

[22] G. Klein, R. Kolanski, and A. Boyton. Separation algebra. *Archive of Formal Proofs*, 2016, first version 2012.

[23] R. D. Maddux. Relation-algebraic semantics. *Theoretical Comput. Sci.*, 160(1–2):1–85, 1996.

[24] V. Preoteasa. Algebra of monotonic Boolean transformers. *Archive of Formal Proofs*, 2016, first version 2011.

[25] V. Preoteasa. Lattice properties. *Archive of Formal Proofs*, 2016, first version 2011.

[26] M. Wampler-Doty. A complete proof of the Robbins conjecture. *Archive of Formal Proofs*, 2016, first version 2010.